Работа с файлами

Последнее обновление: 01.04.2017

Каждая мобильная платформа имеет свою специфику, свою файловую систему, свою структуру каталогов. Поэтому не существует некого единообразного подхода для работы с файлами для всех трех платформ. К счастью, подходы будут несильно отличаться.

Создадим новый проект по типу **Cross Platform App (Xamarin.Forms or Native)** по типу **Portable Class Library (PCL)**. Пусть он будет называться *FileApp*. Вначале добавим в главный проект интерфейс, который будет описывать механизм взаимодействия с файлами:

```
using System.Collections.Generic;
using System.Threading.Tasks;

namespace FileApp
{
    public interface IFileWorker
    {
        Task<bool> ExistsAsync(string filename); // проверка существования файла
        Task SaveTextAsync(string filename, string text);   // сохранение текста в файл
        Task<string> LoadTextAsync(string filename);  // загрузка текста из файла
        Task<IEnumerable<string>> GetFilesAsync();  // получение файлов из определнного каталога
        Task DeleteAsync(string filename);  // удаление файла
    }
}
```

Здесь определено 5 методов, каждый из которых выполняет определенное действие и возвращает объект Task. То есть данные методы будут асинхронными. Конкретная реализация этого интерфейса будет зависеть от конкретной платформы.

Теперь в проект для Androida добавим следующий класс:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Xamarin.Forms;
using System.Linq;

[assembly: Dependency(typeof(FileApp.Droid.FileWorker))]
namespace FileApp.Droid
{
    public class FileWorker : IFileWorker
    {
        public Task DeleteAsync(string filename)
        {
            // удаляем файл
            File.Delete(GetFilePath(filename));
            return Task.FromResult(true);
        }

        public Task<bool> ExistsAsync(string filename)
        {
            // получаем путь к файлу
            string filepath = GetFilePath(filename);
            // существует ли файл
            bool exists = File.Exists(filepath);
            return Task<bool>.FromResult(exists);
        }
```

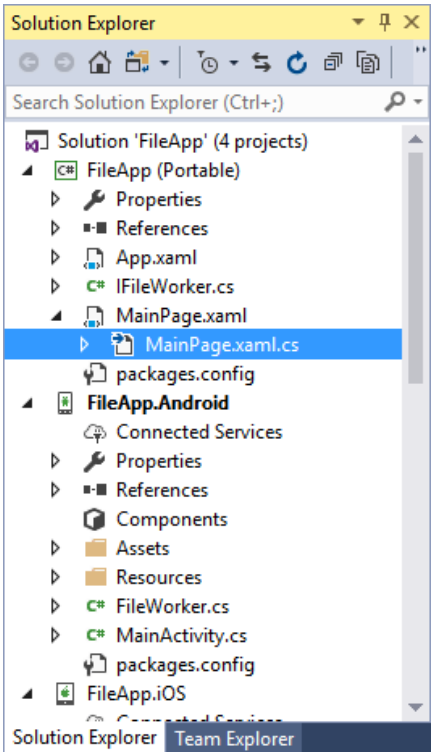
```
        public Task<IEnumerable<string>> GetFilesAsync()
        {
            // получаем все все файлы из папки
            IEnumerable<string> filenames = from filepath in Directory.EnumerateFiles(GetDocsPath())
                                            select Path.GetFileName(filepath);
            return Task<IEnumerable<string>>.FromResult(filenames);
        }

        public async Task<string> LoadTextAsync(string filename)
        {
            string filepath = GetFilePath(filename);
            using (StreamReader reader = File.OpenText(filepath))
            {
                return await reader.ReadToEndAsync();
            }
        }

        public async Task SaveTextAsync(string filename, string text)
        {
            string filepath = GetFilePath(filename);
            using (StreamWriter writer = File.CreateText(filepath))
            {
                await writer.WriteAsync(text);
            }
        }
        // вспомогательный метод для построения пути к файлу
        string GetFilePath(string filename)
        {
            return Path.Combine(GetDocsPath(), filename);
        }
        // получаем путь к папке MyDocuments
        string GetDocsPath()
        {
            return Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
        }
    }
}
```

Для управления файлами используем методы класса System.IO.File. Кроме реализаций методов интерфейса здесь также определены дополнительные методы `GetDocsPath()` (для получения полного пути к папке Мои Документы) и `GetFilePath()` (для получения пути к файлам).

Но чтобы система видела этот класс и могла его связать при использовании контейнера IoS, необходимо использовать атрибут `Dependency`.



Также добавим аналогичный класс в проект для iOS:

```
using System;
```

```
using System.Collections.Generic;
using System.IO;
using System.Threading.Tasks;
using Xamarin.Forms;
using System.Linq;

[assembly: Dependency(typeof(FileApp.iOS.FileWorker))]
namespace FileApp.iOS
{
    public class FileWorker : IFileWorker
    {
        public Task DeleteAsync(string filename)
        {
            File.Delete(GetFilePath(filename));
            return Task.FromResult(true);
        }

        public Task<bool> ExistsAsync(string filename)
        {
            string filepath = GetFilePath(filename);
            bool exists = File.Exists(filepath);
            return Task<bool>.FromResult(exists);
        }

        public Task<IEnumerable<string>> GetFilesAsync()
        {
            IEnumerable<string> filenames = from filepath in Directory.EnumerateFiles(GetDocsPath())
                                            select Path.GetFileName(filepath);
            return Task<IEnumerable<string>>.FromResult(filenames);
        }

        public async Task<string> LoadTextAsync(string filename)
        {
            string filepath = GetFilePath(filename);
            using (StreamReader reader = File.OpenText(filepath))
            {
                return await reader.ReadToEndAsync();
            }
        }

        public async Task SaveTextAsync(string filename, string text)
        {
            string filepath = GetFilePath(filename);
            using (StreamWriter writer = File.CreateText(filepath))
            {
                await writer.WriteAsync(text);
            }
        }

        string GetFilePath(string filename)
        {
            return Path.Combine(GetDocsPath(), filename);
        }
        string GetDocsPath()
        {
            return Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);
        }
    }
}
```

Код для iOS практически не отличается от аналогичного кода для Androida.

И также добавим такой же класс в проект для UWP:

```
using System;
using System.Collections.Generic;
using Windows.Storage;
using System.Linq;
using Xamarin.Forms;
using System.Threading.Tasks;

[assembly: Dependency(typeof(FileApp.UWP.FileWorker))]
namespace FileApp.UWP
{
    public class FileWorker : IFileWorker
    {
        public async Task DeleteAsync(string filename)
        {
            StorageFolder localFolder = ApplicationData.Current.LocalFolder;
            StorageFile storageFile = await localFolder.GetFileAsync(filename);
            await storageFile.DeleteAsync();
```

```
        }

        public async Task<bool> ExistsAsync(string filename)
        {
            StorageFolder localFolder = ApplicationData.Current.LocalFolder;
            try
            {
                await localFolder.GetFileAsync(filename);
            }
            catch { return false; }
            return true;
        }

        public async Task<IEnumerable<string>> GetFilesAsync()
        {
            StorageFolder localFolder = ApplicationData.Current.LocalFolder;
            IEnumerable<string> filenames = from storageFile in await localFolder.GetFilesAsync()
                                            select storageFile.Name;
            return filenames;
        }

        public async Task<string> LoadTextAsync(string filename)
        {
            StorageFolder localFolder = ApplicationData.Current.LocalFolder;
            // получаем файл
            StorageFile helloFile = await localFolder.GetFileAsync(filename);
            // читаем файл
            string text = await FileIO.ReadTextAsync(helloFile);
            return text;
        }

        public async Task SaveTextAsync(string filename, string text)
        {
            // получаем локальную папку
            StorageFolder localFolder = ApplicationData.Current.LocalFolder;
            // создаем файл hello.txt
            StorageFile helloFile = await localFolder.CreateFileAsync(filename,
                                                CreationCollisionOption.ReplaceExisting);
            // запись в файл
            await FileIO.WriteTextAsync(helloFile, text);
        }
    }
}
```

В отличие от реализации IFileWorker для Android и iOS здесь мы уже используем для работы с файлами методы классы StorageFolder. А сами файлы будут храниться в папке ApplicationData.Current.LocalFolder. Собственно из-за того, что эти методы являются асинхронными и потребовалось определение методов в интерфейсе IFileWorker как асинхронных.

В проекте Portable в коде XAML у главной страницу MainPage определим следующее содержимое:

```
<?xml version="1.0" encoding="utf-8" ?>
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
             xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"

             x:Class="FileApp.MainPage">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto" />
      <RowDefinition Height="*" />
      <RowDefinition Height="*" />
    </Grid.RowDefinitions>
    <StackLayout Orientation="Horizontal">
      <Entry x:Name="fileNameEntry" HorizontalOptions="FillAndExpand" />
      <Button Text="Сохранить" Clicked="Save" />
    </StackLayout>
    <Editor Grid.Row="1" x:Name="textEditor" />
    <ListView x:Name="filesList" Grid.Row="2" ItemSelected="FileSelect">
      <ListView.ItemTemplate>
        <DataTemplate>
          <TextCell Text="{Binding}">
            <TextCell.ContextActions>
              <MenuItem Text="Удалить" IsDestructive="True" Clicked="Delete" />
            </TextCell.ContextActions>
          </TextCell>
        </DataTemplate>
      </ListView.ItemTemplate>
    </ListView>
  </Grid>
</ContentPage>
```

В коде C# у MainPage определим все обработчики используемых событий:

```
using System;
using System.Threading.Tasks;
using Xamarin.Forms;

namespace FileApp
{
    public partial class MainPage : ContentPage
    {
        public MainPage()
        {
            InitializeComponent();
        }
        protected override async void OnAppearing()
        {
            base.OnAppearing();
            await UpdateFileList();
        }
        // сохранение текста в файл
        async void Save(object sender, EventArgs args)
        {
            string filename = fileNameEntry.Text;
            if (String.IsNullOrEmpty(filename)) return;
            // если файл существует
            if (await DependencyService.Get<IFileWorker>().ExistsAsync(filename))
            {
                // запрашиваем разрешение на перезапись
                bool isRewrited = await DisplayAlert("Подверждение", "Файл уже существует, перезаписать его?", "Да",
"Нет");
                if (isRewrited == false) return;
            }
            // перезаписываем файл
            await DependencyService.Get<IFileWorker>().SaveTextAsync(fileNameEntry.Text, textEditor.Text);
            // обновляем список файлов
            await UpdateFileList();
        }
        async void FileSelect(object sender, SelectedItemChangedEventArgs args)
        {
            if (args.SelectedItem == null) return;
            // получаем выделенный элемент
            string filename = (string)args.SelectedItem;
            // загружем текст в текстовое поле
            textEditor.Text = await DependencyService.Get<IFileWorker>().LoadTextAsync((string)args.SelectedItem);
            // устанавливаем название файла
            fileNameEntry.Text = filename;
            // снимаем выделение
            filesList.SelectedItem = null;

        }
        async void Delete(object sender, EventArgs args)
        {
            // получаем имя файла
            string filename = (string)((MenuItem)sender).BindingContext;
            // удаляем файл из списка
            await DependencyService.Get<IFileWorker>().DeleteAsync(filename);
            // обновляем список файлов
            await UpdateFileList();
        }
        // обновление списка файлов
        async Task UpdateFileList()
        {
            // получаем все файлы
            filesList.ItemsSource = await DependencyService.Get<IFileWorker>().GetFilesAsync();
            // снимаем выделение
            filesList.SelectedItem = null;
        }
    }
}
```

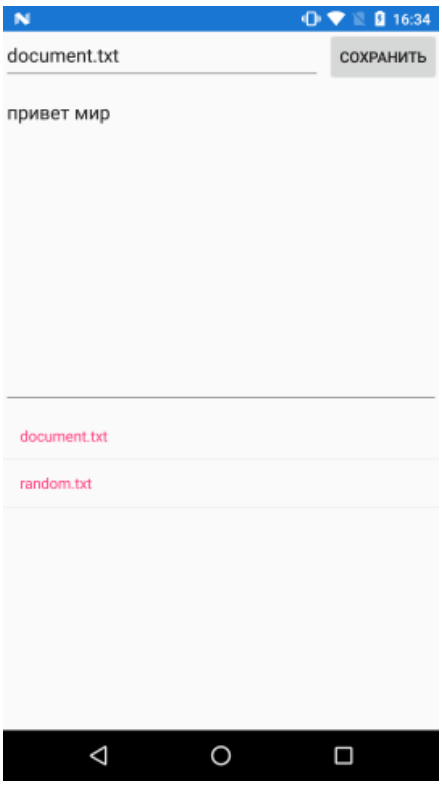
Здесь с помощью метода `DependencyService.Get<IFileWorker>()` вызываем методы, определенные в интерфейсе IFileWorker. При запуске на каждой отдельной платформе будет работать реализация метода, предназначенная непосредственно для данной платформы. И благодаря такому внедрению зависимости данный код вне зависимости от реализации IFileWorker будет работать на любой платформе.

При загрузке срабатывает метод `UpdateFileList()`, который загружает названия всех файлов из папки в список ListView. Исходя из реализаций IFileWorker в качестве папки на Android и iOS будет служить папка MyDocuments, а в Windows 10 - ApplicationData.Current.LocalFolder.

Метод Save - обработчик нажатия кнопки будет проверять наличие файла. Есть файл существует, то отображается диалоговое окно с требованием подтвердить действие. Если файл не существует, то он создается, и в него заносится текст из текстового поля Editor.

Метод `FileSelect()` представляет обработчик выделения элемента в списке ListView. По выделению происходит загрузка текста в текстокое поле Editor.

И метод `Delete()` является обработчиком нажатия на меню. То есть при сильном нажатии на элемент в списке ListView отображается данное контекстное меню из одного пункта, нажатие на которое приводит к удалению выделенного файла.



Назад **Содержание** **Вперед**