Classes and structures

class Animation: header source

Animation helper class with two easing-in animations: linear and exponential.

class AsyncMessageQueue: header

Header-only asynchronous message queue. Used by `TwoWayPipeMessageIPC`.

class TwoWayPipeMessageIPC: header

Header-only asynchronous IPC messaging class. Used by the runner to communicate with the settings window.

class DPIAware: header source

Helper class for creating DPI-aware applications.

struct MonitorInfo: header source

Class for obtaining information about physical displays connected to the machine.

class Settings, class PowerToyValues, class CustomActionObject: header source

Classes used to define settings screens for the PowerToys modules.

class Tasklist: header source

Class that can detect the position of the windows buttons on the taskbar. It also detects which window will react to pressing `WinKey + number`.

struct WindowsColors: header source

Class for detecting the current Windows color scheme.

Helpers

Common helpers: header source

Various helper functions.

Settings helpers: header

Helper methods for the settings.

Start visible helper: header source

Contains function to test if the Start menu is visible.

Toast Notifications

Notifications API header source

To use UWP-style toast notifications, simply include the header and call one of these functions:

```
    void show_toast(std::wstring_view message);        // #1

    void show_toast_background_activated(              // #2
      std::wstring_view message,
      std::wstring_view background_handler_id,
      std::vector<std::wstring_view> button_labels);
```

We might add more functions in the future if the need arises, e.g. `show_toast_xml` which will accept raw XML for rich customization.

Description:

- `#1` is for sending simple notifications without any callbacks or buttons
- `#2` is capable of showing a toast with multiple buttons and background activation
- `message` is a plain-text argument

Implement a toast activation handler/callback as a function in handler_functions.cpp and register its `background_handler_id` via `handlers_map`, e.g.:

```
// Your .cpp where you'd like to show a toast

#include <common/notifications.h>

void some_func() {
// ...
  notifications::show_toast_background_activated(
    L"Toast message!",                                      // text
displayed in a toast
    L"awesome_toast",                                       //
activation handler id
    {L"Press me!", L"Also could press me!", L"I'm here to be pressed!"} // buttons
in a toast
  );
```

```
// handler_functions.cpp
void awesome_toast_handler(IBackgroundTaskInstance, const size_t button_id)
{
  switch(button_id)
  {
    case 0:
      // handle "Press me!" button click
    case 1:
      // handle "Also could press me!" button click
    case 2:
      // handle "I'm here to be pressed!" button click
  }
}

namespace
{
```

```
const std::unordered_map<std::wstring_view, handler_function_t> handlers_map = {
  // ...other handlers...
  {L"awesome_toast", awesome_toast_handler}
};}
```

Note: since *background activation* implies that your toast handler will be invoked in a separate process, you can't share data directly from within a handler and your PT process. Also, since PT is currently a Desktop Bridge app, *foreground activation* is handled the same as background, therefore we don't make a dedicated API for it. You can read more on the rationale of the current design here.