Window Walker plugin

The window walker plugin matches the user entered query with the open windows on the system. The user can switch to the found windows, close them or kill their process.

Image of Window Walker plugin

Remarks

UWP Apps

- The process of an UWP app can't be detected correctly for windows that are minimized while searching. At this time they are assigned to the generic process `ApplicationFrameHost.exe`. If the user searches for such an window while it is not minimized, then the process gets assigned correctly/updated.

Killing processes

- Killing the Explorer process is only allowed, if each folder window is running in its own process. (See section `File Explorer setting` below.)
- You can only kill elevated processes, if you have admin permissions (UAC).
- If you kill the process of an UWP app window, you kill all instances of the app. All windows are assigned to the same process.
- Windows of UWP apps don't know their process, until they are searched in non-minimized state.

File Explorer setting

- To kill the Process of an Explorer window, each window has to run in a separate process. Otherwise the process is the same one as the shell process and killing the shell process will crash the shell (Windows ui).
- To enable this behavior the setting `Launch folder windows in a separate process` under `Folder Options > View` has to be enabled.
- From PowerToys Run you can open the `Folder options` dialog by clicking the information message in the search results. The information message is only shown when searching with action keyword for explorer windows and can be hidden in the plugin settings.
- Note: The folder option/process is evaluated in real time. After changing the setting it is enough to search again for the windows.

Folder options for Window Walker

Optional plugin settings

- The optional plugin settings are implemented via the `ISettingProvider` interface from `Wox.Plugin` project.

- All available settings for the plugin are defined in the `WindowWalkerSettings` class of the plugin. The settings can be accessed everywhere in the plugin code via the static class instance `WindowWalkerSettings.Instance`.
- We have the following settings that the user can configure to change the behavior of the plugin:

Key	Default value	Name/Description
`ResultsFromVisibleDesktopOnly`	`false`	Show only results from visible desktop
`SubtitleShowPid`	`false`	Show process id in subtitle
`SubtitleShowDesktopName`	`true`	Show desktop name in subtitle (If two or more desktops exist)
`ConfirmKillProcess`	`true`	Request confirmation when killing a process
`KillProcessTree`	`false`	Kill process and it's child processes
`OpenAfterKillAndClose`	`false`	Stay open after closing windows and killing processes (Not working with kill process confirmation)
`HideKillProcessOnElevatedProcesses`	`false`	Hide "kill process" button if additional permissions required
`HideExplorerSettingInfo`	`false`	Hide Explorer process information

Technical details

`OpenWindows.cs`

- The window walker plugin uses the `EnumWindows` function to enumerate all the open windows in the `OpenWindows.cs` class.

`SearchController.cs`

- The `SearchController` encapsulates the functions needed to search and find matches.
- It is responsible for updating the search text and performing a fuzzy search on all the open windows.

`Window.cs`

- The `Window` class represents a specific window and has functions to get the name of the window, the state of the window (whether it is visible or not), the `SwitchTowindow` function which switches the desktop focus to the selected window and the `CloseThisWindow` function which closes the window. The `SwitchTowindow` action is performed when the user clicks on a window walker plugin result.
- The `Window` class holds a static cache with the process information of all windows we know so far and each window instance has a property which holds its process information (name, file, ...). The process data in the cache and the window property are of the type `WindowProcess`.
- To get the desktop information for a window, we use the common `VirtualDesktopHelper` in `Wox.Plugin` project. The instance of `VirtualDesktopHelper` is cached in the `Main` class of the plugin at runtime. The desktop information is stored in a property of the type `VDesktop`.

`WindowProcess.cs`

- The `WindowProcess` class represents a specific process for a window.
- It contains static methods to query process information from the system and instance methods/properties to hold/retrieve the process information we want to know about a window's process.
- Additionally, it contains the method `KillThisProcess` to kill the process. (If the user has not enough permissions to kill a process they are requested via UAC.)

`ResultHelper.cs`

- The `ResultHelper` class contains the code to create the list with all results for PT Run based on the data returned from `SearchController` class.
- There is a special result that is added if the folder windows doesn't run in separate processes and the user searches for Explorer windows using the action keyword.
  - This result informs the user that there is a setting that must be enabled to be able to kill Explorer processes.
  - The result can be disabled in plugin options. When it is clicked it opens the folder options.

`ContextMenuHelper.cs`

- The `ContextMenuHelper` class provides the code for the context menu items.

`WindowWalkerSettings.cs`

- The `WindowWalkerSettings` class provides access to all optional plugin settings.
- The class has a static property called `Instance` that holds an instance of the class itself. This allows us to access the settings from everywhere in the plugin code without having additional parameters in our methods.

Score

The window walker plugin uses `FuzzyMatching` to get the matching indices and calculates the score by creating a 2 dimensional array of the window and the query text.

Unit Tests

We have a Unit Test project that executes various test to ensure that the plugin works as expected.

`PluginSettingsTests.cs`

- The `PluginSettingsTests.cs` class contains tests to validate that all settings exist and that they have the correct default values.