Keyboard Selection

Abstract

This spec describes a new set of non-configurable keybindings that allows the user to update a selection without the use of a mouse or stylus.

Inspiration

ConHost allows the user to modify a selection using the keyboard. Holding `Shift` allows the user to move the second selection endpoint in accordance with the arrow keys. The selection endpoint updates by one cell per key event, allowing the user to refine the selected region.

Mark mode allows the user to create a selection using only the keyboard, then edit it as mentioned above.

Solution Design

The fundamental solution design for keyboard selection is that the responsibilities between the Terminal Control and Terminal Core must be very distinct. The Terminal Control is responsible for handling user interaction and directing the Terminal Core to update the selection. The Terminal Core will need to update the selection according to the preferences of the Terminal Control.

Relatively recently, TerminalControl was split into `TerminalControl`, `ControlInteractivity`, and `ControlCore`. Changes made to `ControlInteractivity`, `ControlCore`, and below propagate functionality to all consumers, meaning that the WPF terminal would benefit from these changes with no additional work required.

Fundamental Terminal Control Changes

`ControlCore::TrySendKeyEvent()` is responsible for handling the key events after key bindings are dealt with in `TermControl`. At the time of writing this spec, there are 2 cases handled in this order: - Clear the selection (except in a few key scenarios) - Send Key Event

The first branch will be updated to *modify* the selection instead of usually *clearing* it. This will happen by converting the key event into parameters to forward to `TerminalCore`, which then updates the selection appropriately.

Idea: Make keyboard selection a collection of standard keybindings
One idea is to introduce an `updateSelection` action that conditionally works if a selection is active (similar to the `copy` action). For these key bindings, if there is no selection, the key events are forwarded to the application.

Thanks to Keybinding Args, there would only be 1 new command: | Action | Keybinding Args | Description | |–|–|–| | `updateSelection` | | If a selection

exists, moves the last selection endpoint. | | | `Enum direction { up, down, left, right }` | The direction the selection will be moved in. | | | `Enum mode { char, word, view, buffer }` | The context for which to move the selection endpoint to. (defaults to `char`) |

By default, the following keybindings will be set:

```
// Character Selection
{ "command": {"action": "updateSelection", "direction": "left",  "mode": "char" }, "keys": "
{ "command": {"action": "updateSelection", "direction": "right", "mode": "char" }, "keys": "
{ "command": {"action": "updateSelection", "direction": "up",    "mode": "char" }, "keys": "
{ "command": {"action": "updateSelection", "direction": "down",  "mode": "char" }, "keys": "

// Word Selection
{ "command": {"action": "updateSelection", "direction": "left",  "mode": "word" }, "keys": "
{ "command": {"action": "updateSelection", "direction": "right", "mode": "word" }, "keys": "

// Viewport Selection
{ "command": {"action": "updateSelection", "direction": "left",  "mode": "view" }, "keys": "
{ "command": {"action": "updateSelection", "direction": "right", "mode": "view" }, "keys": "
{ "command": {"action": "updateSelection", "direction": "up",    "mode": "view" }, "keys": "
{ "command": {"action": "updateSelection", "direction": "down",  "mode": "view" }, "keys": "

// Buffer Corner Selection
{ "command": {"action": "updateSelection", "direction": "up",    "mode": "buffer" }, "keys"
{ "command": {"action": "updateSelection", "direction": "down",  "mode": "buffer" }, "keys"
```

These are in accordance with ConHost's keyboard selection model.

This idea was abandoned due to several reasons: 1. Keyboard selection should be a standard way to interact with a terminal across all consumers (i.e. WPF control, etc.) 2. There isn't really another set of key bindings that makes sense for this. We already hardcoded ESC as a way to clear the selection. This is just an extension of that. 3. Adding 12 conditionally effective key bindings takes the spot of 12 potential non-conditional key bindings. It would be nice if a different key binding could be set when the selection is not active, but that makes the settings design much more complicated. 4. 12 new items in the command palette is also pretty excessive. 5. If proven wrong when this is in WT Preview, we can revisit this and make them customizable then. It's better to add the ability to customize it later than take it away.

Idea: Make keyboard selection a simulation of mouse selection It may seem that some effort can be saved by making the keyboard selection act as a simulation of mouse selection. There is a union of mouse and keyboard activity that can be represented in a single set of selection motion interfaces that are commanded by the TermControl's Mouse/Keyboard handler and adapted into appropriate motions in the Terminal Core.

However, the mouse handler operates by translating a pixel coordinate on the screen to a text buffer coordinate. This would have to be rewritten and the approach was deemed unworthy.

Fundamental Terminal Core Changes

The Terminal Core will need to expose a `UpdateSelection()` function that is called by the keybinding handler. The following parameters will need to be passed in: - `enum SelectionDirection`: the direction that the selection endpoint will attempt to move to. Possible values include `Up`, `Down`, `Left`, and `Right`. - `enum SelectionExpansion`: the selection expansion mode that the selection endpoint will adhere to. Possible values include `Char`, `Word`, `View`, `Buffer`.

Moving by Cell For `SelectionExpansion = Char`, the selection endpoint will be updated according to the buffer's output pattern. For **horizontal movements**, the selection endpoint will attempt to move left or right. If a viewport boundary is hit, the endpoint will wrap appropriately (i.e.: hitting the left boundary moves it to the last cell of the line above it).

For **vertical movements**, the selection endpoint will attempt to move up or down. If a **viewport boundary** is hit and there is a scroll buffer, the endpoint will move and scroll accordingly by a line.

If a **buffer boundary** is hit, the endpoint will not move. In this case, however, the event will still be considered handled.

NOTE: An important thing to handle properly in all cases is wide glyphs. The user should not be allowed to select a portion of a wide glyph; it should be all or none of it. When calling `_ExpandWideGlyphSelection` functions, the result must be saved to the endpoint.

Moving by Word For `SelectionExpansion = Word`, the selection endpoint will also be updated according to the buffer's output pattern, as above. However, the selection will be updated in accordance with "chunk selection" (performing a double-click and dragging the mouse to expand the selection). For **horizontal movements**, the selection endpoint will be updated according to the `_ExpandDoubleClickSelection` functions. The result must be saved to the endpoint. As before, if a boundary is hit, the endpoint will wrap appropriately. See Future Considerations for how this will interact with line wrapping.

For **vertical movements**, the movement is a little more complicated than before. The selection will still respond to buffer and viewport boundaries as before. If the user is trying to move up, the selection endpoint will attempt to move up by one line, then selection will be expanded leftwards. Alternatively, if the user is trying to move down, the selection endpoint will attempt to move down by one line, then the selection will be expanded rightwards.

Moving by Viewport For `SelectionExpansion = View`, the selection endpoint will be updated according to the viewport's height. Horizontal movements will be updated according to the viewport's width, thus resulting in the endpoint being moved to the left/right boundary of the viewport.

Moving by Buffer For `SelectionExpansion = Buffer`, the selection endpoint will be moved to the beginning or end of all the text within the buffer. If moving up or left, set the position to 0,0 (the origin of the buffer). If moving down or right, set the position to the last character in the buffer.

NOTE: In all cases, horizontal movements attempting to move past the left/right viewport boundaries result in a wrap. Vertical movements attempting to move past the top/bottom viewport boundaries will scroll such that the selection is at the edge of the screen. Vertical movements attempting to move past the top/bottom buffer boundaries will be clamped to be within buffer boundaries.

Every
map to a keybinding. These pairings are shown below in the UI/UX Design –> Keybindings section.

NOTE: If `copyOnSelect` is enabled, we need to make sure we **DO NOT** update the clipboard on every change in selection. The user must explicitly choose to copy the selected text from the buffer.

UI/UX Design

Key Bindings

There will only be 1 new command that needs to be added: | Action | Keybinding Args | Description | |–|–|–| | `selectAll` | | Select the entire text buffer.

By default, the following key binding will be set:

```
{ "command": "selectAll", "keys": "ctrl+shift+a" },
```

Capabilities

Accessibility

Using the keyboard is generally a more accessible experience than using the mouse. Being able to modify a selection by using the keyboard is a good first step towards making selecting text more accessible.

Security

N/A

Reliability

With regards to the Terminal Core, the newly introduced code should rely on already existing and tested code. Thus no crash-related bugs are expected.

With regards to Terminal Control and the settings model, crash-related bugs are not expected. However, ensuring that the selection is updated and cleared in general use-case scenarios must be ensured.

Compatibility

N/A

Performance, Power, and Efficiency

Potential Issues

Grapheme Clusters

When grapheme cluster support is inevitably added to the Text Buffer, moving by "cell" is expected to move by "character" or "cluster". This is similar to how wide glyphs are handled today. Either all of it is selected, or none of it.

Future considerations

Word Selection Wrap

At the time of writing this spec, expanding or moving by word is interrupted by the beginning or end of the line, regardless of the wrap flag being set. In the future, selection and the accessibility models will respect the wrap flag on the text buffer.

Mark Mode

This functionality will be expanded to create a feature similar to Mark Mode. This will allow a user to create a selection using only the keyboard.

Resources

- https://blogs.windows.com/windowsdeveloper/2014/10/07/console-improvements-in-the-windows-10-technical-preview/