Architecture

Overview

`PowerToys Run` is a plugin-based .net core desktop application. It is written in WPF using `Model-View-ViewModel (MVVM)` structural design pattern. This article provides an overview of `PowerToys Run` architecture and introduces major components in the data flow.

Note : We refer to base application without plugins as `PowerLauncher`, which is same as the name of startup WPF project.

UI

PowerToys Run UI is written in the WPF framework. The UI code is present in the Powerlauncher project and is spanned across three high-level components: `MainWindow.xaml`, `LauncherControl.xaml` and `ResultList.xaml`. These components are discussed below.

 Image of PowerToys Run UI **Fig 1: PowerToys Run UI architecture**

1. **`MainWindow.xaml`** : This is the outermost-level UI control. It is composed of lower-level UI components such as `LauncherControl.xaml` and `ResultList.xaml`. The corresponding code-behind file implements all the UI related functionalities such as autosuggest, key-bindings, toggling visibility of WPF window and animations.
2. **`LauncherControl.xaml`** : This control implements the UI component for editing query text.(marked in red in Fig 1) It consists of two overlapping WPF controls, `TextBox` and `TextBlock`. The outer `TextBox` is used for editing query whereas the inner `TextBlock` is used to display autosuggest text.
3. **`ResultList.xaml`** : This control implements the UI component for displaying results (marked in green in Fig 1). It consists of a `ListView` WPF control with a custom `ItemTemplate` to display application logo, name, tooltip text, and context menu.

Data flow

The backend code is written using the `Model-View-ViewModel (MVVM)` structural design pattern. Plugins act as `Model` in this project. A detailed overview of the project's structure is given here.

Flow of data between UI(view) and ViewModels

Data flow between View and ViewModel follows typical `MVVM` scheme. Properties in viewModels are bound to WPF controls and when these properties are updated, `INotifyPropertyChanged` handler is invoked, which in turn updates UI. The diagram below provides a rough sketch of the components involved.

 Flow of data between UI(view) and ViewModels **Fig 2: Flow of data between UI and ViewModels.**

Flow of data between ViewModels and Plugins(Model)

`PowerLauncher` interact with plugins using `IPlugin` and `IDelayedExecutionPlugin` interface. `IPlugin` is used for initialization and making queries which are fast (typically return results in less than 100ms). `IDelayedExecutionPlugin` is used for long-running queries and is implemented only when required.

For example, `IDelayedExecutionPlugin` is implemented by indexer plugin for searching files with names of form *abc*.

```
public interface IPlugin
{
    // Query plugin
    List<Result> Query(Query query);

    // Initialize plugin
    void Init(PluginInitContext context);
}

public interface IDelayedExecutionPlugin : IFeatures
{
    // Query plugin
    List<Result> Query(Query query, bool delayedExecution);
}
```

 Flow of data between UI(view) and ViewModels **Fig 3: Flow of data between ViewModels and Plugins.**

Requesting services from powerlauncher

Plugins could use the `IPublicAPI` interface to request services such as getting the current theme (for deciding logo background), displaying messages to the user, and toggling the visibility of PowerLauncher.