NAME

Padre::Wx::Main − The main window for the Padre IDE

DESCRIPTION

`Padre::Wx::Main` implements Padre's main window. It is the window containing the menus, the notebook with all opened tabs, the various sub− windows (outline, subs, output, errors, etc).

It inherits from `Wx::Frame`, so check Wx documentation to see all the available methods that can be applied to it besides the added ones (see below).

PUBLIC API

Constructor

There's only one constructor for this class.

new

```
my $main = Padre::Wx::Main−>new($ide);
```

Create and return a new Padre main window. One should pass a `Padre` object as argument, to get a reference to the Padre application.

Accessors

The following methods access the object attributes. They are both getters and setters, depending on whether you provide them with an argument. Use them wisely.

Accessors to GUI elements:

- `title`
- `config`
- `aui`
- `menu`
- `notebook`
- `left`
- `right`
- `functions`
- `todo`
- `outline`
- `directory`
- `bottom`
- `output`
- `syntax`
- `errorlist`

Accessors to operating data:

- `cwd`

Accessors that may not belong to this class:

- `ack`

find

```
my $find = $main−>find;
```

Returns the find dialog, creating a new one if needed.

fast_find

```
my $find = $main−>fast_find;
```

Return current quick find dialog. Create a new one if needed.

replace

```
my $replace = $main->replace;
```

Return current replace dialog. Create a new one if needed.

Public Methods

load_files

```
$main->load_files;
```

Load any default files: session from command-line, explicit list on command− line, or last session if user has this setup, a new file, or nothing.

`lock`

```
my $lock = $main->lock('UPDATE', 'BUSY', 'refresh_toolbar');
```

Create and return a guard object that holds resource locks of various types.

The method takes a parameter list of the locks you wish to exist for the current scope. Special types of locks are provided in capitals, refresh/method locks are provided in lowercase.

The `UPDATE` lock creates a Wx repaint lock using the built in Wx::WindowUpdateLocker class.

You should use an update lock during GUI construction/modification to prevent screen flicker. As a side effect of not updating, the GUI changes happen **significantly** faster as well. Update locks should only be held for short periods of time, as the operating system will begin to treat your\ application as ‘‘hung’’ if an update lock persists for more than a few seconds. In this situation, you may begin to see GUI corruption.

The `BUSY` lock creates a Wx ‘‘busy’’ lock using the built in Wx::WindowDisabler class.

You should use a busy lock during times when Padre has to do a long and/or complex operation in the foreground, or when you wish to disable use of any user interface elements until a background thread is finished.

Busy locks can be held for long periods of time, however your users may start to suspect trouble if you do not provide any periodic feedback to them.

Lowercase lock names are used to delay the firing of methods that will themselves generate GUI events you may want to delay until you are sure you want to rebuild the GUI.

For example, opening a file will require a Padre::Wx::Main refresh call, which will itself generate more refresh calls on the directory browser, the function list, output window, menus, and so on.

But if you open more than one file at a time, you don’t want to refresh the menu for the first file, only to do it again on the second, third and fourth files.

By creating refresh locks in the top level operation, you allow the lower level operation to make requests for parts of the GUI to be refreshed, but have the actual refresh actions delayed until the lock expires.

This should make operations with a high GUI intensity both simpler and faster.

The name of the lowercase MUST be the name of a Padre::Wx::Main method, which will be fired (with no parameters) when the method lock expires.

locked

This method provides the ability to check if a resource is currently locked.

Single Instance Server

Padre embeds a small network server to handle single instance. Here are the methods that allow to control this embedded server.

single_instance_start

```
$main->single_instance_start;
```

Start the embedded server. Create it if it doesn’t exist. Return true on success, die otherwise.

single_instance_stop

```
$main->single_instance_stop;
```

Stop & destroy the embedded server if it was running. Return true on success.

single_instance_running

```
my $is_running = $main->single_instance_running;
```

Return true if the embedded server is currently running.

`single_instance_connect`

```
$main->single_instance_connect;
```

Callback called when a client is connecting to embedded server. This is the case when user starts a new Padre, and preference "open all documents in single Padre instance" is checked.

`single_instance_command`

```
$main->single_instance_command( $line );
```

Callback called when a client has issued a command `$line` while connected on embedded server. Current supported commands are `open $file` and `focus`.

Window Methods

Those methods allow to query properties about the main window.

`window_width`

```
my $width = $main->window_width;
```

Return the main window width.

`window_height`

```
my $width = $main->window_height;
```

Return the main window height.

`window_left`

```
my $left = $main->window_left;
```

Return the main window position from the left of the screen.

`window_top`

```
my $top = $main->window_top;
```

Return the main window position from the top of the screen.

Refresh Methods

Those methods refresh parts of Padre main window. The term `refresh` and the following methods are reserved for fast, blocking, real-time updates to the GUI, implying rapid changes.

`refresh`

```
$main->refresh;
```

Force refresh of all elements of Padre main window. (see below for individual refresh methods)

`add_refresh_listener`

Adds an object which will have its `->refresh()` method called whenever the main refresh event is triggered. The refresh listener is stored as a weak reference so make sure that you keep the listener alive elsewhere.

If your object does not have a `->refresh()` method, pass in a code reference − it will be called instead.

Note that this method must return really quick. If you plan to do work that takes longer, launch it via the Action::Queue mechanism and perform it in the background.

`refresh_title`

Sets or updates the Window title.

`refresh_syntaxcheck`

```
$main->refresh_syntaxcheck;
```

Do a refresh of document syntax checking. This is a "rapid" change, since actual syntax check is happening in the background.

`refresh_outline`

```
$main->refresh_outline;
```

Force a refresh of the outline panel.

`refresh_menu`

```
$main->refresh_menu;
```

Force a refresh of all menus. It can enable / disable menu entries depending on current document or Padre internal state.

`refresh_menu_plugins`

```
$main->refresh_menu_plugins;
```

Force a refresh of just the plug-in menus.

`refresh_windowlist`

```
$main->refresh_windowlist
```

Force a refresh of the list of windows in the window menu

`refresh_recent`

Specifically refresh the Recent Files entries in the File dialog

`refresh_toolbar`

```
$main->refresh_toolbar;
```

Force a refresh of Padre's toolbar.

`refresh_status`

```
$main->refresh_status;
```

Force a refresh of Padre's status bar.

`refresh_cursorpos`

```
$main->refresh_cursorpos;
```

Force a refresh of the position of the cursor on Padre's status bar.

`refresh_functions`

```
$main->refresh_functions;
```

Force a refresh of the function list on the right.

`refresh_directory`

Force a refresh of the directory tree

`refresh_aui`

This is a refresh method wrapper around the `AUI Update` method so that it can be lock-managed by the existing locking system.

Interface Rebuilding Methods

Those methods reconfigure Padre's main window in case of drastic changes (locale, etc.)

`change_style`

```
$main->change_style( $style, $private );
```

Apply `$style` to Padre main window. `$private` is a Boolean true if the style is located in user's private Padre directory.

`change_locale`

```
$main->change_locale( $locale );
```

Change Padre's locale to `$locale`. This will update the GUI to reflect the new locale.

`relocale`

```
$main->relocale;
```

The term and method `relocale` is reserved for functionality intended to run when the application wishes to change locale (and wishes to do so without restarting).

Note at this point, that the new locale has already been fixed, and this method is usually called by `change_locale()`.

`reconfig`

```
$main->reconfig( $config );
```

The term and method `reconfig` is reserved for functionality intended to run when Padre's underlying configuration is updated by an external actor at run-time. The primary use cases for this method are when the user configuration file is synced from a remote network location.

Note: This method is highly experimental and subject to change.

`rebuild_toolbar`

```
$main->rebuild_toolbar;
```

Destroy and rebuild the toolbar. This method is useful because the toolbar is not really flexible, and most of the time it's better to recreate it from scratch.

Tools and Dialogs

Those methods deal with the various panels that Padre provides, and allow to show or hide them.

`show_functions`

```
$main->show_functions( $visible );
```

Show the functions panel on the right if `$visible` is true. Hide it otherwise. If `$visible` is not provided, the method defaults to show the panel.

`show_todo`

```
$main->show_todo( $visible );
```

Show the *to do* panel on the right if `$visible` is true. Hide it otherwise. If `$visible` is not provided, the method defaults to show the panel.

`show_outline`

```
$main->show_outline( $visible );
```

Show the outline panel on the right if `$visible` is true. Hide it otherwise. If `$visible` is not provided, the method defaults to show the panel.

`show_debugger`

```
$main->show_debugger( $visible );
```

`show_directory`

```
$main->show_directory( $visible );
```

Show the directory panel on the right if `$visible` is true. Hide it otherwise. If `$visible` is not provided, the method defaults to show the panel.

`show_output`

```
$main->show_output( $visible );
```

Show the output panel at the bottom if `$visible` is true. Hide it otherwise. If `$visible` is not provided, the method defaults to show the panel.

`show_syntax`

```
$main->show_syntax( $visible );
```

Show the syntax panel at the bottom if `$visible` is true. Hide it otherwise. If `$visible` is not provided, the method defaults to show the panel.

Introspection

The following methods allow to poke into Padre's internals.

`current`

```
my $current = $main->current;
```

Creates a Padre::Current object for the main window, giving you quick and caching access to the current various object members.

See Padre::Current for more information.

pageids

```
my @ids = $main->pageids;
```

Return a list of all current tab ids (integers) within the notebook.

pages

```
my @pages = $main->pages;
```

Return a list of all notebook tabs. Those are the real objects, not the ids (see `pageids()` above).

editors

```
my @editors = $main->editors;
```

Return a list of all current editors. Those are the real objects, not the ids (see `pageids()` above).

Note: for now, this has the same meaning as `pages()` (see above), but this will change once we get project tabs or something else.

documents

```
my @document = $main->documents;
```

Return a list of all current documents, in the specific order they are open in the notepad.

Process Execution

The following methods run an external command, for example to evaluate current document.

on_run_command

```
$main->on_run_command;
```

Prompt the user for a command to run, then run it with `run_command()` (see below).

Note: it probably needs to be combined with `run_command()` itself.

on_run_tdd_tests

```
$main->on_run_tdd_tests;
```

Callback method, to build and then call on_run_tests

on_run_tests

```
$main->on_run_tests;
```

Callback method, to run the project tests and harness them.

on_run_this_test

```
$main->on_run_this_test;
```

Callback method, to run the currently open test through prove.

on_open_in_file_browser

```
$main->on_open_in_file_browser( $filename );
```

Opens the current `$filename` using the operating system's file browser

run_command

```
$main->run_command( $command );
```

Run `$command` and display the result in the output panel.

run_document

```
$main->run_document( $debug )
```

Run current document. If `$debug` is true, document will be run with diagnostics and various debug options.

Note: this should really be somewhere else, but can stay here for now.

Session Support

Those methods deal with Padre sessions. A session is a set of files / tabs opened, with the position within the files saved, as well as the document that has the focus.

`capture_session`

```
my @session = $main->capture_session;
```

Capture list of opened files, with information. Return a list of `Padre::DB::SessionFile` objects.

`open_session`

```
$main->open_session( $session );
```

Try to close all files, then open all files referenced in the given `$session` (a `Padre::DB::Session` object). No return value.

`save_session`

```
$main->save_session( $session, @session );
```

Try to save `@session` files (`Padre::DB::SessionFile` objects, such as what is returned by `capture_session()` − see above) to database, associated to `$session`. Note that `$session` should already exist.

User Interaction

Various methods to help send information to user.

`message`

```
$main->message( $msg, $title );
```

Open a dialog box with `$msg` as main text and `$title` (title defaults to `Message`). There's only one OK button. No return value.

`info`

```
$main->info( $msg );
```

Print a message on the status bar or within a dialog box depending on the users preferences setting. The dialog has only a OK button and there is no return value.

`error`

```
$main->error( $msg );
```

Open an error dialog box with `$msg` as main text. There's only one OK button. No return value.

`prompt`

```
my $value = $main->prompt( $title, $subtitle, $key );
```

Prompt user with a dialog box about the value that `$key` should have. Return this value, or `undef` if user clicked `cancel`.

Search and Replace

These methods provide the highest level abstraction for entry into the various search and replace functions and dialogs.

However, they still represent abstract logic and should NOT be tied directly to keystroke or menu events.

`search_next`

```
# Next match for a new search
$main->search_next( $search );

# Next match on current search (or show Find dialog if none)
$main->search_next;
```

Find the next match for the current search, or spawn the Find dialog.

If no files are open, silently do nothing (don't even remember the new search)

`search_previous`

```
# Previous match for a new search
$main->search_previous( $search );

# Previous match on current search (or show Find dialog if none)
$main->search_previous;
```

Find the previous match for the current search, or spawn the Find dialog.

If no files are open, do nothing.

`replace_next`

```
# Next replace for a new search
$main->replace_next( $search );

# Next replace on current search (or show Find dialog if none)
$main->replace_next;
```

Replace the next match for the current search, or spawn the Replace dialog.

If no files are open, do nothing.

`replace_all`

```
# Replace all for a new search
$main->replace_all( $search );

# Replace all for the current search (or show Replace dialog if none)
$main->replace_all;
```

Replace all matches for the current search, or spawn the Replace dialog.

If no files are open, do nothing.

General Events

Those methods are the various callbacks registered in the menus or whatever widgets Padre has.

`on_brace_matching`

```
$main->on_brace_matching;
```

Jump to brace matching current the one at current position.

`on_comment_block`

```
$main->on_comment_block;
```

Performs one of the following depending the given operation

- Uncomment or comment selected lines, depending on their current state.
- Comment out selected lines unilaterally.
- Uncomment selected lines unilaterally.

`on_autocompletion`

```
$main->on_autocompletion;
```

Try to auto complete current word being typed, depending on document type.

`on_goto`

```
$main->on_goto;
```

Prompt user for a line or character position, and jump to this line or character position in current document.

`on_close_window`

```
$main->on_close_window( $event );
```

Callback when window is about to be closed. This is our last chance to veto the `$event` close, e.g. when some files are not yet saved.

If close is confirmed, save configuration to disk. Also, capture current session to be able to restore it next time if user set Padre to open last session on start-up. Clean up all Task Manager's tasks.

`setup_editors`

```
$main->setup_editors( @files );
```

Setup (new) tabs for `@files`, and update the GUI. If `@files` is `undef`, open an empty document.

`on_new`

```
$main->on_new;
```

Create a new empty tab. No return value.

`setup_editor`

```
$main->setup_editor( $file );
```

Setup a new tab / buffer and open `$file`, then update the GUI. Recycle current buffer if there's only one empty tab currently opened. If `$file` is already opened, focus on the tab displaying it. Finally, if `$file` does not exist, create an empty file before opening it.

`create_tab`

```
my $tab = $main->create_tab;
```

Create a new tab in the notebook, and return its id (an integer).

`on_open_selection`

```
$main->on_open_selection;
```

Try to open current selection in a new tab. Different combinations are tried in order: as full path, as path relative to `cwd` (where the editor was started), as path to relative to where the current file is, if we are in a Perl file or Perl environment also try if the thing might be a name of a module and try to open it locally or from `@INC`.

No return value.

`on_open_all_recent_files`

```
$main->on_open_all_recent_files;
```

Try to open all recent files within Padre. No return value.

`on_filter_tool`

```
$main->on_filter_tool;
```

Prompt user for a command to filter the selection/document.

`on_open_url`

```
$main->on_open_url;
```

Prompt user for URL to open and open it as a new tab.

Should be merged with −>on_open or at least a browsing function should be added.

`on_open`

```
$main->on_open;
```

Prompt user for file(s) to open, and open them as new tabs. No return value.

`on_open_with_default_system_editor`

```
$main->on_open_with_default_system_editor($filename);
```

Opens `$filename` in the default system editor

`on_open_in_command_line`

```
$main->on_open_in_command_line($filename);
```

Opens a command line/shell using the working directory of `$filename`

`on_open_example`

```
$main->on_open_example;
```

Opens the examples file dialog

reload_all

```
my $success = $main->reload_all;
```

Reload all open files from disk.

reload_some

```
my $success = $main->reload_some(@pages_to_reload);
```

Reloads the given documents. Return true upon success, false otherwise.

reload_file

```
$main->reload_file;
```

Try to reload a file from disk. Display an error if something went wrong.

Returns 1 on success and 0 in case of and error.

on_reload_file

```
$main->on_reload_file;
```

Try to reload current file from disk. Display an error if something went wrong. No return value.

on_reload_all

```
$main->on_reload_all;
```

Reload all currently opened files from disk. No return value.

on_save

```
my $success = $main->on_save;
```

Try to save current document. Prompt user for a file name if document was new (see `on_save_as()` above). Return true if document has been saved, false otherwise.

on_save_as

```
my $was_saved = $main->on_save_as;
```

Prompt user for a new file name to save current document, and save it. Returns true if saved, false if cancelled.

on_save_intuition

```
my $success = $main->on_save_intuition;
```

Try to automatically determine an appropriate file name and save it, based entirely on the content of the file.

Only do this for new documents, otherwise behave like a regular save.

on_save_all

```
my $success = $main->on_save_all;
```

Try to save all opened documents. Return true if all documents were saved, false otherwise.

_save_buffer

```
my $success = $main->_save_buffer( $id );
```

Try to save buffer in tab `$id`. This is the method used underneath by all `on_save_*()` methods. It will check if document has been updated out of Padre before saving, and report an error if something went wrong. Return true if buffer was saved, false otherwise.

on_close

```
$main->on_close( $event );
```

Handler when there is a close `$event`. Veto it if it's from the `AUI` notebook, since Wx will try to close the tab no matter what. Otherwise, close current tab. No return value.

close

```
my $success = $main->close( $id );
```

Request to close document in tab `$id`, or current one if no `$id` provided. Return true if closed, false otherwise.

close_all

```
my $success = $main->close_all( $skip );
```

Try to close all documents. If `$skip` is specified (an integer), don't close the tab with this id. Return true upon success, false otherwise.

close_some

```
my $success = $main->close_some(@pages_to_close);
```

Try to close all documents. Return true upon success, false otherwise.

close_where

```
# Close all files in current project
my $project = Padre::Current->document->project_dir;
my $success = $main->close_where( sub {
    $_[0]->project_dir eq $project
} );
```

The `close_where` method is for closing multiple document windows. It takes a subroutine as a parameter and calls that subroutine for each currently open document, passing the document as the first parameter.

Any documents that return true will be closed.

on_nth_path

```
$main->on_nth_pane( $id );
```

Put focus on tab `$id` in the notebook. Return true upon success, false otherwise.

on_next_pane

```
$main->on_next_pane;
```

Put focus on tab next to current document. Currently, only left to right order is supported, but later on it can be extended to follow a last seen order.

No return value.

on_prev_pane

```
$main->on_prev_pane;
```

Put focus on tab previous to current document. Currently, only right to left order is supported, but later on it can be extended to follow a reverse last seen order.

No return value.

on_diff

```
$main->on_diff;
```

Run `Text::Diff` between current document and its last saved content on disk. This allow to see what has changed before saving. Display the differences in the output pane.

on_join_lines

```
$main->on_join_lines;
```

Join current line with next one (à la **vi** with `Ctrl+J`). No return value.

Preferences and toggle methods

Those methods allow to change Padre's preferences.

zoom

```
$main->zoom( $factor );
```

Apply zoom `$factor` to Padre's documents. Factor can be either positive or negative.

open_regex_editor

```
$main->open_regex_editor;
```

Open Padre's regular expression editor. No return value.

on_preferences

```
$main->on_preferences;
```

Open Padre's preferences dialog. No return value.

on_key_bindings

```
$main->on_key_bindings;
```

Opens the key bindings dialog

on_toggle_line_numbers

```
$main->on_toggle_line_numbers;
```

Toggle visibility of line numbers on the left of the document. No return value.

on_toggle_code_folding

```
$main->on_toggle_code_folding;
```

De/activate code folding. No return value.

on_toggle_currentline

```
$main->on_toggle_currentline;
```

Toggle background highlighting of current line. No return value.

on_toggle_right_margin

```
$main->on_toggle_right_margin;
```

Toggle display of right margin. No return value.

on_toggle_syntax_check

```
$main->on_toggle_syntax_check;
```

Toggle visibility of syntax panel. No return value.

on_toggle_errorlist

```
$main->on_toggle_errorlist;
```

Toggle visibility of error-list panel. No return value.

on_toggle_indentation_guide

```
$main->on_toggle_indentation_guide;
```

Toggle visibility of indentation guide. No return value.

on_toggle_eol

```
$main->on_toggle_eol;
```

Toggle visibility of end of line carriage returns. No return value.

on_toggle_whitespaces

```
$main->on_toggle_whitespaces;
```

Show/hide spaces and tabs (with dots and arrows respectively). No return value.

on_word_wrap

```
$main->on_word_wrap;
```

Toggle word wrapping for current document. No return value.

on_toggle_toolbar

```
$main->on_toggle_toolbar;
```

Toggle toolbar visibility. No return value.

on_toggle_statusbar

```
$main->on_toggle_statusbar;
```

Toggle status bar visibility. No return value.

on_toggle_lockinterface

```
$main->on_toggle_lockinterface;
```

Toggle possibility for user to change Padre's external aspect. No return value.

on_insert_from_file

```
$main->on_insert_from_file;
```

Prompt user for a file to be inserted at current position in current document. No return value.

convert_to

```
$main->convert_to( $eol_style );
```

Convert document to `$eol_style` line endings (can be one of `WIN`, `UNIX`, or `MAC`). No return value.

find_editor_of_file

```
my $editor = $main->find_editor_of_file( $file );
```

Return the editor (a `Padre::Wx::Editor` object) containing the wanted `$file`, or `undef` if file is not opened currently.

find_id_of_editor

```
my $id = $main->find_id_of_editor( $editor );
```

Given `$editor`, return the tab id holding it, or `undef` if it was not found.

Note: can this really work? What happens when we split a window?

run_in_padre

```
$main->run_in_padre;
```

Evaluate current document within Padre. It means it can access all of Padre's internals, and wreak havoc. Display an error message if the evaluation went wrong, dump the result in the output panel otherwise.

No return value.

`STC` related methods

Those methods are needed to have a smooth `STC` experience.

on_stc_style_needed

```
$main->on_stc_style_needed( $event );
```

Handler of `EVT_STC_STYLENEEDED` `$event`. Used to work around some edge cases in scintilla. No return value.

on_stc_update_ui

```
$main->on_stc_update_ui;
```

Handler called on every movement of the cursor. No return value.

on_stc_change

```
$main->on_stc_change;
```

Handler of the `EVT_STC_CHANGE` event. Doesn't do anything. No return value.

on_stc_char_needed

```
$main->on_stc_char_added;
```

This handler is called when a character is added. No return value. See <http://www.yellowbrain.com/stc/events.html#EVT_STC_CHARADDED>

TO DO: maybe we need to check this more carefully.

on_stc_dwell_start

```
$main->on_stc_dwell_start( $event );
```

Handler of the `DWELLSTART` `$event`. This event is sent when the mouse has not moved in a given amount of time. Doesn't do anything by now. No return value.

on_aui_pane_close

```
$main->on_aui_pane_close( $event );
```

Handler called upon `EVT_AUI_PANE_CLOSE` `$event`. Doesn't do anything by now.

on_doc_stats

```
$main->on_doc_stats;
```

Compute various stats about current document, and display them in a message. No return value.

on_tab_and_space

```
$main->on_tab_and_space( $style );
```

Convert current document from spaces to tabs (or vice-versa) depending on `$style` (can be either of `Space_to_Tab` or `Tab_to_Space`). Prompts the user for how many spaces are to be used to replace tabs (whatever the replacement direction). No return value.

on_delete_ending_space

```
$main->on_delete_ending_space;
```

Trim all ending spaces in current selection, or document if no text is selected. No return value.

on_delete_leading_space

```
$main->on_delete_leading_space;
```

Trim all leading spaces in current selection. No return value.

timer_check_overwrite

```
$main->timer_check_overwrite;
```

Called every n seconds to check if file has been overwritten outside of Padre. If that's the case, prompts the user whether s/he wants to reload the document. No return value.

on_last_visited_pane

```
$main->on_last_visited_pane;
```

Put focus on tab visited before the current one. No return value.

on_oldest_visited_pane

```
$main->on_oldest_visited_pane;
```

Put focus on tab visited the longest time ago. No return value.

on_new_from_template

```
$main->on_new_from_template( $extension );
```

Create a new document according to template for `$extension` type of file. No return value.

Auxiliary Methods

Various methods that did not fit exactly in above categories...

install_cpan

```
$main->install_cpan( $module );
```

Install `$module` from `CPAN`.

Note: this method may not belong here...

setup_bindings

```
$main->setup_bindings;
```

Setup the various bindings needed to handle output pane correctly.

Note: I'm not sure those are really needed...

key_up

```
$main->key_up( $event );
```

Callback for when a key up `$event` happens in Padre. This handles the various `Ctrl`+key combinations used within Padre.

new_document_from_string

```
$main->new_document_from_string( $string, $mimetype );
```

Create a new document in Padre with the string value.

Pass in an optional mime type to have Padre colorize the text correctly.

Note: this method may not belong here...

COPYRIGHT & LICENSE