

Working with attached to repair documents in the system of automatic registration of repairs

Arseny Zorin



1 INTRODUCTION

Since the appearance of computers, progress isn't standing still. Technology has enormously improved in last decade. Among trends that appear one can find an ambient wireless Internet access to printing prosthesis on 3D printers or cloud storages. The latter became storage de-facto standard for past few years. It is used not only for private, but also for corporates document storing and data archiving.

The object of research is the authorized service center (ASC) that provides warranty and paid repair of household appliances. There are four branches situated in different parts of the city that take equipment for repair. Each repair operation performed by ASC produces an number of attached documents, their names equal to the ticket number:

- A photo of schild from back of equipment. This file is stored in a common folder for schilds' photos of this manufacturer.
- A scanned warranty ticket. This file is stored in a common folder for scanned documents of this manufacturer.
- An outbound repair receipt. Produced in case the repair was carried out at the client's residence. This file is stored in a common folder of outbound repairs for this manufacturer.
- Various acts provided to a client including a technical condition act, a non-repairable act, a replacing device act.

All document folders are stored at branches offices and at the head office server. Data is transmitted between branches and the head office through FTP protocol.

This work was motivated by a need for modernization of tools for documents storage and transmittion. The software programming language was C# instead of VBA. For scanning documents and making photo of them new libraries will be used. All documents storage were moved to the cloud. Data will be stored and treated at the distributed servers in network. While using cloud storages, users have a possibility to access the data from different places and from different devices which have an Internet access. Also cloud storage can prevent losing information in case of failure of a local storage.

2 PRACTICAL IMPLEMENTATION

2.1 Working with AForge.NET

Taking photo of a schild: after the button pressed, a new window with video player element will open. Stream video from selected web-camera will be played on this element. After the button "Take a snapshot" pressed, there will be capture of the picture from video player element and an obtained picture will be stored in the specified directory in .jpg format.

The .NET Framework does not support working with web-cameras it means that there is no video playback element. However, there is namespace AForge.Controls with VideoSourcePlayer element in AForge.NET. This element was created only for using in Windows Forms software. The decision of developing software with using WPF was taken and there are few methods of solution:

- The creation of a hybrid application. In addition to the WPF window, there will be

one Windows Forms window.

- Using a WindowsFormsHost element. This element allows to place Windows Forms element on WPF page.

The latter variant was selected. For its realization we have to add: System.Windows.Forms.Integration and AForge.Controls namespaces, WindowsFormsHost element and VideoSourcePlayer inside it (Fig. 1).

```
xmlns:wfi =
  "clr-namespace:System.Windows.Forms.Integration;assembly=WindowsFormsIntegration"
xmlns:aforge =
  "clr-namespace:AForge.Controls;assembly=AForge.Controls"
<Grid>

  <wfi:WindowsFormsHost HorizontalAlignment="Left" Height="379" Margin="10,41,0,0"
    VerticalAlignment="Top" Width="472">
    <aforge:VideoSourcePlayer x:Name="sourcePlayer" Width="472" Height="379"
      NewFrame="VideoSourcePlayer1_NewFrame"/>
  </wfi:WindowsFormsHost>
```

Fig. 1. Adding namespaces and elements

In addition to the playback element we have to add elements for interaction with VideoSourcePlayer element:

- "Take a snapshot" button
- List of playback sources

After addition of all elements, the window in "Constructor" mode will look like on fig. 2.

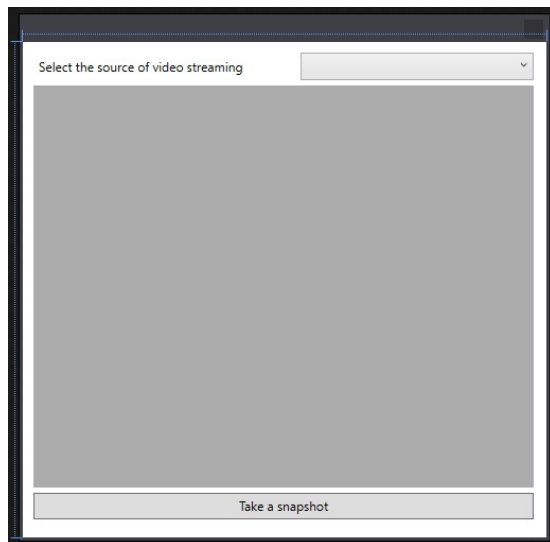


Fig. 2. Window view in the constructor mode

There is the AForge.Video.DirectShow.VideoCaptureDevice class for capturing video from an input-output (I/O) device. It is necessary to set device's moniker (the name of a specific instance of an object. It is a

COM-object that has its own way to create and maintain a special interface - IMoniker), which will capture the picture. In addition, we have to add event handler NewFrame. Every time new frame will be received, this event will raise. Received frame sends to the handler in the Bitmap format. It can be processed and stored in the specified format in the right place.

The list will be filled with names of all connected I/O devices and the default device will be the first in the list.

The event comboBox_Sources_SelectedIndexChanged rises after changing selection of source. It stops streaming from previous I/O device, set a moniker to the VideoCaptureDevice class of a new selected device and resumes streaming.

Pressing the button "Take a snapshot" changes the value of the take_pict boolean variable on true. The value of this variable checks in the event raised after receiving of a new frame. If its value equals to true, then received image saves in .jpg format in specified place. In our case - Yandex.Disk.

The method EndOfWork, that stops streaming and frees used resources, is called after closing the window.

2.2 Working with TWAIN library

We need to program a button for documents scanning. There will be selection of data sources after clicking on the button (if there are several connected scanners). After selecting of the source, scanning will start and received file will be stored at the specific folder in .jpeg format. We have to close data source after finishing the scanning.

There are a lot of different methods in the TWAIN library. However, only several of them will be used for completing the task:

- OpenDataSource() - Opens a data source. Type of return value - bool. Value true if operation is successful, else - false.
- SelectDataSource() - Shows dialog window for selecting a data source. Type of return value - bool. Value true if operation is successful, else - false.

- Property ShowUI - Gets or sets a value indicating whether to show user interface (UI) of TWAIN-source (to speed up value sets as false)
- Acquire() - Gets an image from a data source. Type of return value - void.
- GetImage(int index) - Returns a scanned image. The parameter index - index of return image. Type of return value - System.Drawing.Image.
- CloseDataSource() - Closes a data source. Type of return value - bool. Value true if operation is successful, else - false.

The platform of developed software was changed to 32-bit(x86). The reason is that 64-bit(x64) platform is not supported with all scanners.

2.3 Working with Microsoft.Office.Interop

For beginning the work with Word and Excel documents from Visual Studio, it is necessary to add references into the project: Microsoft.Office.Interop.Word and Microsoft.Office.Interop.Excel.

2.3.1 Working with Microsoft.Office.Interop.Word

The words starting with must be replaced with values of table cells during the creation of acts. Application object, which is the parent of all objects, used for working in the .NET. The work with its methods and properties is available after the reference on this object was received. This object provides a big set of methods and properties that allows to manage the Microsoft Word. Particularly all Word functions require an object parameters.

There are two important moments about working with Word through C#:

- An unmanaged resource is created. It is a separate process in memory with Word application. This process will not be collected with garbage collector. If it will not be closed and showed on the screen it will be remained in the computer memory until turning off. Such processes accumulate invisibly for the user. The programmer must take care of closing the unmanaged resources.

- Word runs invisible by default, we need to display it ourselves.

Despite the latter fact, in our case, Word document will be invisible for the user. However, EndOfWork method will close this process.

Object Range - the main object for working with Word. This object is the region in the document and can include a pair of characters, tables, bookmarks etc. There is Selection object - part of document selected with pointer, that can be converted into the Range. For task execution we have to get Range for the whole document, then to find necessary string in received Range, get Range for this string and inside the latter Range to change the text on required.

There is a rare error in the Microsoft.Office.Interop library that was accepted by Microsoft company. The error is exiting Execute method from Find interface with "Stub received bad data" error. The reason is using Globally Unique Identifiers (GUIDs) in Word that were used in Excel 95. These identifiers begin to specify on Excel library if Excel 95 library was registered later than Word library. It causes to incorrect construction of proxy v-table for out-of-process clients. As a result, queries to that table causes or an error a failure, because instead of sending to the Word they are sending to the Excel.

Standard search and replacing a string with Interop.Word.Find (Fig. 3) interface must be replaced with using library System.Reflection (Fig. 4).

```
wordRange.Find.Execute(item.Key, missingObj, missingObj, missingObj,
    missingObj, missingObj, missingObj, missingObj, missingObj, item.Value,
    replaceTypeObj, missingObj, missingObj, missingObj, missingObj);
```

Fig. 3. Searching and replacing a string

```
var wordFindObj = wordRange.Find;
object[] wordFindParameters =
{
    item.Key, missingObj, missingObj, missingObj,
    missingObj, missingObj, missingObj,
    missingObj, missingObj, item.Value,
    replaceTypeObj, missingObj, missingObj,
    missingObj, missingObj
};
wordFindObj.GetType()
    .InvokeMember("Execute", BindingFlags.InvokeMethod,
        null, wordFindObj, wordFindParameters);
```

Fig. 4. Replaced part of the code

When a work with template is over, it is printing with standard method `PrintOut`. In some cases filled template must be saved. The question about saving is asked.

2.3.2 Working with Microsoft.Office.Interop.Excel

Work with Excel through C# much easier than with Word. We are working with cells instead of Range.

Application object is also used for the work with Microsoft Excel through .NET. And important facts are the same. There are: an unmanaged resource is created and it is invisible by default. In our case, Excel application must be visible. Used resources must be released from the code. However, if document was closed before releasing, it could be an exception which have to be handled in try-catch block.

Working with Excel is the same as working with two-dimensional array.

2.4 Working with Yandex.Disk

The most popular cloud storages were studied and decision to use Yandex.Disk was made. However, according to the overview of different clouds, for office needs the most appropriate are Google Drive or Dropbox. The latter one was not selected because of little amount of free space (only 2Gb). Interaction with our software not only with API but with WebDAV protocol too was the reason for selecting Yandex.Disk.

For beginning, WebDAV API was selected for interaction. With it we can use a cloud storage as an ordinary file system. API compliant to the WebDAV protocol because of it API is compatible with different libraries and clients. Also Yandex has SDK for different popular platforms, including mobile.

OAuth token have to be received for beginning work with Yandex.Disk from software. OAuth is an authorization protocol. According to this protocol, developer have to register his software at the OAuth Yandex Server and requests for access to the data. Authorized user denies or permits access. While using OAuth protocol, user does not enter his password in the software and because of it his account can not be hacked. Received token have to be

sent in Authorization header every time API is used.

Object of `DiskSdkClient` class have to be created and subscribed to its methods because all calls of SDK are asynchronous. Asynchrony in SDK used for providing correct parallel changing of a user interface state from a parallel background thread.

There are methods appropriate to the actions on Yandex.Disk implemented in the SDK. Only used methods are discussed further:

- `GetListAsync` - Requests for content of a directory (paginated list of content can be received with `GetListPageAsync` method)
- `GetItemInfoAsync` - Requests for a file/folder properties
- `MakeDirectoryAsync` - Creates a directory
- `RemoveAsync` - Deletes a file/folder
- `UploadFileAsync` - Uploads a file
- `DownloadFileAsync` - Downloads a file

Received files and folders data stores at the fields of `DiskItemInfo` class:

- `DisplayName` - Name of a file/folder, which should be displayed on the user interface of the software (`OriginalDisplayName` contains name of a file in URL format)
- `FullPath` - The path to a file/folder from the root directory of the user (`OriginalFullPath` contains encoded URL)
- `ContentType` - MIME-type of a file
- `LastModified` - Date and time of the last file modification
- `CreationTime` - Uploading date and time of file on the cloud. MIME-type of a file
- `Etag` - Etag header for a file (MD5-sum)
- `PublicUrl` - External link on a published file/folder
- `ContentLength` - Size of a file/folder (bytes)
- `IsDirectory` - Returns the value that shows whether this element is a directory
- `IsPublished` - Returns the value that shows whether this file/folder is published

Several methods have to be implemented:

- Method which checks an element in the specified directory for existing
- Method which opens selected file. First, it is have to be downloaded

Drawback of the asynchronous method

GetListAsync was found during creation of the method that checks an existing of a file. The background thread which fills list with elements located in the specific directory is not completed before its using because of asynchronous execution. To avoid this drawback it is necessary to wait for the completion of filling the list. Awaiting for filling was implemented with delaying of execution the code with `System.Threading.Thread.Sleep(200)` method which delays the code on 0.2 seconds (Fig. 5).

```
public bool IsItemExist(string path, string itemName)
{
    if (!CurrentPath.Equals("/"))
    {
        InitFolder("/");
        System.Threading.Thread.Sleep(200);
    }

    var items = path.Split('/');
    foreach (var item in items.Where(item => !item.Equals("")))
    {
        if (!FolderItems.Any(elem => elem.DisplayName.ToUpper().
            Equals(item.ToUpper()))) return false;
        var element = FolderItems.First(file => file.DisplayName.ToUpper().
            Equals(item.ToUpper()));

        _sdk.GetListCompleted += SdkOnGetListCompleted;
        InitFolder(element.OriginalFullPath);
        System.Threading.Thread.Sleep(200);
        if (!FolderItems.Any(elem => elem.DisplayName.
            Equals(itemName))) continue;
        InitFolder("/");
        return true;
    }
    InitFolder("/");
    return false;
}
```

Fig. 5. Window view in the constructor mode

However, this solution of the problem is not correct because on another computer recourse to this directory can take more time, in this case list of elements will not be filled correct and it will lead to the incorrect work of the whole method.

For correctly fixing this drawback it is necessary to change all the work of Yandex.Disk SDK. Thus, the decision was made to create the call to the cloud through WebDAV protocol and to liquidate an asynchrony. Class have not got any user interface because of that we can liquidate an asynchrony.

It is necessary to use WebDAV protocol methods to create an interaction with the cloud:

- PUT - Uploads a file
- GET - Downloads a file
- MKCOL - Creating of a directory
- COPY - Copying a file/folder

- MOVE - Movement and renaming of a file/folder
- DELETE - Deleting a file/folder
- PROPFIND - Getting the properties of files/directories
- PROPPATCH - Changing the properties of a file/directory

From listed methods we will need only 4 of them: PUT, GET, MKCOL, PROPFIND.

Classes `HttpRequest` and `HttpResponse`, which included in the .NET Framework from version 2.0, will be used because WebDAV uses HTTP/S. Class `HttpRequest` represents HTTP-request, second - HTTP-response. Namespace `System.Net` for working with network have to be included. It is necessary to generate the request correctly for understanding it by the WebDAV server.

Every method has its own parameters. Some of them are the same for all methods:

- Remote host address (<https://webdav.yandex.ru/>)
- Authentication data. Yandex receives this data in two types: Basic - login and password; OAuth-token. For transmission these data the string is used: `request.Headers["Authorization"] = "OAuth " + AccessToken`.
- The command that has to be executed, e.g. `request.Method="PUT"`

The name of the directory, in which method must be executed, is added to the address of the host in case of execution method in directory other than root.

Creation of a new directory in the cloud, i.e. execution MKCOL method, does not requires any parameters except mandatory. WebDAV-protocol does not allow to create several subfolders in one request. If directory is already exists and request tries to create directory with the same name then exception, that must be handled, rises. To find out does directory exist, we can send a request for creating the directory and handle the exception with code 405 - "Invalid method", which raises in case of creation the directory with the same name.

Requestion the properties -PROPFIND method, does not requires any parameters except mandatory too. However, it is necessary

to parse received response (XML format) from the server. Besides parsing the response, it is necessary to catch an exception with 404 code - "Element not found".

Unlike other methods, PUT and GET requires for an additional parameters, like variables for data reading and writing.

For example, in addition to setting variables, for method GET execution, we will need to get size of a file from the header, create file stream for writing a file on a local drive, receive the stream from the server. The next step is reading the data from received stream and writing them to the open stream. If amount of read bytes are equal to read from the header, then file downloading is succesfull.

For method PUT it is necessary to set parameters:

- AllowWriteStreamBuffering - It enables or disables data buffering before sending it. If it is enable, then file loads to the memory, first, and only after that to the server.
- SendChunked - It allows to upload files of uncertainsize to a remote host
- Expect100Continue - Parameter that includes waiting for a response from the server with code 100. It means that we can continue, in other casse its value will be false.

After completion formation of the query, it is necessary to receive a network flow where the data, which will be sent to the server, written in. We also need to open a local file for reading, allocate byte buffer for temporary storing data read from the file. Then read and send are produced in the cycle and data are written to the flow. Network and file flows are closed after that. HTTP status has to be checked for equality to the flag Created and to compared the size of the file with amount of sendded bytes. Sending was succesfull if both of this conditions are fulfilled, else - there was an error.

After synchronous methods were written, the decision was taken to compare the time of their execution with asynchronous. The Stopwatch class is used for the precise measurement of time. To an object of TimeSpan class the measured time is assigned after measurements were finished. Measurement's results

are presented on the popped up window in hh:mm:ss.ms format (Fig. 5).

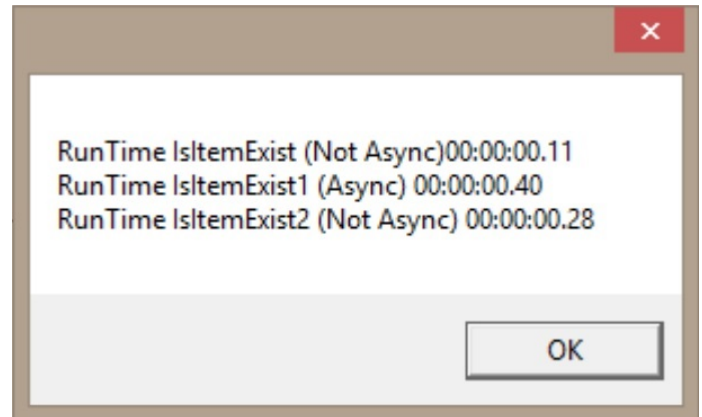


Fig. 6. Measurement's results

According to the results, the best method for our targets is synchronous IsItemExist method.

Except downloading and uploading file to the server, it is necessary to organize the work with it. For example, the file have to be opened if it is an image. This method is realized in the OpenFile method, which downloads a file from the cloud and then opens it with LaunchFile method. We have to make sure that a user is finished working with it to delete a file, for this we show a window with a message. The file closes and deletes if the window was closed.

3 CONCLUSION

The software that allows user to take a snapshot of a document with web-camera and a scanned copy with scanner was developed during execution of this work. An ability of creation different acts based on MS Word and MS Excel templates was orginized. The work with files in the cloud was also orginized.

Each task cold be solved with different ways. The choise of methods depend from targets ans programmer choidse. For example, AForge.NET was selected for making a phostos because of its simplicity in use and easy accessibility. TWAIN library, for working with scanner, was selected because of big amount of scanners are TWAIN-devices. This library is time-tested and reliable besides a lot of TWAIN projects in the Internet. Yandex.Disk was selected mostly because of personal experience.

The problem of lacking the video player WPF element was solved during the photo making task. The WindowsHost element, that allows to include a Windows Forms element into WPF, was used.

The decision to move from 64-bit (x64) on 32-bit(x86) version of software was made, because of 64-bit version is not supported by all TWAIN-compatible scanners.

The work with templates was complicated by an error that was accepted with Microsoft company. To avoid this error, the method, that could be executed with one string, must be divided into three strings. This solution was suggested by Microsoft.

It was determined that the work with Yandex.Disk will not be an optimal due to asynchronous. For the solution this defect, methods of appealing to the cloud server through WebDAV were created. At the end of the work measurements of execution synchronous and asynchronous methods were made. As was shown in the results, in this case, synchronous methods are faster.

We can conclude that all tasks were fully solved. The target of the creation the software as a subsystem for an ASC, that provides an automating of tasks related to the attached documents, was achieved.