Masking, Anonymization, De-Identification, & Encryption Library for .NET developers.

Supports .NET 3.5, .NET 4.0, .NET 4.5, & 4.5.1

Index

Masking tools class	Page 3
deIdentify tools class	Page 6
Anonymize tools class	Page 8
Encryption tools class	Page 10
Test Data class	Page 11
SQLite class	Page 12
MS SQL class	Page 14
CSV Tools class	Page 16
MaskingData class	Page 17

public class MaskingTools () basic usage

Basic usage:

```
private string MaskMyUrl(string instr)
{
    Masking Masker = new Masking();
    Masker.CharacterSign = '*';
    Masker.NumberSign = '#';
    string MaskUrl = Masker.MaskURL("https://google.com", true);
    return MaskUrl;
}
```

Above code will replace all occurrences of https://google.com with anonymized urls. If bool is set to false it will mask the occurrences with the standard masking character. The result will be https://######.###

Most methods have one or more overloaded versions. You can opt to return anonymized data instead of masking. Some methods have the option to match parts of the anonymized or masked return values.

Most methods default to masking instead of anonymizing when the CreateFakeData parameter is not passed.

For example:

```
string PhoneNumber(string instr, bool CreateFakePhoneNumber, string MatchAreaCode);

Masking Masker = new Masking();
string  MyPhoneNumber = Masker.MaskPhoneNumber("(613)-123-1234", true,"613")
```

This will return the all anonymized phone numbers with the areacode (613). This will allow you to group data if you pull your data from a database.

The following functions are single string masking only.

```
firstName           (overload to mask a List<> exists)
Surname             (overload to mask a List<> exists)
Date()              (no overload available at this time)

EG. string fname = Firstname("Bob", true) works
    string fname = Firstname("My name is Earl", true) doesn't
```

public class MaskingTools ()

Available methods:

```
char CharacterSign ='$';                                    Character to use for masking characters
char NumberSign = '#';                                      Character to use for masking numbers

string URL(string instr, bool CreateFakeUrl);

string AllText(string instr, bool CreateFakeData);
Masks data in passed string based on all available methods. See example project for details

string CanadianPostalCode(string instr, bool CreateFakePostal);
string CanadianPostalCode(string instr, bool CreateFakePostal, string Match);
string DutchPostalCode(string instr, bool CreateFakePostal);
string ZipCode(string instr, bool CreateFakeZip);
Masks both standard and extended zip codes (90210 & 90210-1234)

string CreditCard(string instr, bool CreateFakeCard);
string CreditCard(string instr, bool CreateFakeCard, string MatchFirst4Digits);

DateTime Date();
Single string masking only

string Date(string instr, bool CreateFakeDate);

string EmailAddress(string instr, bool CreateFakeData);
string mailAddress(string instr, bool CreateFakeData, string MatchUsername);
EmailAddress(in, true, "Bob");

string EmailAddress(string instr, bool CreateFakeData, string Matchdomain, string DomainExtension);
EmailAddress(in, true, "microsoft",".com")

string FirstName(string instr, bool Male);
string FullName(bool Male);
string Surname(string instr);

List<string> FirstName(List<string> Names, bool Male); *)
List<string> Surname(List<string> Names); *)
*) uses List as input and output, only handles one string per list row

string IP4Address(string instr, bool CreateFakeData);
string IP4Address(string instr, bool CreateFakeData, string MatchFirstDigits);
string IP6Address(string instr, bool CreateFakeData);

string Pattern(string instr, string Type);
string Pattern(string instr, string Type, bool CreateFakeData);

string PhoneNumber(string instr, bool CreateFaxPhoneNumber);
string PhoneNumber(string instr, bool CreateFakxNumber, string MatchAreaCode);

string FaxNumber(string instr, bool CreateFakePhoneNumber);
string FaxNumber (string instr, bool CreateFakePhoneNumber, string MatchAreaCode);
```

public class MaskingTools () continued

```
string StringMask (string instr, string maskchar, int CharsToExclude);
Masks complete string with maskchar, with exception of the last X chars (int)


string ScrambleWords(string input)
Scrambles words in passed string

string WordsToCrypticValues(string instr)
Changes words in passed string to cryptic hash values

string SelectedWords(string instr, string[] Words, bool IgnoreCase = false)
string SelectedWords(string instr, List<string> Words, bool IgnoreCase = false)
Masks selected words in passed string from either a string[] or List.

string numbers(string instr, bool CreateFakeData);
Masks or randomizes number in passed strings.



string SIN(string instr, bool CreateFakeSin);
string SIN(string instr, bool CreateFakeSin, string MatchFirst3Digits);

string SSN(string instr, bool CreateFakeSSN);
string SSN(string instr, bool CreateFakeSSN, string MatchFirst3Digits);

string OHIP(string instr, bool CreateFakeOHIPNumber);
string OHIP(string instr, bool CreateFakeOHIPNumber, string MatchFirst4Digits);

string Country(string instr, bool CreateFakeData = false)
string Suffix(string instr, bool CreateFakeData = false)
string Title(string instr, bool CreateFakeData = false)
string Company(string instr, bool CreateFakeData = false)
string City(string instr, bool CreateFakeData = false, List<string> ExternalData=null)
string Password(string instr, bool CreateFakeData=false)

string Age(string instr, bool CreateFakeData = false)
string State(string instr, bool CreateFakeData = false)
string ProvinceAbbr(string instr, bool CreateFakeData = false)
string StateAbbr(string instr, bool CreateFakeData = false)
string Province(string instr, bool CreateFakeData = false)
string Street(string instr)
string Gender(string instr,bool Abbreviation=true, bool CreateFakeData=false)
```

public class deIdentify()

Available methods:

```
string AllText(string instr);
string CanadianPostalCode(string instr);
string CanadianPostalCode(string instr, string Match);
string Country(string instr);
string CreditCard(string instr);
string CreditCard(string instr, string MatchFirst4Digits);
string Date(string instr);
string EmailAddress(string instr);
string EmailAddress(string instr, string MatchUsername);
string EmailAddress(string instr, string Matchdomain, string DomainExtension);
string FaxNumber(string instr);
string FaxNumber(string instr, string MatchAreaCode);
string FirstName(string instr, bool Male);
string FullName(bool Male);
string IP4Address(string instr);
string IP4Address(string instr, string MatchFirstDigits);
string IP6Address(string instr);
string Numbers(string instr);
string OHIP(string instr);
string OHIP(string instr, string MatchFirst4Digits);
string Pattern(string instr, string Type);
string PhoneNumber(string instr);
string PhoneNumber(string instr, string MatchAreaCode);
string SIN(string instr);
string SIN(string instr, string MatchFirst3Digits);
string SSN(string instr);
string SSN(string instr, string MatchFirst3Digits);
string Surname(string instr);
string URL(string instr);
string ZipCode(string instr);

List<string> FirstName(List<string> Names, bool Male);
List<string> Surname(List<string> Names);

string Country(string instr)
string Suffix(string instr)
string Title(string instr)
string Company(string instr)
string City(string instr, List<string> ExternalData=null)
string Password(string instr)

string RandomizeWords(string instr, bool UpperCase = false, List<string> ExternalData = null)
string ScrambleWords(string input, bool UpperCase = false)
string WordsToCrypticValues(string instr, bool UpperCase = false)
```

public class deIdentify() continued

```
string Age(string instr)
string StateAbbr(string instr)
string State(string instr)
string ProvinceAbbr(string instr)
string Province(string instr)
string Street(string instr, bool IncludeStreetNumber = false)
string Gender(string instr, bool Abbreviation = true)
```

public class Anonymize()

Available methods:

```
string AllText(string instr);
string CanadianPostalCode(string instr);
string CanadianPostalCode(string instr, string Match);
string Country(string instr);
string CreditCard(string instr);
string CreditCard(string instr, string MatchFirst4Digits);
string Date(string instr);
string EmailAddress(string instr);
string EmailAddress(string instr, string MatchUsername);
string EmailAddress(string instr, string Matchdomain, string DomainExtension);
string FaxNumber(string instr);
string FaxNumber(string instr, string MatchAreaCode);
string FirstName(string instr, bool Male);
string FullName(bool Male);
string IP4Address(string instr);
string IP4Address(string instr, string MatchFirstDigits);
string IP6Address(string instr);
string Numbers(string instr);
string OHIP(string instr);
string OHIP(string instr, string MatchFirst4Digits);
string Pattern(string instr, string Type);
string PhoneNumber(string instr);
string PhoneNumber(string instr, string MatchAreaCode);
string SIN(string instr);
string SIN(string instr, string MatchFirst3Digits);
string SSN(string instr);
string SSN(string instr, string MatchFirst3Digits);
string Surname(string instr);
string URL(string instr);
string ZipCode(string instr);

List<string> FirstName(List<string> Names, bool Male);
List<string> Surname(List<string> Names);

string Country(string instr)
string Suffix(string instr)
string Title(string instr)
string Company(string instr)
string City(string instr, List<string> ExternalData=null)
string Password(string instr)

string RandomizeWords(string instr, bool UpperCase = false, List<string> ExternalData = null)
string ScrambleWords(string input, bool UpperCase = false)
string WordsToCrypticValues(string instr, bool UpperCase = false)
```

public class Anonymize() continued

```
string Age(string instr)
string StateAbbr(string instr)
string State(string instr)
string ProvinceAbbr(string instr)
string Province(string instr)
string Street(string instr, bool IncludeStreetNumber = false)
```

public class Encryption()

Available variables:

```
string passPhrase;                          Default passphrase for encryption
                                            Defaults to= P@$sphra3E

string saltValue;                           Default Saltvalue for encryption
                                            Defaults to $ALt3d$tr1n90nT0pof1@
```

Change those values in the application that use this library. Using the default values is not recommended practice.

Available methods:

```
bool DecryptFile(string inputFile, string outputFile);
bool DecryptFile(string inputFile, string outputFile, string Key16digitLong);

bool EncryptFile(string inputFile, string outputFile);
bool EncryptFile(string inputFile, string outputFile, string Key16digitLong);

string EncryptString(string data, string algo);
string DecryptString(string data, string algo);

string EncodeString(string input)
string MD5Hash(string strInput);
string SHA256Hash(string instr);
```

Basic usage:

```
private bool EncryptDecrypt()
{
    bool _Status = true;
    Encryption EncFile = new Encryption();

    if (!EncFile.EncryptFile(@"TestData.csv", @"TestData.Encrypted.csv", "1234567890123456"))
    {
        _Status=false;
    }
    if (!EncFile.DecryptFile(@"TestData.Encrypted.csv", @"TestData.Decrypted.csv", "1234567890123456"))
    {
        _Status = false;
    }
    return _Status;
}
```

public class TestData

```
GenerateTestData(string filename, int Records);
```

Basic usage:

```
private void GenerateTestData()
{
    TestData td = new TestData();
    td.GenerateTestData(@"c:\temp\TestData.csv", 1000);
}
```

Generates 1000 rows of anonymized data (divided to 50% male & 50% female)

SQLite class

```
namespace Lionytics.Data.SQLite

void OpenDatabase(String baseName)
void CloseDatabase()
void ExecuteNonQuery(String query)
DataTable ExecuteQuery(String query)


public void SQLiteDBExample()
{
    //  Make sure you have the sqlite dll in the application directory

   try
    {
        SQLite db = new SQLite();
        db.OpenDatabase(@"g:\db\testdata.s3db");
        DataTable table = db.ExecuteQuery("SELECT * FROM table1 WHERE id = 1;");
        db.CloseDatabase();
    }
   catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
```

SQLTools class

```
namespace Lionytics.Data.SQL
```

public class SQLCredentials

```
    {
        public string Server = "";
        public string Database = "";
        public string UserName = "";
        public string Password = "";
        public bool WindowsAuthentication = false;
        public int Timeout = 10;
        public bool Pooling = true;
        public bool Asynchronous = true;
    }
```

public class DBInformation

```
   {
       public string Database = "";
       public string TableName = "";
       public int Timeout = 10;
   }
```

public class SQLTools

```
List<string> GetColumns(string DatabaseName, string TableName, string Delimiter=",");
string GetIndexedColums(string DatabaseName, string TableName, string Delimiter=",");
DataTable ToDataTable(string DataBase, string table, string[] Fields = null);
List<string> GetTables(string DatabaseName);
void BuildSQLConnectionString();
void BuildSQLConnectionString(SQLCredentials InfoClass);
string Escape(string instr, bool RemoveEscapes=false);
List<string> GetStoredProcedures(string DatabaseName, bool ExcludeSystemProcedures=true);
List<string> GetViews(string DatabaseName, bool ExcludeSystemProcedures = true);
```

The following methods have extra attributes not listed in this document;

```
       List<string> ToList(string DataBase, string table);
       bool ExportToFile(string FileName, string DataBase, string table);
```

These attributes are only needed if you want to pass extra parameters. (ctrl+shift+space will expose the extra attributes)

```
void Export(SQLCredentials Credentials, string Filename)
SQLCredentials Import(string Filename)
```

MySQLTools class

```
namespace Lionytics.Data.MySQL


   public class MySQLCredentials
    {
        public string Server = "";
        public string Database = "";
        public string UserName = "";
        public string Password = "";
        public bool ConnectionPooling = false;
        public int MinimumPoolSize=0;
        public int maximumpoolsize = 100;
        public int Port = 3306;
        public bool TableCache = false;
        public bool UseCompression = false;
        public int ConnectionLifeTime = 300;
        public bool WindowsAuthentication = false;
        public int KeepAlive = 10;
        public int ConnectionTimeout = 30;

        public void Export(MySQLCredentials Credentials, string Filename)
        public MySQLCredentials Import(string Filename)

     }

   public class MySQLDBInformation
     {
      public string Database = "";
      public string TableName = "";
     }
```

MySQLTools class (continued)

```
public class MySQLTools:IDisposable
  {
      public string MySQLConnectionString = "";
      public bool IsConnected = false;
      private MySqlConnection _Connection;
      public string LastError = "";
      string BuildMySQLConnectionString(MySQLCredentials Credentials)
      bool MySQLConnect(string ConnectionString=null)
      void MySQLDisconnect()
      bool MySQLConnectionOpen(bool ThrowExecptions=false)
      void MySQLConnectionClose()
      Int64 Count(string Database, string TableName)
      string Escape(string instr, bool RemoveEscapes = false)
      List<string> SelectToList(string Database, string TableName, string[]=null,Int64
      Limit=-1)
      DataTable SelectToDataTable(string Database, string TableName, string[] Fields
      =null, Int64 Limit = -1)
      List<string> GetStoredProcedures(string Database)
      List<string> GetViews(string Database
   }
```

CSV Tools class

```
namespace Lionytics.Data.CSV

public class ReadCSV()
public class WriteCSV
```

Sample Code:

```
using (var reader = new ReadCSV(@"E:\TestData\AnonymizedTestData10K.csv"))
    {
        reader.Delimiter = ',';

        while (reader.ReadRow(Rows))
        {
            Application.DoEvents();
            string[] row = Rows.ToArray();
            string  _row="";
            foreach (string st in row)
            {
            _row+=st+reader.Delimiter.ToString();
            }
            listBox1.Items.Add(_row);
            _row = "";
        }
    }
    MessageBox.Show("CSV File loaded");
```

public class MaskingData

```
public class MaskingData
{
  public void Export(MaskingData MyMaskingData, string Filename)
  public MaskingData Import(string Filename)
}
```

Allows for exporting to XML and importing from XML of masking data used for anonymization of data.

Coding examples