User Interface (UI) architecture

The present paragraph is meant to help developers understand the UI source code.

One of neovim goals is to make it possible to create (n)vim GUIs (Graphical UIs) without touching the core code. This is done via UI events sent over msgpack-rpc. These UIs (nvim-qt, oni, nyaovim, ...) are called **remote UI clients**.

Yet for compatibility reasons there exists another type of UI simply called **(local?) UI clients** which are more dependant on nvim changes. For instance the terminal UI (TUI) bypasses the msgpack-rpc layer. So does vimR.

The source files most directly involved with UI events are:

1. `src/nvim/ui.*` : calls handler functions of registered UI structs (independant from msgpack-rpc)
2. `src/nvim/api/ui.*` : forwards messages over msgpack-rpc to remote UIs.
3. `src/nvim/api/ui_bridge.*` : similar to 2/ but forwards messages to UI threads (TUI for instance) as events

UI events are defined in `src/nvim/api/ui_events.in.h` , this file is not compiled directly, rather it parsed by `src/nvim/generators/gen_api_ui_events.lua` which autogenerates wrapper functions used by the source files above. It also generates metadata accesible as `api_info().ui_events` .

Please note that in both cases (local and remote), UI events may be deferred to UIs, which implies to do deepcopy of the UI events data. The difference between 2 and 3 is best seen in https://github.com/neovim/neovim/pull/6044, `ui_bridge_cursor_styleset_event` and `remote_ui_cursor_style_set` are very similar, except `remote_ui_cursor_style_set` sends the event over msgpack, while `ui_bridge_cursor_style_set` pass it on to the local UI scheduler (src/nvim/tui.c:tui_scheduler for the TUI).

Once you've finished your changes to neovim UI protocol, don't forget to update `set(NVIM_API_LEVEL 2) # Bump this after any API change.` in `CMakeLists.txt`

Here are a few more references:

- :help msgpack-rpc
- :help ui
- https://github.com/neovim/neovim/pull/3246
- http://tarruda.github.io/articles/neovim-smart-ui-protocol/