Program for creating invoices in SEPA form

Table of Contents

Purpose of the project :

Being in computer science studies and interested in the world of programming, my desire was to improve my coding skills in order to get ahead and broaden my knowledge. That's why I proposed myself to my network to code eventual programs that could push me to practice self-training and consolidate my knowledge.

Purpose of the programme :

This project is a request from a company programming a solution for a subcontractor. As part of the development of automatic invoicing, I was asked to create a program that, based on information from a database, formulates an invoice in SEPA format.

Development:

What is the SEPA :

As this project was an unconfirmed option on the part of the client, the specifications were very simple: create a program creating an invoice in a ".xml" type file containing all the necessary transfer information.

First of all, I had to find out about SEPA-type bills. These invoices are in the form of tags and give all the information necessary for a bank transfer: BIC, IBAN, name, amount, etc. Here are some interesting links about this format and how it works:

https://fr.wikipedia.org/wiki/Espace_unique_de_paiement_en_euros

https://docs.oracle.com/cd/E39124_01/doc.91/e60210/fields_sepa_pay_file_appx.htm#EOAEL00519

Data Recoveries:

The purpose of this program, as mentioned above, is to retrieve information on the billing of services and to create from this information an invoice in a format that allows the automatic initialization of transfers.

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[SousTraitant](
    [idold] [int] NOT NULL,
    [Client] [varchar](255) NULL,
    [CodeSousTraitant] [int] IDENTITY(1,1) NOT NULL,
    [RaisonSociale] [varchar](255) NULL,
    [Capital] [varchar](255) NULL,
    [NomRepresentant] [varchar](255) NULL,
    [PrenomRepresentant] [varchar](255) NULL,
    [Poste] [varchar](255) NULL,
    [Responsable] [varchar](255) NULL,
    [Adresse] [varchar](255) NULL,
    [ComplAdresse] [varchar](255) NULL,
    [CodePostal] [varchar](255) NULL,
    [Ville] [varchar](255) NULL,
    [VilleImmatric] [varchar](255) NULL,
```

```
// Creer une facture
Virement facture = creerFacture();
```

The data is therefore retrieved on a database, here a monotable with all the information on each invoice and complete this with predefined information that does not change such as the name of the company making the invoice or its IBAN. For reasons of confidentiality and simplicity, I decided to make an automatic data entry in order to focus on the creation of the XML file.

```
Virement facture = new Virement();
Random aleatoire = new Random();
// Information Société Créditée et facture
facture.IdIntern = "NumSoc/NumCB/74/01";
facture.Date = DateTime.Now.ToString("yyyy-MM-ddTHH:mm:ss");
```

Other parameters are filled automatically such as the date and then a loop fills the number of invoices requested with the same information but which will be different upon delivery.

```
for (int i = 0; i <= facture.NbVirmTotal; i = i + facture.Debitee[cpt-1].NbFac)
{
    Debiteur debitee = new Debiteur();
    //Information Société Débitée et motif
    debitee.Bic = "CRLYFRPP";
    debitee.Nom = "Nom Societe Debiteur " + Convert.ToString(cpt + 1);
```

Transfer class :

The transfer class contains all the attributes required to create an invoice in XML format. It contains a ToXML() method that creates the requested invoice file by calling on the one hand the variables contained in these attributes: internal identifier of the invoice file, the invoice number of the company editing the file; and then calls the ToXML() methods of the debit and credit classes that are instantiated as attributes of the transfer class.

```
7 références
class Virement
{
    private string idIntern;
    private Crediteur crediteur;
    private List<Debiteur> debiteurList;
    private string date;
```

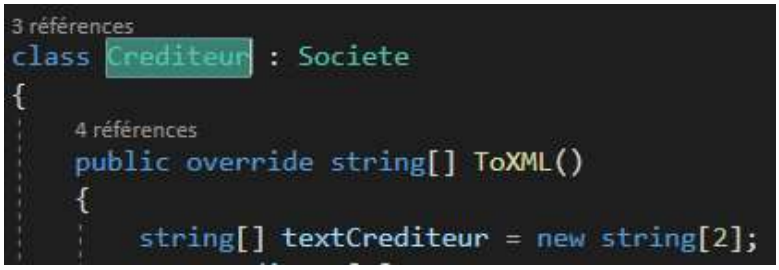
Due to a lot of testing and launching of this program, I had to find a temporary name for the created files in order to recognize them and that they are correctly filed in a file. So the solution is simple: name the file with the time at which it is created. This made it possible to see them displayed in the same folder from the oldest to the newest.

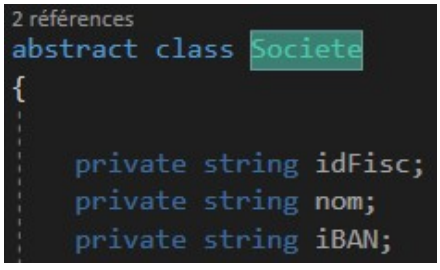
```
facture.Nom = DateTime.Now.ToString("yyyy-MM-dd_HH-mm-ss");
StreamWriter sw = new StreamWriter("FichierXML/" + "Virement_" + facture.Nom + ".xml", false, Encoding.UTF8);
```

2020-02-10_14-45-38.xml	10/02/2020 14:45	Document XML	6 Ko
2020-02-10_14-47-30.xml	10/02/2020 14:47	Document XML	6 Ko
2020-02-10_14-55-17.xml	10/02/2020 16:59	Document XML	6 Ko
2020-02-10_17-03-05.xml	10/02/2020 17:03	Document XML	6 Ko
2020-02-10 17-06-20.xml	10/02/2020 17:06	Document XML	6 Ko

Creditor and Debtor class :

As seen in the previous paragraph, the creditor and debtor classes contain the company information needed for the invoice. These classes inherit the company code class, which defines the attributes of both companies and then adds the data required for the billing document.





The information on the two companies is not used consecutively during the creation of the file, so it was necessary to separate the information into different parts which are therefore stored in an array of string variables. This is why the ToXML() methods of these two classes return arrays of string variables that are used in the method of the transfer class.

The "debtorList" variable is a list of debtors to be included in the invoice. All "boxes" of this list are therefore put in the invoice in turn.

```
string[] textDebiteur = debiteurList[cpt].ToXML();
```

Verification:

Invoicing in SEPA format is a very strict standard and therefore requires template-based verification. Because of the importance of this step, I have taken a large part of it from safe sources such as the file verification on Microsoft documentation, or a site on which the schema of xml type files is given.

```
//verification :
XmlReaderSettings booksSettings = new XmlReaderSettings();
booksSettings.Schemas.Add("urn:iso:std:iso:20022:tech:xsd:pain.001.001.03", "pain.001.001.03.xsd"); // Possible aussi 2pain.001.001.03
booksSettings.ValidationType = ValidationType.Schema;
booksSettings.ValidationEventHandler += new ValidationEventHandler(booksSettingsValidationEventHandler);

XmlReader books = XmlReader.Create("FichierXML/" + "Virement_" + facture.Nom + ".xml", booksSettings);
while (books.Read()) { }
```

https://docs.microsoft.com/fr-fr/dotnet/api/system.xml.schema.xmlschemaset.validationeventhandler?view=netframework-4.8

Example of execution :

```
  Espace de facturation


Le fichier: Virement_2020-04-15_16-36-24.xml a été créé
```

Error messages may be displayed if the program does not run normally, for example if the created file does not conform and does not pass the check.

Conclusion :

The initiative to carry out this program was a good choice because it fully meets the objectives I set myself as a young developer to improve in this field. This project contained a lot of new concepts for me which was beneficial even though its final goal was changed. New basics have been hardened such as creating files in C# or knowing what XML files made of tags are and what their uses are.

Contact :

If you wish to test or see more of this software you can contact me using my professional address: pro@gavazziadrien.fr.