FancyZones library brief overview

1. `Zone` - Class which is basically wrapper around rectangle structure, representing one zone inside applied zone layout. ZoneSet is holding array of these which represent zone layout.

2. `ZoneSet` - Class implementing actual zone layout applied. What this means is that ZoneSet is responsible for actual calculation of rectangle coordinates (whether is grid or canvas layout) and moving window through them. WorkArea holds ZoneSet structure which represents currently active zone set.

3. `WorkArea` - Class representing work area, which is defined by monitor and current virtual desktop. For an example, if You have two monitors connected and two virtual desktops, You have 4 work areas available, and each of them can have separate zone layout. WorkArea is describing single work area. As mentioned before it holds active ZoneSet.

4. `FancyZones` - Top level entity and entry point for all user actions (which goes through actual module interface). Some of the main responsibilities of FancyZones class:

   ```
    1. Starting FancyZones Editor (C#) with appropriate command line arguments on
   user request.
    2. Keeping track of WorkArea per monitor (currently active work area on each
   connected monitor).
    3. Keeping track of active virtual desktops. This is performed in separate
   thread by polling VirtualDesktopIDs registry key and parsing its content.
    4. Detecting every change in work environment, such as creating / destroying /
   switching between virtual desktops, closing FancyZones Editor, changing display
   settings and handling those changes.
   ```

Proposal for modifications of handling described in 4.4:

Currently after each of the mentioned changes in work environment we are calling EnumDisplayMonitors windows API, and passing callback function to it. EnumDisplayMonitors works asynchronous and triggers that callback for each work area available (as mentioned in previous example, for two monitors and two virtual desktops, we have this callback triggered four times). As mentioned previously, we have WorkArea class as our representation of this work area. And what we do, every time this callback is triggered we destroy previous WorkArea object for that work area and create new one, even though that it is most likely that nothing has changed (e.g. just switching back and forth between virtual desktops). This constant creation and deletion of WorkArea has caused some problems in the past and it's not ideal for some other fixes we would like to make in the multi-monitor/multi-desktop scenario.

As mentioned in 4.3 we already have tracker of virtual desktops implemented. Idea is to use this functionality and to extend it bit more, so we can track if work area (WorkArea) is new one, or already processed and skip creating new WorkArea objects and deleting old ones every time, even if nothing changed in it. We will keep map, where virtual desktop id is the key, and values are already processed monitors (virtual desktop exists across all monitors). Once we receive callback from EnumDisplayMonitors, indicating work area (defined by virtual desktop id and monitor) we can check if it’s new or not, and act accordingly (create new WorkArea for it or not). Deleting virtual desktop (which is also registered in 4.3), will trigger updates in this map, and also updates in our JSON storage.