Keyboard Event Handlers (Remapping Logic)

This file contains documentation for all the methods involved in key/shortcut remapping.

Table of Contents:

HandleSingleKeyRemapEvent

This method is used for handling the key to key and key to shortcut remapping logic. The general logic is as follows: - Check if the `dwExtraInfo` field contains the `KEYBOARDMANAGER_INJECTED_FLAG` bit set. This bit is used to indicate that the key event was generated by KBM using `SendInput`. This ensures that we don't read events generated by the key or shortcut remap methods. - Check if the current key is present in the list of remaps. If it isn't, return 0 (i.e. do not suppress the event). - If it is remapped to Disable, suppress the event. - If it is remapped to a key, we send the key down/up message for the target key and suppress the current key event. We have a check for filtering artificial keys, such as `VK_WIN` (which is a keycode added by us), so that it is translated to `VK_LWIN` instead. - If it is remapped to a shortcut, for key down we set the target modifiers first, followed by the target action key, and for key up we release the action key first, followed by the modifiers. - All the remapped key events that we send above are sent with `KEYBOARDMANAGER_SINGLEKEY_FLAG` on the `dwExtraInfo` field.

HandleShortcutRemapEvent

This method is used for handling the shortcut to shortcut and shortcut to key remapping logic. The general logic is as follows: - Check if any shortcut remap is currently invoked. This is required to ensure that two remaps don't occur simultaneously at a time, and we send key up events for the shortcuts only if they are actually invoked and not for artificial key up events. In addition to that, while a remap is in the middle of execution, the keyboard state will not match the physical keys, so we do not want a remap Ctrl+A to Ctrl+V to also trigger the remap from Ctrl+V to Alt+V on pressing Ctrl+A on the keyboard. - Get the remap table as per the the `activatedApp` argument (i.e. if it is empty, we get the global shortcut remap table and otherwise we get the corresponding

app-specific shortcut remap table). - Iterate over the list of remaps in descending order of number of keys in the shortcut. This is required **for shortcut to key remaps** to ensure that if a user has both Ctrl+A and Ctrl+Shift+A remapped to some keys, and the user presses Ctrl+Shift+A, then we prefer the Ctrl+Shift+A remap. This logic would not be required if there were only shortcut to shortcut remaps, as they are invoked only on exact match. - If any shortcut was found to be invoked (from the first step), then we skip till we find the matching shortcut remap. If not we check if the modifiers of the original shortcut are pressed down. If they are, we check if the current key event is a key down event and it matches the action key of the original shortcut. For shortcut to shortcut and for disabling a shortcut we have an additional step where we check if any other key is pressed apart from the original shortcut. This is required because for these two features we allow the remaps only if those exact keys are pressed. The method used for this is described in detail here. If a win key was pressed, we store whether it was the left or the right one, in order to determine which key to set for remaps from/to the common Win key code which we added. This is so that pressing and releasing Left Win key results in that Win key getting modified and not the Right Win key. - If the remap is to a key, we send a dummy key event followed by releasing the original shortcut's modifiers and setting the target key (or doing nothing if it is remapped to disable) and we suppress the event. - If the remap is to a shortcut, if the modifiers in the original shortcut are present in the target, we only set the additional modifiers and the action key of the target. If it isn't, we send a dummy key event followed by releasing the modifiers which are not common, and setting the remaining ones in the target along with the action key. - For both cases, we set the `isShortcutInvoked` flag to true, and set the `KeyboardManagerState.activatedApp` if it is an app-specific shortcut remap. - For the `isShortcutInvoked` is true scenario (i.e. the initial remap keydown section is done) there are several cases depending on the key pressed or released: - **Case 1:** If a modifier in the original shortcut is released, we need to reset back to the physical keys pressed. - For remap to shortcut, we release the target action key if it is currently pressed, and depending on whether all the modifiers of the original shortcut are present in the target, we release the target modifiers that are not common, and set the remaining original shortcut modifiers except the one that was released. We do not need to send the original action key as that will get generate it's own key event if it is held down. - For remap to key, we release the target key if it is pressed (and it is not remapped to Disable), and we set the original shortcut modifiers. - For both the cases we send a dummy key event at the end, since we are setting modifiers without any other key after that, and we reset all the remap variables. - **Case 2:** If the original shortcut's action key is pressed again, we send the target shortcut's action key or the target key again (or for disable we just suppress the event). - **Case 3:** If the original shortcut's action key is released - For remap to shortcut, we just release the target shortcut's action key - For remap to disable, we suppress the event - For remap to key, we check if any other keys are pressed apart from the target key. If not, we just release the target key. If there are, we reset back to the physical keys by releasing the target key and setting the original shortcut's modifiers along with

a dummy key, and we reset all the remap variables. This behavior is different from remap to shortcut because if the action key is released while other keys are pressed the remap should be inactive, but such a state can't occur for shortcut to shortcut remaps since they happen only when the exact keys are pressed. - **Case 4:** If a modifier in the original shortcut is pressed, suppress the event - **Case 5:** If any other key is pressed - For remap to shortcut, we need to reset back to physical keys as the shortcut remaps can't be pressed in combination with other keys. We release the target action key if it was pressed, and we release the modifier keys of the target shortcut that are not common and set the remaining ones in the original shortcut. We then send the original shortcut's action key if the target action key was found to be pressed, and we send the current key press at the end. - For remap to key, if it is remapped to disable or if the target key is not found to be pressed, we reset to the physical keys, we set the original shortcut's modifiers and if is remap to Disable and the original shortcut's action key is physically pressed (this is checked by the `isOriginalActionKeyPressed` flag which we keep track of whenever the action key is pressed or released for remap to Disable), then we set the original shortcut's action key, followed by the current key press. If it is not remapped to disable and the target key is pressed, then we don't suppress the event as we allow shortcut to key remappings to be pressed along with other keys. - For all the above cases, dummy key isn't required as we want the current key press to behave like a normal key. - **Case 6:** If any other key is released, do not suppress the event as this event didn't appear with a corresponding key down event (such as an app sending a key up event) or we processed the key down and let it continue (for shortcut to key scenario). - All the remapped key events that we send above are sent with `KEYBOARDMANAGER_SHORTCUT_FLAG` on the `dwExtraInfo` field, except the usage of the current key press in Case 5, for which we don't send any extra info so that it is considered as a normal key event which may in turn invoke some other remap.

Note: Shortcuts are considered valid if they have modifiers and an action key. The reason why we haven't supported key combinations of just modifiers (which is requested in this issue) (like remapping Ctrl+Alt) is because this would require more cases and handling as these remappings have to take place only on press and release and if there is no key pressed in between similar to what Start Menu does. The remapping would have to be invoked only for this specific sequence Ctrl key down, Alt key down, Alt key up, Ctrl key up (ordering between Ctrl and Alt can be swapped). If any other key is pressed in between it shouldn't be invoked, and since this logic requires tracking exact states instead of using GetAsyncKeyStates, this could cause false positives if a user is not running as admin.

HandleOSLevelShortcutRemapEvent

This method is used for handling global shortcut to shortcut and shortcut to key remaps. The general logic is as follows: - Check if the `dwExtraInfo` field

is set to `KEYBOARDMANAGER_SHORTCUT_FLAG`. This indicates that the key event was generated by the KBM shortcut remap method using `SendInput`. This ensures that we don't read events generated by the shortcut remap method, but we still read events which are generated by the key remap method. - Call `HandleShortcutRemapEvent` without the `activatedApp` argument so that global shortcut remapping takes place if it applies for the current key event.

HandleAppSpecificShortcutRemapEvent

This method is used for handling app-specific shortcut to shortcut and shortcut to key remaps. The general logic is as follows: - Check if the `dwExtraInfo` field is set to `KEYBOARDMANAGER_SHORTCUT_FLAG`. This indicates that the key event was generated by the KBM shortcut remap method using `SendInput`. This ensures that we don't read events generated by the shortcut remap method, but we still read events which are generated by the key remap method. - Get the name of the process in the foreground. This is done using `GetCurrentApplication` which uses `GetForegroundWindow` to get the window handle and `get_process_path` from the common lib. This approach can fail for UWP apps in full screen, so for that scenario we use the `GetGUIThreadInfo` approach to find the correct window handle, and hence the correct process name. This method is described in more detail here - By checking `KeyboardManagerState.GetActivatedApp` we check if an app-specific shortcut is currently invoked. If so, we consider this application to be the activated app. This is required because some shortcut remaps could cause the current app to lose focus and hence until the shortcut is completely released we should allow that remap to continue, otherwise the user could end up in a state where some keys do not get released. For example: remap Ctrl+A to Alt+Tab for Edge, when a user presses Ctrl+A the window loses focus as Alt+Tab gets executed. - If there is no app-specific shortcut currently invoked, we check if the foreground process is present in the list of app-specific remaps, either with or without the file extension and case insensitive. If it is, this is considered to be the activated app. - Call `HandleShortcutRemapEvent` with the `activatedApp` argument so that app-specific shortcut remapping takes place if it applies for the current key event.

HandleSingleKeyToggleToModEvent (Obsolete - Code from PoC which is commented out)

This method was added to support a feature for converting the behavior of a key from behaving like a toggle (like Caps Lock/Num Lock) to a modifier (like Ctrl), such that when you hold Caps Lock it would behave as if Caps Lock was active, and when it was not pressed Caps Lock would be off. For Caps Lock this would be similar to behaving like Shift, but for Num Lock there is no existing key which can substitute for this. This was added while testing out remapping for the KBM PoC, but wasn't added as a feature since it wasn't a priority.

Tests

In order to test the remapping logic, a mocked keyboard input handler had to be created because otherwise the tests would process and send actual key events. For this the `InputInterface` was made, and in production code the methods are implemented using `SendInput` and `GetAsyncKeyState`. In addition to this, `GetCurrentApplication` had to be mocked so that app-specific remapping can be tested.

MockedInput

The `MockedInput` class uses a 256 size `bool` vector to store the key state for each key code. Identifying the foreground process is mocked by simply setting and getting a string value for the name of the current process.

To mock the `SendInput` method, the steps for processing the input are as follows. This implementation is based on public documentation for SendInput and the behavior of key messages and keyboard hooks: - Iterate over all the inputs in the INPUT array argument - If the event is a key up event, then it is considered `WM_SYSKEYUP` if Alt is held down, otherwise it is `WM_KEYUP`. - If the event is a key down event, then it is considered `WM_SYSKEYDOWN` if either Alt is held down or if it is F10, otherwise it is `WM_KEYDOWN`. - An optional function which can be set on the `MockedInput` handler can be used to test for the number of times a key event is received by the system with a particular condition using `sendVirtualInputCallCondition`. - The hook logic for a low level hook which returns 0 or 1 can be set on the `MockedInput` handler such that it behaves like a low level hook would behave with actual keyboard input. If the method returns 1, then the keyboard state is not updated, and if it returns 0 the corresponding key event is used to update the key state. This works in the recursive way as well similar to low level hooks, as `SendVirtualInput` can be called from within the hook, thus simulating identical behavior to calling `SendInput` in a low level hook (as soon as SendInput is called, the low level hook is called for the new input event, and only after those are processed it returns back to the current event, check this blog for more details). - For updating the keyboard state, KEYUP messages result in the state for that key code being set to false, and KEYDOWN result in the state for that key code being set to true. - For modifiers the behavior is slightly different as if the key state of the L/R version is modified, it should also modify the common version, and if a common version is released, it should release both the L and R versions.

Tests for single key remaps and shortcut remaps

Using the MockedInput handler, all the expected (and known) key scenarios that can occur for while pressing a remapped key or remapped shortcut are tested. The foreground app behavior which is specific to app-specific shortcuts is tested here.