Using scanners with Axapta Document Management

Pedro Rodríguez Parra
pedro.rodriguez@lorca.es

Foreword

I was seeking a way to make more agile the work of introducing documents in Axapta and in the last days I have found a very suitable solution that is both simple and powerful so I am writing this document to share this idea to the rest of the suffered Axapta community that like me are all day struggling with the good ERP but not so good in the documentation side…

Introduction

So, in what consist this project? This project is done using three tools. First I use the Document Management tables of Axapta, second I use COM Interop in Axapta and third I have developed a .NET class library that use a .NET SDK that let you to use easily any TWAIN compliant scanner.

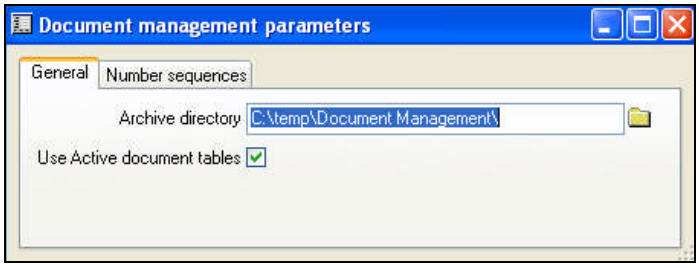
I will first show how it works and later I will give all the details needed to do something similar and surely better!

Example of use

Here are the basic steps needed to scan documents directly to the Document Management module of Axapta.

1. We have to configure the Document Management Parameters if we have not done it yet.

   1.1. We have to go to **Basic, Setup, Document Management, Parameters**.

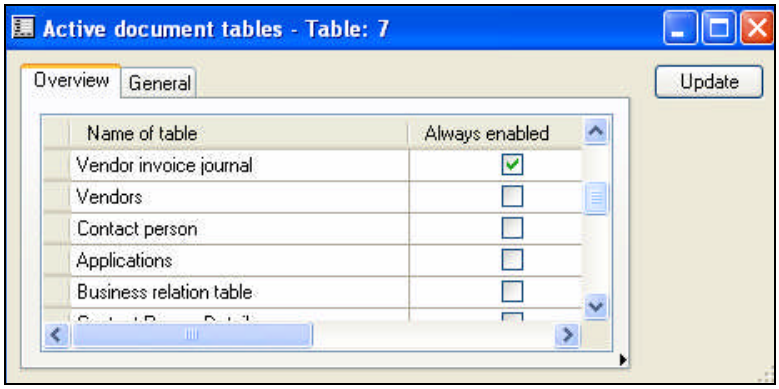


   1.2. We have to introduce the PATH where our documents will be stored in the **Archive Directory** field.

   1.3. We have to activate the **Use Active Document Tables** if we want to store documents for only some tables like *CustTable, VendTable* and so on.

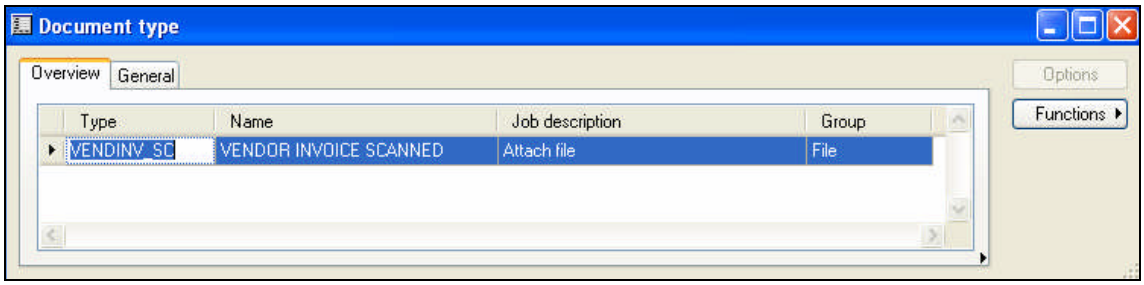
1.4. We also have to create a **Number Sequence** for the Document Management in the Number Sequences tab.

1.5. We have to go to **Basic, Setup, Document Management, Active Document Tables.**



Here we configure what tables are Document Management "enabled". In this example I will need at less the Vendor Invoice Journal Table.

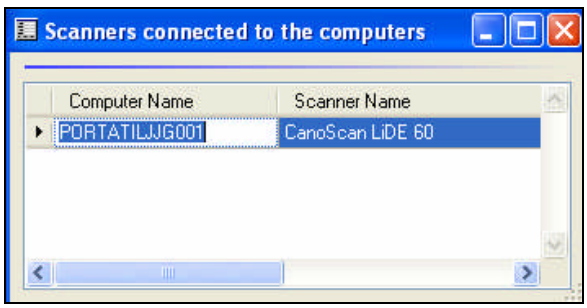
1.6. We have to go to **Basic, Setup, Document Management, Document Types.**



Here we create the type of documents that we are going to associate to our tables in the Document Management.

For this example I need a **type** called "VENDINV_SC", with **Job Description** "Attach file" and **Group** "File".

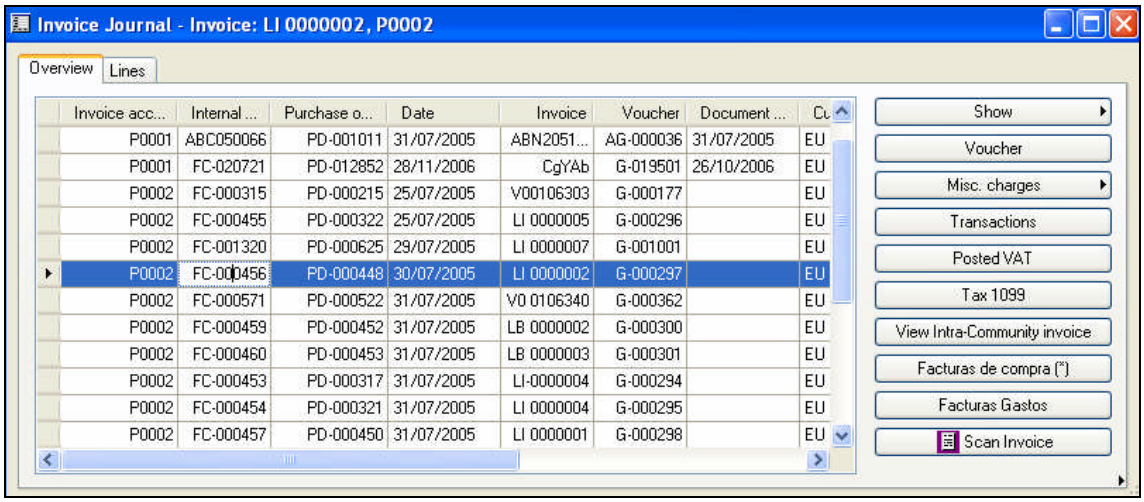
1.7. I have created an own table & form on **Basic**, **Setup**, **Document Management**, **Scanners**.



I use this table to get the name of the scanner connected to the computer where the Axapta client is executing so the users don't have to be indicating the name of the scanner every scan they do.

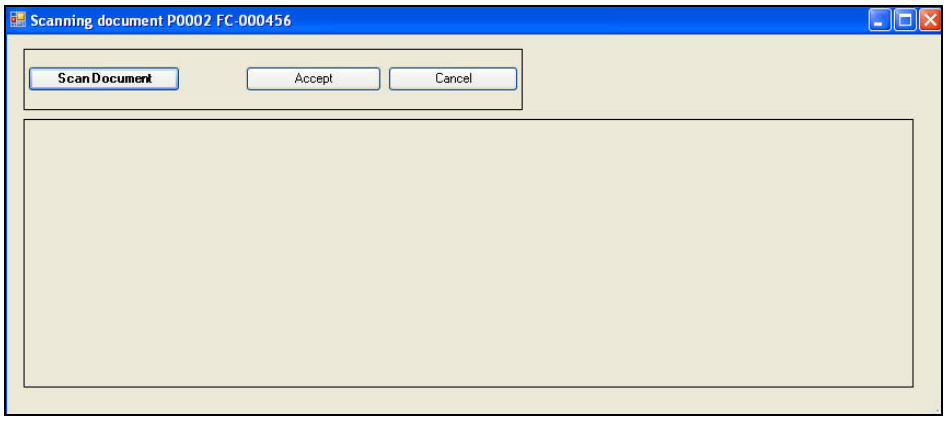
2. Finally we have to scan a document. For example a Vendor Invoice and associate it with a record of the *VendInvoiceJournal* Table.

2.1. We can go to the Vendor Invoice Journals form where I have put a sample button to scan vendor invoices. (**Purchase Ledger, Inquiries, Journals, Invoice**).



2.2. We select a vendor invoice and press the **Scan Invoice** button.

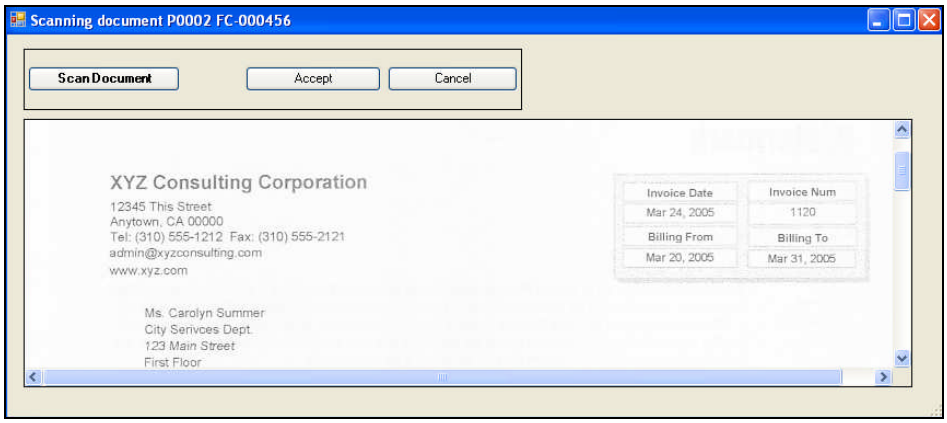
In this moment appears the .NET application developed to scan the documents.



2.3. We press the Scan Document button and the application starts scanning the document using the scanner connected to the computer.

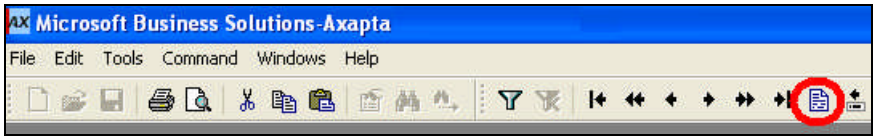
The application knows the name of the scanner to use because we indicated it in the step 1.7. If the name of the scanner is not found we will show a dialog where we will have to select one of the TWAIN devices that are connected in our computer.

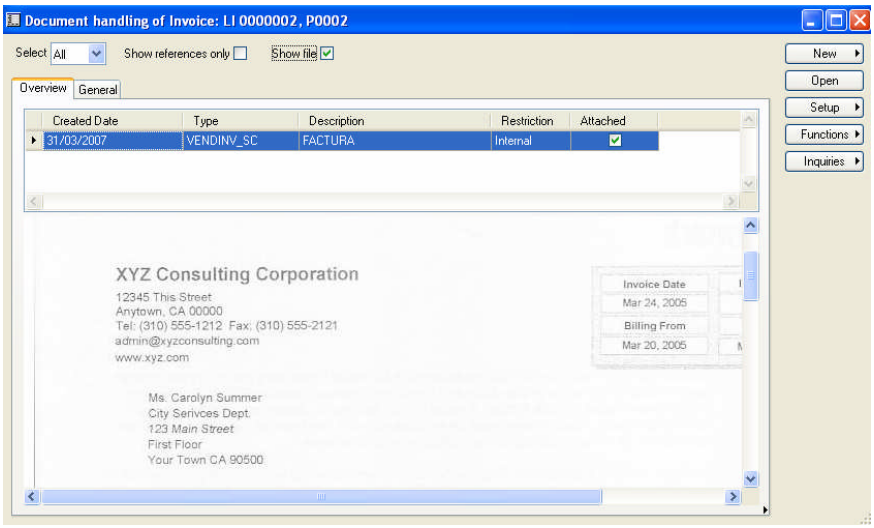
2.4. After the scanning process we will see the result in the preview screen.



2.5. If the scan is OK we press the **Accept** button and the scanned invoice will be stored in the Document Management.

2.6. Now that the invoice is store in the Document Management we will can see it selecting the Invoice Journal record and later pressing in the **Document Handling** in the Tool Bar.





2.7. We also can open the document pressing in the **Open** Button.

And this all what is needed to scan, store and associate documents with records in our Axapta system. Simple, isn't?

In the following pages I will describe how is all done.

.NET SDKs FOR TWAIN SCANNERS

The first thing needed to do a project like this is to identify a suitable .NET SDK which will let us to control our TWAIN scanner in a easy way through a DLL Class library that we will use from Axapta.

During this project I have evaluated 2 SDK:

- **AtalaSoft DotTwain 5.0**

  This SDK cost about $399 and has a good documentation and support forum site. This seem the most professional and can be purchased like part of a more big Imaging SDK pack.

- **VintaSoft VintaSoftTwain.NET library**

  This SDK cost about 59$ and has the right functionality needed to make a project.

I have tried both and I have chose **VintaSoft SDK** because it has the functionality I just need and is by far the cheaper one.

THE .NET CLASS LIBRARY DEVELOPED

Using VintaSoftTwain.NET library I have created a new Class library project in Visual Studio 2005 with a Class named **`scannerTwain.ScannerTwain`** in it.

This class has the following method:

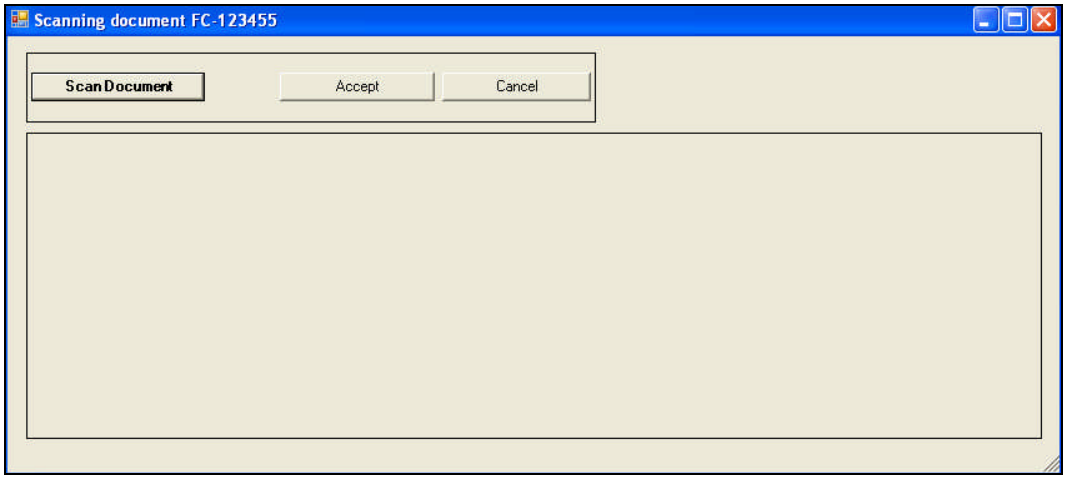
```
public int scanDocument(String documentName,
                                String scannerName,
                                int resolution,
                                String fileName)
        {
            this.documentName = documentName;
            this.scannerName = scannerName;
            this.resolution = resolution;
            this.fileName = fileName;

            formMain.Text = String.Format("Scanning document {0}", this.documentName);
            formMain.ScannerTwain = this;
            formMain.ShowDialog();

            return returnValue;
        }
```

As we can see I pass by parameters the identify name of the document, the name of the scanner to use, the resolution to use and the final filename where I want my scanned document stored.

In this method I simple show the main form and I wait for the form to be closed.



When we press in the Scan Document button the form call the **`startScan`** method of the class **`scannerTwain.ScannerTwain`**.

```
public void startScan()
        {
            vsTwain = new VSTwain();
            vsTwain.PostScan += new
VintaSoft.Twain.VSTwain.PostScanEventHandler(this.VSTwain_PostScan);
            vsTwain.StartDevice();

            //selecting the Twain device
            Boolean sourceFinded = false;
            for (int i = 0; i < vsTwain.sourcesCount; i++)
            {
                    if (scannerName.Equals(vsTwain.GetSourceProductName(i)))
                    {
                        vsTwain.sourceIndex = i;
                        sourceFinded = true;
                        break;
                    }
            }

            if (!sourceFinded)
                vsTwain.SelectSource();

            vsTwain.showUI = false;
            vsTwain.disableAfterAcquire = true;

            vsTwain.OpenDataSource();
            vsTwain.unitOfMeasure = VintaSoft.Twain.UnitOfMeasure.Inches;
            vsTwain.pixelType = VintaSoft.Twain.PixelType.GRAY;
            vsTwain.resolution = this.resolution;
            vsTwain.brightness = 400;
            vsTwain.contrast = 0;
            vsTwain.pageSize = VintaSoft.Twain.PageSize.A4;

            vsTwain.Acquire();
        }
```

In this method we can see the functionality provided by the .NET TWAIN library chose. We soon realise how easy is to control a TWAIN scanner with one of these libraries.

In this method I just initialize the TWAIN handler class **vsTwain**, I add my event handler for the **PostScanEvent** (it will be the method `VSTwain_PostScan`), I select the Twain device or I let it to show me the TWAIN devices available, and I initialise several parameters for the scanning process like the resolution, the page size and so on.

Finally I call the `Acquire` method and the scanner will scan my document.

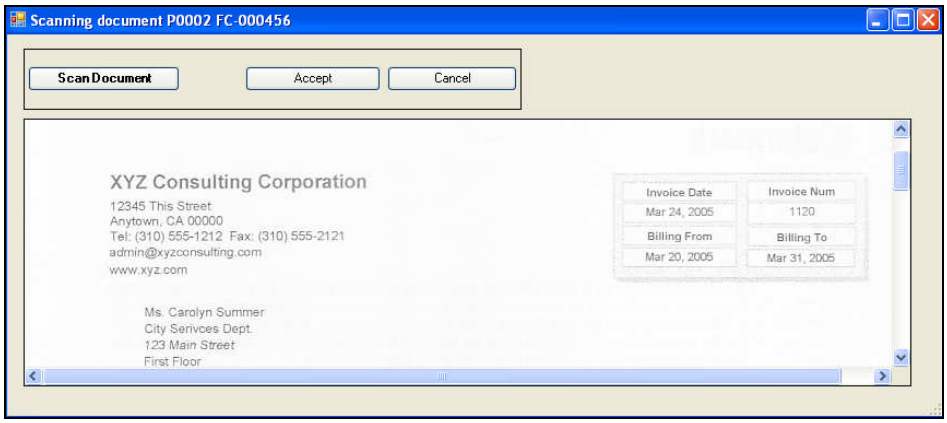
When the scan process is completed the **PostScanEvent** is raised and my method `VSTwain_PostScan` will be called.

```
private void VSTwain_PostScan(object sender, VintaSoft.Twain.PostScanEventArgs e)
        {
            if (!e.Flag)
            {
                if (vsTwain.errorCode != 0)
                {
                    MessageBox.Show(vsTwain.errorString);
                }
            }
            else
            {
                formMain.setImage(vsTwain.GetCurrentImage());
            }
            vsTwain.showUI = false;
        }
```

Here I only put the scanned image in a **PictureBox** control on the form.

```
public void setImage(Image image)
        {
            if (this.pictureBox.Image != null)
            {
                pictureBox.Image.Dispose();
                pictureBox.Image = null;
            }

            this.pictureBox.Image = image;
        }
```



If all is correct we press in the **Accept** button and the Image will be saved like a JPG image with the filename and path indicated in the parameters.

```
private void buttonAceptar_Click(object sender, EventArgs e)
        {
            if ((this.scannerTwain != null))
            {
                if (this.pictureBox.Image == null)
                {
                    MessageBox.Show("No hay ninguna imagen que guardar!","Error");
                }
                else if (this.scannerTwain.saveImage(this.pictureBox.Image))
                    this.scannerTwain.scanFinished(true);
            }
        }
```

```
public Boolean saveImage(System.Drawing.Image image)
        {
            Boolean result = true;
            System.Drawing.Imaging.ImageFormat saveFileType;

            saveFileType = System.Drawing.Imaging.ImageFormat.Jpeg;
            try
            {
                image.Save(this.fileName, saveFileType);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error al guardar la imagen. " + ex.Message);
                return false;
            }

            return result;
        }
```

Finally the **scanFinished** method closes the form and makes the method **`scanDocument`** to return a value of 1 (Correct scanning and storing of image).

REGISTERING OF THE CLASS LIBRARY DLL

In order to be able to use our new DLL Class library from Axapta we have to register it for COM Interop.

1. In the Assembly info of our Class library project we have to modify the COM Visible property to true.

```
// Setting ComVisible to false makes the types in this assembly not visible
// to COM components.  If you need to access a type in this assembly from
// COM, set the ComVisible attribute to true on that type.
[assembly: ComVisible(true)]
```

2. In the Visual Studio 2005 project properties we have to indicate that we want to **sign** our DLL with a **strong name key file** in the **Signing** section.

3. Finally in the **Build** section of the project properties we activate the "**Register for COM Interop**".

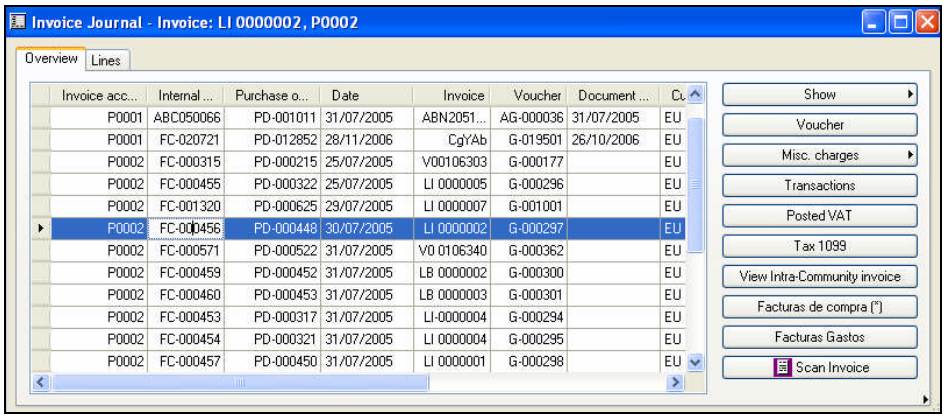
   If we move our generated DLL to another computer and we want to use it with the Axapta client we have to use this command to Register the DLL like COM Interop:

   Regasm myAssembly.dll /codebase /tlb

THE AXAPTA PROJECT

Once our DLL Class library is build and registered in the system we are ready to use it from our Axapta client.

As we have seen in the example, the scanning process starts when the user press on the **Scan Invoice** button added on the **VendInvoiceJournal** form.



When the user press in the button the method **scanVendInvoiceJour** is called on the **JJ_EscaneadoDocumentos** class created for this project:

```
/*
  scanVendInvoiceJour
  Params:
    VendInvoiceJour VendInvoiceJour

  Scan a vendor invoice, store the scanned file on the Document Management path
  and creates the needed records in the Document Management tables so the scanned
 document will be associated with the VendInvoiceJournal record.
*/
public void scanVendInvoiceJour(VendInvoiceJour VendInvoiceJour)
{
  //file name
  Str fileName;
  //absolute path of the file
  Str filePath;
  //result of the scan process
  int resultScan;
  //COM object used to scan
  COM myObject = new COM("scannerTwain.ScannerTwain");
 //record of an own table that contains a relations of computers and scanners
  JJ_DocuScanners JJ_DocuScanners = JJ_DocuScanners::find(Winapi::getcomputername());
  ;

  if (!vendInvoiceJour)
    throw error('You need to pass a VendInvoiceJour record like parameter.');

  //File name
  //Número de factura - Id de Usuario actual - Fecha Actual.JPG
  fileName = strfmt("%1-%2-%3.JPG",
                    vendInvoiceJour.InternalInvoiceId,
                    CurUserId(),
                    date2str(systemdateget(), 123, 2, 3, 2, 3, 4));

  //Absolute file path
  //Document Management path + fileName
  filePath = this.archivePath( fileName);
```

```
//we call the scanDocument method of the .NET Class library
 resultScan =  myObject.scanDocument(strfmt('%1 %2',
                    VendInvoiceJour.OrderAccount,
                    VendInvoiceJour.InternalInvoiceId), //descripción
                JJ_DocuScanners.ScannerName, //Escaner a utilizar
                100,  //Resolución
                filePath);//ruta del fichero


 if (resultScan)
 {
   //if the document si scanned and stored correctly we store it in the
   //Document Management tables
   this.writeDocuValue(
                int2str(VendInvoiceJour.RecId),
                'VendInvoiceJour',
                'VENDINV_SC',
                "JPG",
                "FACTURA",
                filePath);
 }
}
```

In this method first I initialize the COM object so I can call the .NET Class library methods:

```
COM myObject = new COM("scannerTwain.ScannerTwain");
```

The filename will be “Internal Invoice Id” + “Current User id” + “Current Date”.

I use the filePath method to concatenate the filename with the Document Management Path in order to get the absolute path where the file will be stored:

```
/*
  archivePath

  Gets the absolute path of the file in the Document Management module.

  Returns
  Document Management path + file name
*/
private str archivePath(String fileName)
{
str filePath = DocuParameters::find().archivePath;
;
 if (! filePath)
  throw error("@SYS62843",
              '',
              SysInfoAction_formRun::newFormname(formstr(docuParameters),
              identifierstr(Archive_ArchivePath),"@SYS4157"));

 filePath = Docu::fileCheckPath(filePath);
 if (! winapi::pathExists(filePath))
  throw error("@SYS62844",
              '',
              SysInfoAction_formRun::newFormname(formstr(docuParameters),
              identifierstr(Archive_ArchivePath),"@SYS4157"));

 filePath = filePath + filename;

return filePath;
}
```

I use the COM object in order to call the **scanDocument** method created in the .NET Class Library.

```
//we call the scanDocument method of the .NET Class library
  resultScan =  myObject.scanDocument(strfmt('%1 %2',
                        VendInvoiceJour.OrderAccount,
                        VendInvoiceJour.InternalInvoiceId), //description
                  JJ_DocuScanners.ScannerName, //Twain scanner to use
                  100,  //Resolution
                  filePath);//Absolute file path
```

If resultScan has the value 1 (correct scanning and storing of file) I call the method WriteDocuValue in order to create the records in the Document Management tables and associate the VendInvoice current record with the document.

```
/*
  writeDocuValue

  Create the DocuValue and DocuRef records needed to store the scanned document
  in the Document Management and associate it with the VendInvoiceJournal record.

  PARAMS
  -refRecId
        RecId of the VendInvoiceJournal record.
  -tableName
        Name of the VendInvoiceJournal table.
  -_typeId
        Type of document we are going to generate in the Document Management.. (Table DocuType, Field TypeId)
  -fileType
        File extension of the file we want to link with the Document Management records. (JPG, PDF, etc.)
  -name
        Description of the Document Management record.
  -filePath
        Absolute path of the file we want to store in the Document Management.
*/
private void  writeDocuValue(
                  str refRecId,
                  TableName tableName,
                  str _typeId,
                  str fileType,
                  str name,
                  str filePath)
{
 DocuValue docuValue;
 DocuRef docuRef;
 Dictionary dict;
;

 dict = new Dictionary();

 ttsbegin;
 docuValue.clear();
 [docuValue.fileName, docuValue.fileType, docuValue.path] =
        Docu::splitFilename(filePath);
 docuValue.Name = name;
 docuValue.insert();

 docuRef.clear();
 docuRef.Name = name;
 docuRef.TypeId = _typeId;
 docuRef.ValueRecId = docuValue.RecId;
 docuRef.RefTableId = dict.tableName2Id(tableName);
 docuRef.RefRecId = str2int(refRecId);
 docuRef.insert();

 ttscommit;
}
```

And that is all! We have scanned a document and associated it with a record or the **VendInvoiceJournal** Table so we can see the document easily in the Document Management module of Axapta.

SCANNERS TESTED

At first the every TWAIN SDK should work will most scanners without major problems because the TWAIN interfaces seen to be well defined. For this project I have tested the SDKs with a **Canon Scan LiDE 60** and a **HP Photosmart C380** and they work fine.

Pedro Rodriguez is the chief engineer of the Spanish cattle company Juan Jimenez García, S.A. and is working with Axapta 3.0 since 2 years. He is also interested in .NET and Java development especially in mobile projects with J2ME.
pedro.rodriguez@lorca.es