Table of contents

1. Background

Definitions

Query window: A window that is containing a text edit box. Examples in trunk: sign name edit window, save dialog, query string (used by advanced settings for example).

Key: Through this report the word *key* will not be used of the meaning “main” such as in “key point” with the meaning “main point”, to avoid confusion.

Globally focused widget: The focused widget on the focused window. (in this patch each window has a focused widget variable, hence the need for a term describing focused widget on focused window)

Aim

The aim of this patch is to give foundation for more use of text edit boxes. In current trunk only one window with a text edit box can be allowed to be open at a given time due to limitations that will be mentioned later. My motivation for this patch was to be able to add text edit boxes to windows that the user may want to keep open in combination with another window that make use of a text edit box. For example if a filter edit box is added to most list-windows then only one such window can be open at every time with the current window management. Also having a window with text input open currently makes it steal all key input, making impossible to use hotkeys.

What is it not

- The aim is not to remove the limitation of maximum one text edit box per window
- This patch does not aim to change the input model to a fully focus based input model, only when the focused widget is a text box the focused window receives all key input events. And with this patch it is still up to the window to make sure the input events goes to the text box.

2. Limitations in trunk (that are addressed by this patch)

- **No scroll flag:** A query window needs to set a flag `_no_scroll` when it is opened and unset the flag when it is closed. This flag stops the arrow keys from being used to scroll the main viewport.

  If two query windows would be allowed to be open side by side then closing one of them would cause this flag be unset and the arrow keys would be enabled for scrolling.

- **Focus:** Usually the key input routine loops through all windows and let them one after one process the key event until a window returns `ES_HANDELED`. If a query window is open however only that window will receive the event and no other window will have the chance to process the event.

  The problem with this is that a window with a text edit box is a query window and will need to be closed before a user can type hotkeys again.

  For windows with the only purpose of editing a string this is not a big problem, but it limits the use of text edit boxes as having a text box on a list window to type a filter for filtering the list.

3. Solution

Variables

- Global pointer variable `_focused_window` points out the focused window and in some cases null.
- Each window has a pointer variable `focused_widget` that points out the focused widget within that window. Also this variable may be null in some cases.

  Since a possible solution is to have a global `_focused_widget` variable instead of one per window I have created a list of arguments in favor and against per window variables:

Per window focused widget variables – arguments

Arguments in favor:

+ Query windows may want to give the text box focus when they open. With per window focus variables this can be done without assuming that the opened window has focus.
+ I think that focused widget fits in the window class better with respect to object oriented design.
+ When a window loses focus the widget that had focus in its window is still remembered and can be given focus when that window gets focus again. (For example when focused window is changed by clicking on the caption bar since the caption bar is a special case and does not receive widget focus)
+ One global variable less

Arguments against:

- Makes use of more memory
- More complex with many focused widgets, one for each window than having only one globally focused widget variable.

So the globally focused widget becomes `_focused_window->focused_widget`, given that `_focused_window` is not null.

Implementation

I will not mention at every situation `_focused_window` and `focused_widget` is checked for null as that would make this description unnecessary lengthy. If an action is told to happen then if not else written it won't happen if respective variable is null.

➔ In `DispatchLeftClickEvent` in window.cpp: When a previously not focused window is clicked, `_focused_window` is changed to point on that window and the previously focused window is informed. Also `focused_widget` on the clicked window is changed to the widget that is clicked on unless it is the caption bar.

➔ In `HandleKeypress` in window.cpp: The type of the focused widget of the focused window is checked, and if it is `WWT_TEXTBOX` the window will be treated the same way as query windows are treated in trunk (the window becomes the only receiver of key events). Else if focused widget on focused window is not a text box then all windows will be asked to process the key input until a window returns that it has processed the key input - just as in trunk.

- ➔ In `HandleKeyScrolling` in window.cpp: don’t scroll main viewport using arrow keys if globally focused widget is a text box. (trunk uses `_no_scroll` flag to determinate when not to allow scroll)

- ➔ `HandleEditBoxKey` makes sure that the edit box that it handles is globally focused before it accepts key input. A new return value has been added to inform caller that the edit box does not have focus. (the return values has also been enumified)

- ➔ `DrawTextBox` only draws a flashing caret on globally focused text boxes.

- ➔ Functions such as `Window::SetFocusedWidget` to encapsulate focus change book keeping tasks such as marking unfocused widget dirty (so caret can be removed from text boxes).

Minor changes:

- ➔ Removal of set/unset of `_no_scroll` flag. (since it is not used for anything anymore)

- ➔ Setting text input box to default focused widget in existing query windows by setting a call to `Window::SetFocusedWidget` of that window in the constructor.

- ➔ Setting of global focus to the text box of parent window if the text box of the OSK window is clicked.

- ➔ Relaxed what windows to close when a new window with edit box is opened:
  - o *When opening a sign window:* don’t close the save/load window
  - o *When opening a QueryString window:* don’t close the save/load window
  - o *When opening a save/load window:* don’t close `WC_QUERY_STRING` windows (sign + query string windows)

4. Alternative solutions

- Change query string dialog so that the parent window get string updates as the user types. And then use query string dialog to type a filter string

  Drawbacks:
  - The GUI becomes cluttered.
  - In my option less intuitive to have to click on a text label or similar to open a query string window than typing straight into a text edit box.
  - When the user hit enter the query string window closes. With a text box in the parent window the enter key can in the sign list example move the viewport to selected sign, but keep the edit active.
  - In string list filter patch I plan to use the arrow up/down keys when user edits the filter to select a sign from the displayed list.

- Dynamically allow windows to tell if they are a query window as I've done in my *dynamic query window patch*. This makes it possible for windows to in the default state have the text edit box hidden and in some other state show it and only then have the requirement of being the only query window on the screen.

  An advancement of this could even include that other windows are told to hide their edit box and become non-query windows before a close all query windows action is performed.

  Still it has some drawbacks:
  - While it allows for more windows to temporary be query windows I think that with many enough such windows there will be a problem of scalability. Starting editing at one place will close another edit somewhere else that the user did not want to have aborted.
  - It requires a trigger widget on the window to unhide the text edit box. This is might not only be sub-optimal from users/GUI perspective but also requires quite some repeated code for this procedure for all windows that make use of the trick. It can surely become more generic than my search sign list patch but still is kind of a work around.

These two alternative solutions however both has the possibility to in future be enhanced to work with multiple text edit boxes per window, as well as the solution that I have selected to implement in this patch.

5. Suggestions on what to be looked on when reviewing this patch

1. Look for "`enum HandleEditBoxResult`" in querystring_gui.h.

   Code:
   ```
   enum HandleEditBoxResult
   {
         HEBR_DEFAULT = 0, // This is returned when none of the
   below are the case. Suggestions for a better name than default'
   are welcome.
         HEBR_RETURN_KEY,
         HEBR_ESCAPE_KEY,
         HEBR_NOT_FOCUSED,
   };
   ```

   Usage: This enum is used to describe the return values from `HandleEditBoxKey` function of the `QueryString` and `QueryStringBaseWindow` classes.

   Question: The first enum constant "`HEBR_DEFAULT`" is used when no other value is returned. I don't really like the word "default" here, but can't think of a better name. Can you come up with a better name?

2. Look for "`Widget *focused_widget;`" in window_gui.h

   Usage: This variable is used to keep track of focused widget within each window.

   Type: I have decided to use `Widget*` as type since `focused_widget` need to be able to indicate no widget is focused. I've noticed that functions that take a widget "reference" do so by taking the index of the widget in the `window->widget` array. But that index is unsigned and currently there seems to be no definition of a high index that means "no widget". So therefore I've chosen to use `Widget*` instead of `unsigned int`. Also it simplifies the code a bit since it is common to check the type of the focused widget of the focused window which then becomes `_focused_window->focused_widget->type`. (given that all pointers are non-null)

3. **Summary:** Constness of `focused_widget`

   Currently `focused_widget` points on a const widget.

   <u>Arguments for defining as const</u>
   - `DispatchLeftClickEvent` becomes slightly cleaner. (a local const variable can be used to shorten code a bit)
   - The patch works fine if `focused_widget` is defined to point on a const widget. So why not be as strict as possible?

   <u>Arguments for not defining as const</u>
   - In future there might be useful to be able to modify the focused widget directly without using pointer arithmetic's[1] to find out the widget index in the widget array.

   Question: Change declaration of `focused_widget` to point on a non-const widget?

[1] `window::widget[window::focused_widget - window::widget]`