Listing 1: IDE

```

//
// ExpiIDE
// FileInspectorPresenter.cs
//
// Copyright © 2022 Nikolai Tiunin. All rights reserved.
//

using ExpiIDE.Core;
using Presentation.Modules.EXP;
using Presentation.Modules.IDE;
using Presentation.Modules.NEC;
using Presentation.Modules.SVG;
using Presentation.Modules.TXT;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Windows.Forms;

namespace Presentation.Modules.FileInspector
{

    public class FileHierarchyItem
    {
        public string name;
        public string path;

        public bool IsExists
        {
            get
            {
                return File.Exists(path);
            }
        }

        public FileHierarchyItem(string name, string path)
        {
            this.path = path;
            this.name = name;
        }

        public static FileHierarchyItem Item(string path)
        {
            if (Directory.Exists(path))
            {
                return FolderItem(path);
            }
            return ContentItem(path);
        }

```

```
53          public static FolderItem FolderItem(string path)
54          {
55              var name = new FileInfo(path).Name;
56              return new FolderItem(name, path);
57          }
58
59          public static ContentItem ContentItem(string path)
60          {
61              var name = new FileInfo(path).Name;
62              var ext = name.Split('.').Last();
63              var type = FileItemType.unknown;
64              switch (ext)
65              {
66                  case "nec":
67                      type = FileItemType.nec;
68                      break;
69                  case "svg":
70                      type = FileItemType.svg;
71                      break;
72                  case "exp":
73                      type = FileItemType.exp;
74                      break;
75                  case "txt":
76                      type = FileItemType.txt;
77                      break;
78              }
79              return new ContentItem(name, path, type);
80          }
81
82          public FileHierarchyItem Find(string keyPath)
83          {
84              if (name == keyPath)
85              {
86                  return this;
87              }
88              if (keyPath.Length == 0)
89              {
90                  return null;
91              }
92              var slashIndex = keyPath.IndexOf('\\');
93              var title = keyPath;
94              var remaining = "";
95              if (slashIndex > 0)
96              {
97                  title = keyPath.Substring(0, slashIndex);
98                  remaining = keyPath.Remove(0, slashIndex + 1);
99              }
100             if (title != name)
101             {
102                 return null;
103             }
104             if (title == name && remaining.Length == 0)
105             {
106                 return this;
```

```
107             }
108             if (this is FolderItem)
109             {
110                 var folder = (FolderItem)this;
111                 foreach (var item in folder.items)
112                 {
113                     var subItem = item.Value.Find(remaining);
114                     if (subItem != null)
115                     {
116                         return subItem;
117                     }
118                 }
119             }
120             return null;
121         }
122     }
123     public class FolderItem: FileHierarchyItem
124     {
125         public Dictionary<string, FileHierarchyItem> items;
126         public bool isLoaded = false;
127         public bool isExpanded = false;
128 
129         public bool IsExists
130         {
131             get
132             {
133                 return Directory.Exists(path);
134             }
135         }
136 
137 
138         public FolderItem(string name, string path): base(name, path)
139         {
140             items = new Dictionary<string, FileHierarchyItem>();
141         }
142 
143         public bool Load()
144         {
145             var isChanged = false;
146             var checkList = new Dictionary<string, FileHierarchyItem>();
147             foreach (var item in items)
148             {
149                 checkList[item.Key] = item.Value;
150             }
151             var files = Directory.GetFiles(path);
152             var directories = Directory.GetDirectories(path);
153             var content = new List<string>();
154             content.AddRange(files);
155             content.AddRange(directories);
156             foreach(var path in content)
157             {
158                 FileHierarchyItem item;
159                 if (checkList.ContainsKey(path) == false)
160                 {
```

```
                    item = Item(path);
                    items[path] = item;
                    isChanged = true;
                }
                else {
                    item = checkList[path];
                    checkList.Remove(item.path);
                }

                if (item is FolderItem)
                {
                    isChanged |= ((FolderItem)item).Load();
                }
            }
            if (checkList.Count > 0)
            {
                foreach (var item in checkList)
                {
                    items.Remove(item.Key);
                }
                isChanged = true;
            }
            return isChanged;
        }

        public void Toggle()
        {
            isExpanded = !isExpanded;

        }
    }

    public class ContentItem : FileHierarchyItem
    {
        public FileItemType type;

        public ContentItem(string name, string path, FileItemType type) :
            base(name, path)
        {
            this.type = type;
        }
    }


    public partial class FileInspectorPresenter
    {
        IDEModuleOutput output;
        public FileInspectorView view;
        EXPModule expModule;
        SVGModule svgModule;
        NECModule necModule;
        TXTModule txtModule;
        FolderItem root;
        System.Timers.Timer timer;
```

```
214         Dictionary<string, FileItem> fileItems = new Dictionary<string,
                FileItem>();
215         FileItem currentFileItem = null;
216         private int timerTicks = 0;
217 
218         public FileInspectorPresenter(
219             string path,
220             EXPModule expModule,
221             SVGModule svgModule,
222             NECModule necModule,
223             TXTModule txtModule,
224             IDEModuleOutput output)
225         {
226             this.expModule = expModule;
227             this.svgModule = svgModule;
228             this.necModule = necModule;
229             this.txtModule = txtModule;
230             var directory = path;
231             if (Directory.Exists(path) == false)
232             {
233                 var info = new FileInfo(path);
234                 directory = info.Directory.FullName;
235             }
236             root = FileHierarchyItem.FolderItem(directory);
237             root.isExpanded = true;
238             this.output = output;
239             StartTimer();
240         }
241 
242         ~FileInspectorPresenter()
243         {
244             StopTimer();
245         }
246 
247         public void DidSelect(string path)
248         {
249             var item = root.Find(path);
250             if (item == null || item is ContentItem == false)
251             {
252                 return;
253             }
254             var contentItem = (ContentItem)item;
255             var fileItem = FindOrCreateFileItem(contentItem);
256 
257             switch (contentItem.type)
258             {
259                 case FileItemType.exp:
260                     currentFileItem = fileItem;
261                     expModule.Input.Open(fileItem);
262                     view.previewView.Show(expModule.View);
263                     break;
264                 case FileItemType.nec:
265                     currentFileItem = fileItem;
266                     necModule.Input.Open(item.path);
```

```
                    view.previewView.Show(necModule.View);
                    break;
                case FileItemType.svg:
                    currentFileItem = null;
                    svgModule.Input.Open(item.path);
                    view.previewView.Show(svgModule.View);
                    break;
                case FileItemType.txt:
                    currentFileItem = fileItem;
                    txtModule.Input.Open(fileItem);
                    view.previewView.Show(txtModule.View);
                    break;
                default:
                    currentFileItem = null;
                    view.previewView.ShowPlaceholder();
                    break;
            }
            output.DidUpdate(fileItem);
        }

        public void UpdateContent()
        {
            if (currentFileItem == null)
            {
                view.previewView.ShowPlaceholder();
                return;
            }
            switch (currentFileItem.type)
            {
                case FileItemType.exp:
                    expModule.Input.Update();
                    break;
                case FileItemType.txt:
                    txtModule.Input.Update();
                    break;
                case FileItemType.nec:
                    break;
                case FileItemType.svg:
                    break;
                default:
                    view.previewView.ShowPlaceholder();
                    break;
            }
        }

        public void DidRequestOptions(string path, Point location)
        {
            var item = root.Find(path);

            if (item == null)
            {
                return;
            }

```

```
321             var del = new ToolStripMenuItem("    ", null, (o, e) => {
322                 Delete(item.path);
323             });
324             if (item is FileItem)
325             {
326                 Show(new ToolStripItem[] { del }, location);
327             } else
328             {
329                 var folder = item.path;
330                 var createFolder = new ToolStripMenuItem("   ", null, (o,
                        e) => {
331                     view.ShowCreateFileView("folder", (name) => {
332                         CreateFolder($"{folder}\\{name}");
333                     });
334                 });
335                 var createFile = new ToolStripMenuItem("  ", null, (o, e)
                        => {
336                     view.ShowCreateFileView("file", (name) => {
337                         CreateFile($"{folder}\\{name}");
338                     });
339                 });
340                 var create = new ToolStripMenuItem("    ", null, new
                        ToolStripItem[] {
341                    createFolder, createFile
342                 });
343                 Show(new ToolStripItem[] {
344                     create, del
345                 }, location);
346             }
347         }
348 
349         public void Expand(string path)
350         {
351             var item = root.Find(path);
352             if (item is FolderItem == false)
353             {
354                 return;
355             }
356             var folder = (FolderItem)item;
357             folder.isExpanded = true;
358         }
359 
360         public void Collapse(string path)
361         {
362             var item = root.Find(path);
363             if (item is FolderItem == false)
364             {
365                 return;
366             }
367             var folder = (FolderItem)item;
368             folder.isExpanded = false;
369         }
370 
371         private void Show(ToolStripItem[] items, Point location)
```

```
372         {
373             view.hierarchyView.ShowContextMenu(items, location);
374         }
375 
376         private void CreateFile(string path)
377         {
378             File.WriteAllText(path, "");
379         }
380 
381         private void CreateFolder(string path)
382         {
383             Directory.CreateDirectory(path);
384         }
385 
386         private void Delete(string path)
387         {
388             if (Directory.Exists(path))
389             {
390                 Directory.Delete(path, true);
391             }
392             else if (File.Exists(path))
393             {
394                 File.Delete(path);
395             }
396         }
397 
398         private void StartTimer()
399         {
400             timerTicks = 0;
401             timer = new System.Timers.Timer(300);
402             timer.Elapsed += new System.Timers.ElapsedEventHandler(
                    TimerFired);
403             timer.AutoReset = true;
404             timer.Start();
405         }
406 
407         private void StopTimer()
408         {
409             if (timer != null)
410             {
411                 timer.Stop();
412             }
413         }
414 
415         private void TimerFired(object sender, EventArgs e)
416         {
417             timerTicks++;
418             if (timerTicks % 10 == 0)
419             {
420                 timerTicks = 0;
421                 UpdateUndoIfNeeded();
422             }
423             view.Invoke(new Action(() => {
424                 UpdateToolsIfNeeded();
```

```
425             }));
426             if (view == null)
427             {
428                 return;
429             }
430             if (root.Load() == false)
431             {
432                 return;
433             }
434             view.Invoke(new Action(() => {
435                 UpdateView();
436             }));
437         }
438 
439         private FileItem FindOrCreateFileItem(ContentItem contentItem)
440         {
441             if (fileItems.ContainsKey(contentItem.path))
442             {
443                 return fileItems[contentItem.path];
444             }
445             var item = new FileItem(contentItem.path, contentItem.type);
446             fileItems[contentItem.path] = item;
447             return item;
448         }
449     }
450 
451     partial class FileInspectorPresenter : FileInspectorModuleInput
452     {
453         public void Redo()
454         {
455             if (currentFileItem == null)
456             {
457                 return;
458             }
459             currentFileItem.Redo();
460             UpdateContent();
461         }
462 
463         public void Undo()
464         {
465             if (currentFileItem == null)
466             {
467                 return;
468             }
469             currentFileItem.Undo();
470             UpdateContent();
471         }
472         public void Save()
473         {
474             if (currentFileItem == null)
475             {
476                 return;
477             }
478             currentFileItem.Save();
```

```
479         }
480
481         public void ToggleRun()
482         {
483             if (currentFileItem == null)
484             {
485                 return;
486             }
487             switch (currentFileItem.type)
488             {
489                 case FileItemType.exp:
490                     expModule.Input.ToggleRunning();
491                     break;
492                 default:
493                     break;
494             }
495         }
496
497
498         public void UpdateView()
499         {
500             var items = (root.IsExists) ?
501                 new FileHierarchyItem[] { root } :
502                 new FileHierarchyItem[] { };
503             view.hierarchyView.UpdateHierarchyItems(items);
504         }
505
506         private void UpdateToolsIfNeeded()
507         {
508             output.DidUpdate(currentFileItem);
509         }
510
511         private void UpdateUndoIfNeeded()
512         {
513             if (currentFileItem == null)
514             {
515                 return;
516             }
517             currentFileItem.UpdateHistory();
518         }
519
520         public bool IsRunning(FileItem fileItem)
521         {
522             return expModule.Input.IsRunning(fileItem);
523         }
524     }
525 }
```