Adapter less POC

Here I will go into detail explaining the implementation of the adapter less POC for Native Window snapshot integration.

Basic object shapes used:

These names are POC names and should not be considered for the actual implementation.

External window config; ExternalWindowConfig:

This object is used to set the left hand menu options to launch configured apps, filtering logic that builds the list of native apps to track and in snapshot fragments used to restore native window state.

```
{
  name: string,
  title: string,
  launch: ExternalProcessRequestType - https://cdn.openfin.co/docs
/javascript/19.89.58.7/global.html#ExternalProcessRequestType
}
```

External window Info; ExternalWindowInfo:

This object is assembled with the External Window API given an ExternalWindowConfig object and contains all the information we will need to restore a native window.

```
{
  name: string,
  nativeId: string,
  process: { injected: boolean, pid: number, imageName: string},
  title: string,
  visible: boolean,
  uuid: string
  alwaysOnTop: boolean,
  bounds: { x: number, y: number, width: number, height: number},
  className: string,
  dpi: number,
  dpiAwareness: number,
  focused: boolean,
  maximized: boolean,
  minimized: boolean
}
```

Snapshot fragment; SnapShotFragment:

This object is used to store store a ExternalWindowConfig along with an array of ExternalWindowInfo objects that matched the ExternalWindowConfig at a given time.

```
{
    config: externalWindowConfig,
    info: ExternalWindowInfo[]
}
```

Creating a snapshot that includes Native Windows

Step 1:

Given an array of `ExternalWindowConfig` we obtain the list of matching native windows using `fin.System.getAllExternalWindows` filtering results by the name/title combination.

Step 2:

We build a snapshot fragment by mapping over the array built in Step 1 and attaching the specific Window Instance information, resulting in a list of `ExternalWindowInfo` per `ExternalWindowConfig`.

Example:

```
[
  {
    "config": {
      "name": "Notepad",
      "title": "my_platform_notes",
      "launch": {
        "alias": "my_platform_notes",
        "target": "my_platform_notes.txt",
        "lifetime": "application",
        "arguments": ""
      }
    },
    "info": [
      {
        "name": "Notepad",
        "nativeId": "0x000F0878",
        "process": {
          "injected": false,
          "pid": 12268,
          "imageName": "C:\\Windows\\System32\\notepad.exe"
        },
        "title": "my_platform_notes.txt - Notepad",
        "visible": true,
        "uuid": "73a632d9-5f34-496e-9a76-b947cc19726c",
        "alwaysOnTop": false,
        "bounds": {
          "x": 1382,
          "y": 810,
          "width": 499,
```

```
          "height": 443
        },
        "className": "Notepad",
        "dpi": 96,
        "dpiAwareness": -1,
        "focused": false,
        "maximized": false,
        "minimized": false
      },
      {
        "name": "Notepad",
        "nativeId": "0x000A06C2",
        "process": {
          "injected": false,
          "pid": 9040,
          "imageName": "C:\\Windows\\System32\\notepad.exe"
        },
        "title": "my_platform_notes.txt - Notepad",
        "visible": true,
        "uuid": "74f9426e-2d08-4ee0-9211-d5d64fbb1ff0",
        "alwaysOnTop": false,
        "bounds": {
          "x": 1379,
          "y": 332,
          "width": 499,
          "height": 443
        },
        "className": "Notepad",
        "dpi": 96,
        "dpiAwareness": -1,
        "focused": false,
        "maximized": false,
        "minimized": false
      }
    ]
  }
]
```


Step 3

This `SnapShotFragment` will get added to the platform snapshot as `externalWindows` and returned via the normal Platform API getSnapshot


```
{
    "snapshotDetails": ...,
    "windows": [...],
    "externalWindows": [...]
}
```

Restoring a snapshot that includes Native Windows

Step 1

Given a `SnapshotFragment` we combine the information in the Snapshot Fragment with the current desktop state into a temporary object object described as a `snapshotFragmentWithDesktopState`

Snapshot fragment with desktop state; `snapshotFragmentWithDesktopState`:

```
{
    config: externalWindowConfig,
    info: ExternalWindowInfo[], //Native window info at snapshot time
    desktopState: ExternalWindowInfo[] //Native window info at restore
time
}
```

Step 2

We use a combination of three properties (HWND, CLASS_NAME, Window Title) to determine window Identity, we depend on the **application's implementation** in order to uniquely identify each window on the screen.

Table describing heuristics:

HWND is unique	Class is unique	Title is unique	Is Identifiable	Example scenario
				Snapshot is operating on a set of windows that have not been closed.
				Re-launched windows that have unique Titles
				Windows that have not been closed but title is ever changing (Content updates title property)
				Re-launched windows but title is ever changing (Content updates title property)

As we can see on the table, if an Application does not have stable Window Titles our Window matching is off, if an Application re-uses the same Window Title for multiple child Windows then our matching is off.

The matching will be 100% accurate if the user does not close any of the active Windows, and this is a real concern as the user will encounter two distinct behaviors depending if the snapshot was taken when the current apps were still present.

In order to support the heuristics we build a temporary object with the following Information:

Key	Description
matched	Windows on the desktop that match Native Ids (HWND) Windows on the snapshot
unmatched	Windows on the desktop that match a snapshot rule but not the Native Id (HWND)
reminder	Windows on the snapshot that do not match any window on the desktop by Native Id (HWND)

Matched state; `matchedState`:

```
{
  snapshotFragement: snapshotFragement[],
  matched: ExternalWindowInfo[],
  unmatched: ExternalWindowInfo[],
  reminder: ExtXernalWindowInfo[],
}
```

Step 3

Given the `matchedState` data we will need to make some choices, matched windows are restored to their positions, the remaining delta will be resolved by either launching or adopting existing windows.

POC Implementation

Configuration:

The POC ships with a few files to make it easier to launch/restore Native Widows. We also provide an array of `ExternalWindowConfig` by means of customData.NativeApplications.

Manifest

```
"platform": {
    ...
    "appAssets": [
        {
            "src": "http://localhost:5552/my_platform_notes.zip",
            "alias": "my_platform_notes",
            "version": "1.1.6"
        }
    ],
    "customData": {
        "nativeApplications": [],
                {
                    "name": "Notepad",
                    "title": "my_platform_notes",
                    "launch": {
                        "alias": "my_platform_notes",
                        "target": "my_platform_notes.txt",
                        "lifetime": "application",
                        "arguments": ""
                    }
                },
    }
}
```

Launching the Applications

The `left-menu.js` module has been modified to list the Native Applications and launch them, with a hook to add the native Id's to a track list.

Generating snapshots

The `platform-provider.js` file has been modified to extend the getSnapshot and setSnapshot functions with a custom module found in `external-window-snapshot.js`, it keeps both a list of `ExternalWindowConfig` objects passed by configuration and also any Native Applications launched by the Platform.

Restoring snapshots

If the "externalWindows" key is present in a snapshot, the `platform-provider.js` will invoke the restore functionality exposed by `external-window-snapshot.js`