# Characteristics of the automation object

## Brief information about the automation object

The object of automation is the user of the operating system, the function of which is to save his data. Data backup is a critical procedure to ensure the security and availability of important user information. Here are a few reasons why data backup is important for the average operating system user.

#### Protection against accidental data loss

People can accidentally delete or damage files without warning. Backup allows you to restore important data in case of loss or damage.

#### Protection against hardware malfunctions

Hard disks, flash drives, and other hardware can fail due to accidental breakdowns or failures. Backup allows you to restore data that was saved on damaged media.

#### Protection against device loss

If your device is stolen or lost, you could lose all your important data. Backing up allows you to save copies of your data elsewhere so that it's available even if your primary device is lost or stolen.

## Description of the processes to be automated

### General provisions

The program for backing up data and virtual safes automates a number of processes for convenient and secure user management. Here is a description of the processes that this program automates:

#### Automatic data backup

The user can set up automatic backup intervals, for example, every time the operating system starts. The program automatically backs up data according to the specified parameters.

#### Keyboard shortcut for quick data transferring

Users can set keyboard shortcuts to quickly transfer data to a virtual safe directly from the program interface.

#### Simplified backups organization

The program allows the user to create virtual safes to store backup copies of data, which simplifies the work with backup and the organization of this backup.

#### Logging of data backup operations

The program automatically keeps a log of data backup operations, allowing the user to track and analyze the actions performed.

### General description of the development stage and its components

The development is divided into the following stages:

#### Project development planning

During this stage, approaches will be evaluated and technologies will be selected for project development. Documentation for the future project will be drawn. Drawing up a theoretical approximate project layout. For example, to figure out the naming convention for this project, etc.

#### GUI development

The main goal of this stage is to create a GUI for the application that will not yet perform any actions related to the file backup functionality, except for the transition between Views and some basic implementations. The result should be а defined, as far as possible, GUI concept that will only need to implement the functionality.

#### Developing library for functionality

Once you have the GUI, you start coding a set of methods that will implement the functioning of the GUI. The result is a set of methods for implementing the project's functionality.

#### Combining GUI and library

At this stage, you need to link the GUI and functional parts by writing classes and services in the GUI that will use library implementations. As a result, you should have a ready-made program that uses the library to function.

#### Software testing

At the end of development, the program is tested and bugs are fixed. The result is a tested program that should ideally be bug-free.

# System requirements

## Requirements for the system in general

The program will consist of a GUI and functional part. The GUI will be coded in the C# programming language using the .NET MAUI Blazor Hybrid technology. After the GUI of the program is created, C++ functions will be added to perform the functionality related to the operation of files. The corresponding C++ functions will be stored in a separate project of the Dynamic-Link Library (DLL) type.

## Requirements for the structure and operation of systems

### Functionality requirements

Below are the requirements for functionality

#### Virtual safe

The program should be able to create virtual safes where data can be stored and backed up. The safe can be a separate local disk, a place on a local disk, or a flash drive.

It should be possible to set up automatic data backup. For example, a file should be backed up when the operating system starts. To simplify backup, it should be possible to create a shortcut to quickly back up files.

Each action on the safes and files in them should be stored in the history, which should be available for viewing.

#### Smartphones and optical disks

The program should be able to download photos and videos from a smartphone and transfer them to one of the safes, as well as record the contents of the safes on an optical disk.

### Naming convention

Requirements for the naming convention are shown in Table 1.

Table 1. List of naming conventions.

|  |  |  |  |
| --- | --- | --- | --- |
| Language | Object | Description | Example |
| C# і С++ | Variable | <Camel Case> | // C# and C++  int myField; |
| C# і С++ | Enum | <Pascal Case> + “Enum” | // C#  public enum MyEnum {}  // C++  enum class MyEnum {}; |
| C# і С++ | Flags enum | <Pascal Case> + “Flags” | // C#  [Flags]  public enum MyFlags {}  // C++  enum class MyFlags : int{}; |
| C# і С++ | Class | <Pascal Case> | // C#  public class MyClass {}  // C++  class MyClass {}; |
| C# і С++ | Struct | <Pascal Case> + “Struct” | // C#  public struct MyStruct {}  // C++  struct MyStruct {}; |
| C# і С++ | Class field | <Camel Case> | // C#  private int myField;  // C++  int myField; |
| C# і С++ | Class method | <Pascal Case> | // C#  public void MyMethod() {}  // C++  void MyMethod() {} |
| C# | Class property | <Pascal Case> | public int MyProperty{ get; set; } |
| C# | Property field | “\_” + <Camel Case> | private int \_myProperty; |
| C++ | Getter, setter | <“get” / “set”> + <Camel Case> | int getMyProperty() const { }  int setMyProperty(int value){ } |
| C# і С++ | Interface | “I” + <Pascal Case> | // C#  public interface IMyInterface{}  // C++  class IMyInterface{}; |
| C# і С++ | Asynchronous method | <Pascal Case> + “Async” | // C#  public async Task MyMethodAsync()  // C++  void myMethodAsync() {} |
| C# і С++ | Model | <Pascal Case> + “Model” | // C#  public class MyClassModel {}  // C++  class MyClassModel {}; |
| C# і С++ | Entity | <Pascal Case> + “Entity” | // C#  public class MyClassEntity {}  // C++  class MyClassEntity {}; |
| C# і С++ | Data transferring object | <Pascal Case> + “Dto” | // C#  public class MyClassDto {}  // C++  class MyClassDto {}; |
| C# і С++ | Request | <Pascal Case> + “Request” | // C#  public class MyClassRequest {}  // C++  class MyClassRequest {}; |
| C# і С++ | Repository | <Pascal Case> + “Repository” | // C#  public class MyClassRepository {}  // C++  class MyClassRepository {}; |
| C# і С++ | Service | <Pascal Case> + “Service” | // C#  public class MyClassService {}  // C++  class MyClassService {}; |
| C# і С++ | Mapper | <Pascal Case> + “Mapper” | // C#  public class MyClassMapper {}  // C++  class MyClassMapper {}; |
| C# | Extension | <Pascal Case> + “Extension” | public static class MyClassExtension {} |
| C# | Command | <Pascal Case> + “Command” | public class MyClassCommand {} |
| C# і С++ | Controller | <Pascal Case> + “Controller” | // C#  public class MyClassController {}  // C++  class MyClassController {}; |
| C# і С++ | ViewModel | <Pascal Case> + “ViewModel” | // C#  public class MyClassViewModel {}  // C++  class MyClassViewModel {}; |
| XAML C# | View | <Pascal Case> + “View” | public partial class MyClassView {} |
| Razor C# | Razor page | <Pascal Case> + “Page” | public partial class MyClassPage {} |

### Programming style recommendations

Below are some programming style recommendations. In the listings, it is advisable to replace the code with a green background with a code with a red background.

#### Number of namespaces per file (C#)

There should be only one namespace per file.

#### Number of classes per file (C#/C++)

There should be only one class per file.

#### Using Types (C#)

Use predefined types, not framework types.

#### var VS target-typed new (C#)

If faced with a choice between using var or new()/new[] in a project, it is better to use new()/new[] as shown in Listing 1. There is no performance difference.

Listing 1. Code style recommendation.

|  |
| --- |
| var person = new PersonModel(); |
| PersonModel person = new(); |

#### “” VS string.Empty (C#)

It is recommended to use string.Empty instead of “”, as shown in Listing 2.

Listing 2. Code style recommendation.

|  |
| --- |
| var str1 = ""; |
| var str2 = string.Empty; |

#### Namespace (C#)

Since only one namespace can be used per file, we use file syntax instead of block syntax, as shown in Listing 3.

Listing 3. Code style recommendation.

|  |
| --- |
| namespace some  {  public class PersonModel  {  private string \_name = string.Empty;  public string Name  {  get => \_name;  set  {  if (!Regex.IsMatch(value, "^[a-zA-Z]+$"))  throw new ArgumentException("Invalid characters in the name");  \_name = value;  }  }  }  } |
| namespace some;  public class PersonModel  {  private string \_name = string.Empty;  public string Name  {  get => \_name;  set  {  if (!Regex.IsMatch(value, "^[a-zA-Z]+$"))  throw new ArgumentException("Invalid characters in the name");  \_name = value;  }  }  } |

#### Method group conversion (C#)

Avoid such expressions in LINQ, but generally use as shown in Listings 4 and 5.

Listing 4. Code style recommendation.

|  |
| --- |
| Action<object> writeObject2 = obj => Console.Write(obj); |
| Action<object> writeObject1 = Console.Write; |

Listing 5. Code style recommendation.

|  |
| --- |
| list.Select(Math.Abs); |
| list.Select(el => Math.Abs(el)); |

#### Switch expression (C#)

It is preferable to use the switch statement as shown in Listing 6.

Listing 6. Code style recommendation.

|  |
| --- |
| int number = 1;  string result;  switch (number)  {  case 1:  result = "One";  break;  case 2:  result = "Two";  break;  case 3:  result = "Three";  break;  default:  result = "Other";  break;  } |
| int number = 1;  string result = number switch  {  1 => "One",  2 => "Two",  3 => "Three",  \_ => "Other"  }; |

#### Index operator (C#)

It is preferable to use the index operator as shown in Listing 7.

Listing 7. Code style recommendation.

|  |
| --- |
| value[value.Length - 1] |
| value[^1] |

#### Range operator (C#)

It is preferable to use the range operator as shown in Listing 8.

Listing 8. Code style recommendation.

|  |
| --- |
| value.Substring(1, value.Length - 2) |
| value[1..^1] |

#### Tuple swap (C#)

It is preferable to exchange values without an extra variable, as shown in Listing 9.

Listing 9. Code style recommendation.

|  |
| --- |
| int a = 5;  int b = 10;  int temp = a;  a = b;  b = temp; |
| int a = 5;  int b = 10;  (a, b) = (b, a); |

#### Null check (C#)

It is preferable to check for null as shown in Listing 10.

Listing 10. Code style recommendation.

|  |
| --- |
| // 1  if (s == null)  {  throw new ArgumentNullException(nameof(s));  }  // 2  if (func != null) { func(args); }  // 3  var v = x == null ? y : x; // or  var v = x != null ? x : y;  // 4  var v == null ? null : o.ToString(); // or  var v != null ? o.ToString() : null; |
| // 1  this.s = s ?? throw new ArgumentNullException(nameof(s));  // 2  func?.Invoke(args);  // 3  var v = x ?? y;  // 4  var v = o?.ToString(); |

#### Pattern matching (C#)

It is recommended to use pattern matching to increase code readability, as shown in Listing 11.

Listing 11. Code style recommendation.

|  |
| --- |
| // 1  return num == 1 || num == 2;  // 2  if (o is int)  {  var i = (int)o;  }  // 3  var s = o as string;  if (s != null) {}  // 4  if (!(o is string s)) |
| // 1  return num is 1 or 2;  // 2  if (o is int i) {}  // 3  if (o is string s) {}  // 4  if (o is not string s) |

#### STL usage (С++)

It is preferable to use standard STL algorithms instead of the imperative style, as shown in Listing 12.

Listing 12. Code style recommendation.

|  |
| --- |
| int sum = 0;  for (int i = 0; i < size; ++i) {  sum += array[i];  } |
| int sum = std::accumulate(array, array + size, 0); |