Documenten genereren vanaf je desktop

Door combinatie van technieken ook in SharePoint 2010

Remco van Beek en Gustavo Velez

Een van de meest aansprekende nieuwe elementen van SharePoint 2010 voor ontwikkelaars is de mogelijkheid om software te laten werken vanaf de desktop. In SharePoint 2007 moesten applicaties nog fysiek op SharePoint servers draaien maar in 2010 is dat niet meer nodig.

Door gebruik te maken van Open XML, het documentformaat van Office 2007 en 2010, en de combinatie van bovenstaande technieken van SharePoint 2010, kun je op de client documenten generen. Denk bijvoorbeeld aan rapporten en brieven. Hiermee krijgt de gebruiker andere mogelijkheden die niet van oorsprong aanwezig zijn in het systeem.

Een probleem van SharePoint 2007 is dat je het Object Model niet op afstand kunt gebruiken. Zelfgemaakte uitbreidingen moet je dus altijd op SharePoint servers installeren, tenzij je de webservices gebruikt. Met het Client Object Model introduceert SharePoint 2010 objectgeoriënteerde interactie met SharePoint-data vanaf een niet-SharePoint computer.
Je kunt een SharePoint 2010 server benaderen middels vier objectmodellen: Client Object Model, Silverlight, ECMAScript en REST. Het Client Object Model is een .NET managed programmeermodel dat een subset aan instructies van het Server Object Model gebruikt om componenten van SharePoint aan te spreken. Hiermee schrijf je .NET code die een soortgelijke syntax gebruikt als het Server Object Model om informatie van Site Collections, Sites, Lists en ListItems te raadplegen en manipuleren. Om de beveiliging van de server te kunnen waarborgen zijn bepaalde administration classes echter niet beschikbaar.

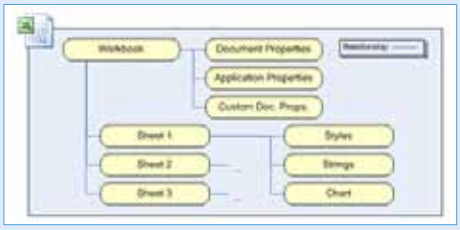
Open XML: werken met Office files

Het Office Open XML formaat beschrijft de bestandsstructuur die door Microsoft Office Word, Excel en PowerPoint wordt gebruik om bestanden op te slaan. Het formaat definieert een set van XML syntax en de bijbehorende bestandsstructuur om deze in op te slaan die gezamenlijk de inhoud van elke van deze documenttypes vormen. Open XML is een door ECMA en ISO/IEC erkende standaard en Microsoft heeft software ter beschikking gesteld om deze programmatisch te kunnen verwerken.
In feite is een Office .docx (Word), .xslx (Excel) of .pptx (PowerPoint) file niets anders dan een .zip file met een aantal directories en bestanden. Deze bestanden bestaan uit de XML met de inhoud van het document en overige resources zoals plaatjes en figuren. De extensie van elk van deze files kun je naar .zip veranderen en unzippen. De Open XML SDK bied je de hulpmiddelen om het proces van openen, lezen van informatie, modificeren en opnieuw comprimeren in software te automatiseren. Figuur 1 laat bijvoorbeeld de interne structuur van een Excel file zien.

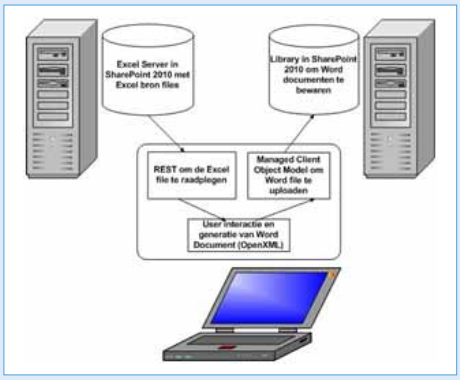
Twee processen

Als voorbeeld van het gebruik van de nieuw mogelijkheden van het SharePoint 2010 Client Object Model in combinatie met Open XML, nemen we een scenario waarbij twee processen binnen de organisatie ongeveer dezelfde informatie nodig hebben. Echter voor één van deze processen moet de informatie in een Excel sheet worden aangeleverd, terwijl voor het andere proces de informatie in Word moet worden aangeleverd. Voor het gemak gaan we in dit voorbeeld uit van een scenario met één server. Het is slechts een kwestie van configuratie om de oplossing ook over meerdere servers te laten werken. Het verbindingselement is een Windows programma gebaseerd op de Windows Presentation Foundation, dat de verschillende onderdelen van het spreadsheet zal tonen. Het Windows programma zal aan de hand van de gekozen informatie een Word-document genereren en deze in een SharePoint-library opslaan.

Figuur 2 laat het proces zien: aan de linkerkant bevindt zich de SharePoint server met de Excel databron, en aan de rechterkant zijn evenknie om het Word-document te bewaren. De client applicatie draait op een lokale computer. Deze maakt verbinding met de eerste server door middel van het Client Object Model en Excel-REST en bewerkt de ontvangen data door de User Interface. Vervolgens genereert het een Word-document met gebruik van Open XML en stuurt het document naar de tweede server. Figuur 3 toont een voorbeeld van een gegenereerd Word-document.
Als je dit zelf wilt doen, heb je Visual Studio en Open Xml nodig. In dit voorbeeld gebruiken we de december 2009 CTP versie van Open XML SDK 2.0 for Microsoft Office. Je start een nieuwe solution en maakt drie referenties naar DocumentFormat.OpenXml.dll , Microsoft.SharePoint.Client.dll en Microsoft.SharePoint.Client.Runtime.dll.
Voor dit voorbeeld is een Excel file 'Sales.xlsx' gemaakt, met twee grafieken en een tabel, zie Figuur 4. Namen zijn toegekend voor



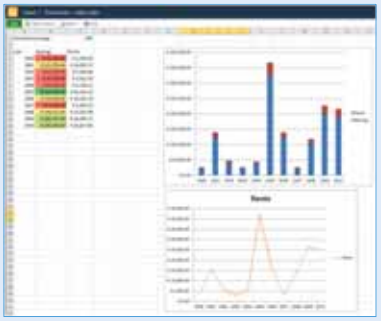
FIGUUR 1: DE INTERNE STRUCTUUR VAN OPEN XML.



FIGUUR 2: SCHEMATISCHE WEERGAVE VAN HET VOORBEELD.

de verschillende onderdelen: voor de tabel 'RenteTabel', en 'RenteOverzicht' en 'GrafiekRente' voor de grafieken.
Figuur 5 toont de WPF User Interface met diverse controls voor de data uit SharePoint. Door middel van het Client Object Model van SharePoint worden in eerste instantie alle lijsten getoond en afhankelijk van de geselecteerde lijst wordt de inhoud in een listbox opgesomd en een geselecteerd item geeft een overzicht van de objecten in de Excel File. Door het selecteren van een object zoals een grafiek worden gegevens via Excel-REST opgehaald en getoond in een Image control.

FIGUUR 3: RESULTAAT VAN HET GEGENEREERDE WORD-DOCUMENT.



FIGUUR 4: EXCEL FILE SALES.XLSX.

Lijsten ophalen

In codefragment 1 worden de Web en ListCollection objecten geladen met behulp van het SharePoint Client Object Model, waarna door middel van de ExecuteQuery methode de gegevens daadwerkelijk worden opgehaald uit SharePoint. Een loop laadt alle lijstnamen in de control op scherm.

```
using (ClientOM.ClientContext context = new    ClientOM.
ClientContext(txtURL.Text))
{
// context naar web
                ClientOM.Web teamWeb = context.Web;
                // opstellen van list collectie binnen het huidige
 web
                ClientOM.ListCollection lists = teamWeb.Lists;
                context.Load(lists);
                context.ExecuteQuery();
                lstLists.DataContext = lists;
                lstLists.Items.Clear();

// voor elke aanwezige list een item toevoegen aan een listbox
                foreach (ClientOM.List item in lists)
                {   lstLists.Items.Add(item.Title.ToString());  }
            }
```

CODEFRAGMENT 1: OPHALEN LIJSTEN UIT SHAREPOINT.

Als een lijst geselecteerd is, wordt het lstList_SelectionChanged event gevuurd en de items van de lijst ophaalt. Eerst laadt je een object met de waarden van de lijst: de LoadQuery van het clientContext object accepteert hier een LINQ query als parameter, die zorgt dat alleen de lijst met de geselecteerde naam wordt geladen.

```
           using (ClientOM.ClientContext clientContext = new
           ClientOM.
ClientContext(txtURL.Text))
            {
                // ophalen geselcteerde lijst
                string selectedListName = e.AddedItems[0].
ToString();
                ClientOM.List selectedList;
                ClientOM.Web teamWeb = clientContext.Web;
                ClientOM.ListCollection lists = teamWeb.Lists;


IEnumerable<ClientOM.List> resultLists = clientContext.
LoadQuery(lists.Include(
                    list => list.Title,
                    list => list.Description,
                    list => list.RootFolder,

list => list.Id).Where(list => list.RootFolder.Name == selectedList
Name));
                clientContext.ExecuteQuery();
```

CODEFRAGMENT 2: OPHALEN GESELECTEERDE LIJST.

Om de listitems uit de lijst op te halen is helaas een CAML Query noodzakelijk. Ook hier worden de objecten geladen, geïnitieerd en met de ExecuteQuery verzonden richting SharePoint. Omdat de in de vorige paragraaf beschreven code slechts een lijst teruggeeft, is het gebruik van de methode 'First()'wel op zijn plek.

```
                // natuurlijk afvangen of we een lijst
terugkrijgen
                if (resultLists.Count() == 1)
                {

// we verwachten maar een lijst dus First is hier op zijn plek
                    selectedList = resultLists.First();
                    ClientOM.CamlQuery query = new ClientOM.Caml
Query();

 // jammer genoeg geen mogelijkheid tot gebruik maken Linq to
 SharePoint, dan maar ouderwets een Caml query.
```

```
query.ViewXml = "<View><Query><Where><BeginsWith><FieldRef
Name='ContentTypeId'/><Value Type='ContentTypeId'>0x0101</Value></
BeginsWith></Where></Query></View>";

ClientOM.ListItemCollection listitemCollection = selectedList.
GetItems(query);
                    clientContext.Load(listitemCollection);
                    clientContext.ExecuteQuery();
                    if (listitemCollection.Count != 0)
                    {
                        lstItems.Items.Clear();

foreach (Microsoft.SharePoint.Client.ListItem listitem in
listitemCollection)
                        {

lstItems.Items.Add(listitem["FileLeafRef"].ToString());
                        }
                    }
```

CODEFRAGMENT 3: LISTITEMS VERZAMELEN.

Objecten uit Excel verkrijgen

Dan komen we uiteindelijk bij het vinden van de juiste informatie in Excel. Een goed hulpmiddel is de 'Open XML SDK 2.0 Productivity Tool CTP for Microsoft Office'-tool, die inzicht geeft in de structuur van Open XML bestanden en een reflectie van de code kan geven (zie hiervoor Figuur 6).
In dit voorbeeld worden de 'Defined Names' en grafiek namen uit Excel opgehaald. Om tot de informatie te komen, wordt de Excel file die geselecteerd is geopend met behulp van de functie 'OpenBinaryDirect' en wordt de 'FileInformation.Stream' door middel van een routine gekopieerd naar een 'MemoryStream' (zie codefragment 4 en 5).

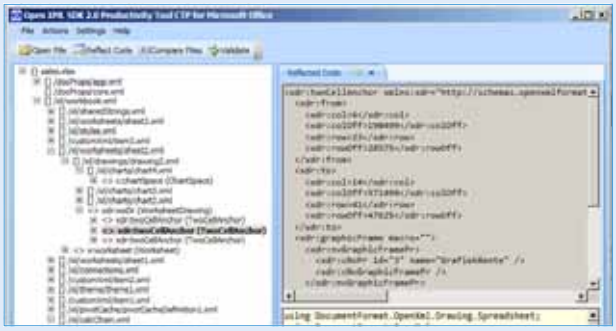
```
            using (ClientOM.ClientContext clientContext = new
            ClientOM.
ClientContext(txtURL.Text))
            {

ClientOM.List selectedList = clientContext.Web.Lists.
GetByTitle(lstLists.SelectedItem.ToString());
                ClientOM.CamlQuery q = new CamlQuery();
                q.ViewXml = @"<View><Query><Where><Eq><FieldRef
Name='FileLeafRef'/><Value Type='Text'>"
                    + lstItems.SelectedItem.ToString()

+ "</Value></Eq></Where><RowLimit>1</RowLimit></Query></View>";


ClientOM.ListItemCollection listitems = selectedList.GetItems(q);
                clientContext.Load(selectedList);
                clientContext.Load(listitems);
                clientContext.ExecuteQuery();

                if (listitems.Count == 1)
                {

                    ClientOM.ListItem item = listitems[0];

FileInformation fileInformation = ClientOM.File.
OpenBinaryDirect(clientContext, (string)item["FileRef"].ToString());
                    using (MemoryStream memoryStream = new Memo
ryStream())
                    {
                        CopyStream(fileInformation.Stream, memo
ryStream);

using (SpreadsheetDocument xlsx = SpreadsheetDocument.
Open(memoryStream, false))
                        {
```

CODEFRAGMENT 4: OPENEN VAN EEN EXCEL FILE.

```
        static private void CopyStream(Stream source, Stream
destination)
        {
            byte[] buffer = new byte[32768];
            int bytesRead;
            do
            {
                bytesRead = source.Read(buffer, 0, buffer.Length);
                destination.Write(buffer, 0, bytesRead);
            } while (bytesRead != 0);
        }
```

CODEFRAGMENT 5: COPYSTREAM ROUTINE.

FIGUUR 5: WPF SCHERM.



FIGUUR 6: TOOLING OPEN XML.

Samenstellen van de aanwezige Defined Names is niet meer dan een routine die de Defined Names ophaalt uit een collectie binnen een workbook. Zie codefragment 6 ter illustratie.

```
                        foreach (DefinedName name in workbookPart.
                        Workbook.
GetFirstChild<DefinedNames>())
                            {
                                lstDefinedNames.Items.Add(name.
Name);
                            }
```

CODEFRAGMENT 6: DEFINED NAMES.

Voor de grafieken komt de tool opnieuw van pas. De 'namen' van de grafieken bevinden zich in de 'drawing2.xml' en zijn onderdeel van een 'GraphicFrame'; vervolgens moet de code op zoek gaan naar een 'ChartReference' en de 'NonVisualGraphicFrameProperties' uitvragen, zoals codefragment 7 laat zien.

```
            using (ClientOM.ClientContext clientContext = new
            ClientOM.
ClientContext(txtURL.Text))
            {


ClientOM.List selectedList = clientContext.Web.Lists.
GetByTitle(lstLists.SelectedItem.ToString());
                ClientOM.CamlQuery q = new CamlQuery();
```

```
                q.ViewXml = @"<View><Query><Where><Eq><FieldRef
Name='FileLeafRef'/><Value Type='Text'>"
                    + lstItems.SelectedItem.ToString()

+ "</Value></Eq></Where><RowLimit>1</RowLimit></Query></View>";


ClientOM.ListItemCollection listitems = selectedList.GetItems(q);
                clientContext.Load(selectedList);
                clientContext.Load(listitems);
                clientContext.ExecuteQuery();

                if (listitems.Count == 1)
                {

                    ClientOM.ListItem item = listitems[0];

FileInformation fileInformation = ClientOM.File.
OpenBinaryDirect(clientContext, (string)item["FileRef"].ToString());
                    using (MemoryStream memoryStream = new
MemoryStream())
                    {
                        CopyStream(fileInformation.Stream, memor
yStream);

using (SpreadsheetDocument xlsx = SpreadsheetDocument.
Open(memoryStream, false))
                        {
                            WorkbookPart workbookPart = xlsx.
WorkbookPart;
                            lstDefinedNames.Items.Clear();
                            lstCharts.Items.Clear();

// collectie van defined names samenstellen en weergeven.

foreach (DefinedName name in workbookPart.Workbook.
GetFirstChild<DefinedNames>())
                            {
                                lstDefinedNames.Items.Add(name.
Name);
                            }

//ophalen named charts (niet hetzelfde als defined names)

foreach (WorksheetPart sheetPart in xlsx.WorkbookPart.Worksheet
Parts)
                            {
                                if (sheetPart.DrawingsPart ==
null) continue;

DocumentFormat.OpenXml.Drawing.Spreadsheet.WorksheetDrawing wks =
sheetPart.DrawingsPart.WorksheetDrawing;

IEnumerable<TwoCellAnchor> twocellAnchors = wks.
Descendants<TwoCellAnchor>();

 foreach (TwoCellAnchor twocellAnchor in twocellAnchors)
                                {

IEnumerable<GraphicFrame> graphicFrames = twocellAnchor.
Descendants<GraphicFrame>();

foreach (GraphicFrame graphicFrame in graphicFrames)
                                    {

A.Graphic graphic = graphicFrame.GetFirstChild<A.Graphic>();

A.GraphicData graphicData = graphic.GraphicData;

A.Charts.ChartReference chartReference = graphicData.
GetFirstChild<A.Charts.ChartReference>();
                                        if (chartReference !=
null)
                                        {

NonVisualGraphicFrameProperties nvdp = graphicFrame.GetFirstChild<
NonVisualGraphicFrameProperties>();

lstCharts.Items.Add(nvdp.NonVisualDrawingProperties.Name.ToSt
ring());
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
```

CODEFRAGMENT 7: NONVISUALGRAPHICFRAMEPROPERTIES.

Chart Images tonen met Excel-REST

Met SharePoint 2010 is het mogelijk om Excel bestanden te benaderen met Excel-REST: een aspx pagina in de _vti_bin directory, waar, met behulp van een URL als 'http://Localhost/_vti_bin/ExcelRest.aspx/Financieel/Model' de Excel informatie is uit te lezen. Door de URL verder op te bouwen met '/Charts('GrafiekRente')?$format=image' wordt de grafiek 'GrafiekRente' van het voorbeeld getoond als PNG bestand.

In codefragment 8 wordt aan de hand van de gemaakte selecties (Excel File, Lijst en grafieknaam) een Uri gemaakt die als Source gebruikt wordt om een 'BitMapImage' te genereren. Het control 'Image1' krijgt dan als source de BitMapImage waarmee dan de grafiek op het scherm wordt getoond. Ook wordt de BitmapImage als een globale variabele opgeslagen om later te kunnen gebruiken voor het samenstellen van een Word-document.

```
private void lstCharts _ SelectionChanged(object sender, Selection
ChangedEventArgs e)
        {

System.Windows.Media.Imaging.BitmapImage bitmapImage = new System.
Windows.Media.Imaging.BitmapImage();
            bitmapImage.BeginInit();

// samenstellen van een Uri die gebruikt wordt om een image
ophalen van de Excel-REST Service

bitmapImage.UriSource = new Uri(txtURL.Text.ToString() + @"/ _ vti _
bin/ExcelRest.aspx/"

+ lstLists.SelectedItem.ToString() + @"/" + lstItems.SelectedItem.
ToString() + @"/model/Charts('" + lstCharts.SelectedItem.ToString()
+ @"')?$format=image");
            bitmapImage.EndInit();
            image1.Source = bitmapImage;
            //referentie zetten naar global variabele
            _ myPicture = bitmapImage;        }
```

CODEFRAGMENT 8: SAMENSTELLING VAN DE EXCEL-REST URI.

Word-document uit een Excel-grafiek

Alle informatie is nu aanwezig om het Word-document te kunnen genereren. Er zijn twee manieren om een Word Open XML document te genereren: Je kunt gebruikmaken van een bestaande template en deze aanpassen of je kunt een nieuw document maken. In beide gevallen wordt het document bewaard in een 'MemoryStream' zoals getoond in code fragment 9.

```
// Aanmaken van een stream tbv in memory maken van een Word
document.
            using (MemoryStream myStream = new MemoryStream())
            {
                using (WordprocessingDocument package =
WordprocessingDocument.Create(myStream, WordprocessingDocument
Type.Document))
                {
```

CODEFRAGMENT 9: MEMORYSTREAM OM EEN WORD DOCUMENT TE GENEREREN.

Daarna wordt het document verder in memory verrijkt met data. Een body wordt gemaakt met daarin een tabel en twee cellen. In

de eerste rij 'colRowTxt' samen met een kolom 'colCellTxt' wordt de tekst uit de textbox in de User Interface geplaatst in een zogenaamde 'Paragraph' en de kolom als element toegevoegd aan de rij en vervolgens de rij aan de tabel.

```
TextRange textRange = new TextRange(richSummary.Document.
ContentStart, richSummary.Document.ContentEnd);
                    WP.Body body = new WP.Body();
                    WP.Table table = new WP.Table();

                    //Maak de Main Document Part
                    package.AddMainDocumentPart();
                    package.MainDocumentPart.Document = new WP.
Document(body);

                    //Voeg een regel en cel met wat tekst toe.
                    WP.TableRow colRowTxt = new WP.TableRow();
                    WP.TableCell colCellTxt = new WP.TableCell();


colCellTxt.Append(new WP.Paragraph(new WP.Run(new WP.Text(text
Range.Text))));
                    colRowTxt.Append(colCellTxt);
                    table.Append(colRowTxt);
```

CODEFRAGMENT 10: TEKST PLAATSEN MET HULP VAN OPEN XML.

Dan is het de bedoeling dat de Excel-afbeelding in het document komt: eerst wordt met een 'ImagePart' een 'RelationShip' naar de afbeelding gemaakt. Aangezien we weten dat de afbeelding van het type PNG is, zal deze ook zo worden gedefinieerd in de 'ImagePartType.Png'.
We gebruiken een MemoryStream om de afbeelding naar bitmap te encoden en in de Imagepart te krijgen.
Om de Image de juiste breedte en hoogte te geven, worden twee variabelen gevuld met de data uit de encoder. De encoder geeft pixels terug, terwijl de drawing English Metric Units verwacht. Om de EMU waarde te krijgen in Pixels moeten zij vermenigvuldigd worden met 9525.

```
MemoryStream picMemStream;
                    System.Windows.Media.Imaging.PngBitmapEncoder
 myEncoder;

                    picMemStream = new MemoryStream();
                    myEncoder = new System.Windows.Media.Imaging.
PngBitmapEncoder();

myEncoder.Frames.Add(System.Windows.Media.Imaging.BitmapFrame.
Create(_myPicture.UriSource));
                    myEncoder.Save(picMemStream);
                    // Ga naar het begin van de stream.
                    picMemStream.Seek(0, SeekOrigin.Begin);
                    imagePart.FeedData(picMemStream);

Int64Value imgPixelHeight =                (Int64Value)myEncoder.
Frames[0].PixelHeight * 9525;

Int64Value imgPixelWidth = (Int64Value)myEncoder.Frames[0].
PixelWidth * 9525;
```

CODEFRAGMENT 11: CONVERSIE VAN DE AFBEELDING.

De volgende stap is de afbeelding als een embedded image toe te voegen in een nieuw aan te maken Paragraph. Belangrijk om hier te noemen is dat de relatie naar de afbeelding goed gezet wordt door middel van het embed attribute van het Blip object. Daarin staat een aanroep naar de methode 'MainDocumentPart.GetIdOfPart' met het 'imagePart' als object. De breedte en hoogte variabelen worden gebruikt voor zowel de drawing als de transformatie van de afbeelding.

```
//Voeg de afbeelding toe aan een rij en cel
WP.TableRow colRowIma = new WP.TableRow();
WP.TableCell colCellIma = new WP.TableCell();
colCellIma.Append(new WP.Paragraph(new WP.Run(
   new WP.Drawing(
   new WD.Inline(new WD.Extent(){ Cx = imgPixelWidth, Cy =
imgPixelWidth },

new WD.DocProperties() { Id = (UInt32Value)1U, Name = "Picture 0",
                    Description = "OnePicture.png" },
                  new A.Graphic(new A.GraphicData (new Pic.Picture
                          (new Pic.NonVisualPictureProperties (
                            new Pic.NonVisualDrawingProperties()

{ Id = (UInt32Value)0U, Name = "OnePicture.png" },
                                    new Pic.NonVisualPictureDrawing
Properties()),
                                        new Pic.BlipFill(
                                            new A.Blip()
                                  { Embed =Package.MainDocument
Part.GetIdOfPart(imagePart),

CompressionState = A.BlipCompressionValues.Print },
                                          new A.Stretch(new A.
FillRectangle())),
                                                                  new Pic.
ShapeProperties

(new A.Transform2D

  (new A.Extents()

{ Cx = imgPixelWidth, Cy = imgPixelHeight },

   new A.Offset()

{ X = 0L, Y = 0L }),

 new A.PresetGeometry

(new A.AdjustValueList())

{ Preset = A.ShapeTypeValues.Rectangle}))
                                                      ) { Uri =

"http://schemas.openxmlformats.org/drawingml/2006/picture" }))))));
            colRowIma.Append(colCellIma);
           table.Append(colRowIma);
            body.Append(table);
```

CODEFRAGMENT 12: IMAGE TOEVOEGEN.

Nu gaan we alleen nog het Word-document uploaden naar SharePoint. Met de volgende drie regels gebeurt dat: eerst bepaal je waar en met welke naam, dan zet je de stream aan het begin en vervolgens sla je het bestand op met 'SaveBinaryDirect'.

```
// Opslaan van het word document in SharePoint.
 string fileUrl = "/Shared Documents/" + txtFileName.Text;
 myStream.Seek(0, SeekOrigin.Begin);
 ClientOM.File.SaveBinaryDirect(clientContext, fileUrl, myStream,
true); }}
```

CODEFRAGMENT 13. UPLOADEN NAAR SHAREPOINT.

Code fragment 14 laat de gehele routine zien om het Word document te genereren met Open XML en op te slaan in een Library van SharePoint 2010.

```
private void btnSaveWord_Click(object sender, RoutedEventArgs e)
        {
            ClientOM.ClientContext clientContext = new ClientOM.
ClientContext(txtURL.Text);


// Aanmaken van een stream t.b.v. in memory maken van een
Word document.
            using (MemoryStream myStream = new MemoryStream())
            {
                using (WordprocessingDocument package =
WordprocessingDocument.Create(myStream, WordprocessingDocument
```

```
Type.Document))
                {

TextRange textRange = new TextRange(richSummary.Document.
ContentStart, richSummary.Document.ContentEnd);
                    WP.Body body = new WP.Body();
                    WP.Table table = new WP.Table();

                    //Maak de Main Document Part
                    package.AddMainDocumentPart();
                    package.MainDocumentPart.Document = new WP.
Document(body);

                    //Voeg een regel en cel met wat tekst toe.
                    WP.TableRow colRowTxt = new WP.TableRow();
                    WP.TableCell colCellTxt = new WP.TableCell();


colCellTxt.Append(new WP.Paragraph(new WP.Run(new WP.Text
(textRange.Text))));
                    colRowTxt.Append(colCellTxt);
                    table.Append(colRowTxt);

                    //Maak een relatie met de afbeelding

ImagePart imagePart = package.MainDocumentPart.
AddImagePart(ImagePartType.Png);


//Converteer de afbeelding naar een MemoryStream en upload naar
 media dir in het docx bestand
                    MemoryStream picMemStream;
                    System.Windows.Media.Imaging.PngBitmapEncoder
myEncoder;

                    picMemStream = new MemoryStream();
                    myEncoder = new System.Windows.Media.Imaging.
PngBitmapEncoder();

myEncoder.Frames.Add(System.Windows.Media.Imaging.BitmapFrame.
Create( _myPicture.UriSource));
                    myEncoder.Save(picMemStream);
                    // Ga naar het begin van de stream.
                    picMemStream.Seek(0, SeekOrigin.Begin);
                    imagePart.FeedData(picMemStream);


Int64Value imgPixelHeight = (Int64Value)myEncoder.Frames[0].
PixelHeight * 9525;

Int64Value imgPixelWidth = (Int64Value)myEncoder.Frames[0].
PixelWidth * 9525;

                    //Voeg de afbeelding toe aan een rij en cel
                    WP.TableRow colRowIma = new WP.TableRow();
                    WP.TableCell colCellIma = new WP.TableCell();
                    colCellIma.Append(
                                    new WP.Paragraph(new WP.Run(
                                        new WP.Drawing(
                                            new WD.Inline(

new WD.Extent() { Cx = imgPixelWidth, Cy = imgPixelWidth },

new WD.DocProperties() { Id = (UInt32Value)1U, Name = "Picture 0",
Description = "OnePicture.png" },
                                                    new A.Graphic(
                                                        new A.Graphic
Data
                                                            (
                                                            new Pic.
Picture
                                                                (

new Pic.NonVisualPictureProperties
                                                                (

new Pic.NonVisualDrawingProperties() { Id = (UInt32Value)0U,
Name = "OnePicture.png" },

new Pic.NonVisualPictureDrawingProperties()
                                                                ),
```

```
new Pic.BlipFill
                                                                (

new A.Blip() { Embed = package.MainDocumentPart.
GetIdOfPart(imagePart), CompressionState = A.BlipCompressionValues.
Print },

new A.Stretch(new A.FillRectangle())
                                                                ),

new Pic.ShapeProperties
                                                                (

new A.Transform2D

    (

  new A.Extents() { Cx = imgPixelWidth, Cy = imgPixelHeight },

  new A.Offset() { X = 0L, Y = 0L }),

 new A.PresetGeometry

(

new A.AdjustValueList()

) { Preset = A.ShapeTypeValues.Rectangle }))

) { Uri = "http://schemas.openxmlformats.org/drawingml/2006/pic
ture" }))))));
                    colRowIma.Append(colCellIma);
                    table.Append(colRowIma);
                    body.Append(table);
                }
                // Opslaan van het word document in SharePoint.

string fileUrl = "/Shared Documents/" + txtFileName.Text;
                myStream.Seek(0, SeekOrigin.Begin);

ClientOM.File.SaveBinaryDirect(clientContext, fileUrl, myStream,
true);
            }
        }
```

CODEFRAGMENT 14: COMPLETE CODE MAKEN WORD DOCUMENT.

Conclusies

De verschillende manieren om met client software direct met het SharePoint object model te kunnen communiceren bieden nieuwe interessante mogelijkheden. Open XML is een krachtige en eenvoudige te gebruiken standaard en met de SDK begin je er eenvoudig mee. SharePoint is de documentrepository bij uitstek, de Office programma's zijn de dagelijkse manier om de data te ontsluiten en Open XML en het Client Object Model van SharePoint 2010 zijn de bindende elementen om alles bij elkaar te houden en te laten samenwerken.

Links

Open XML SDK 2.0 for Microsoft Office - http://www.microsoft.com/downloads/details.aspx?FamilyID=c6e744e5-36e9-45f5-8d8c-331df206e0d0&DisplayLang=en

Remco van Beek en Gustavo Velez, zijn Manager bij Avanade Nederland. Met gedegen SharePoint-ervaring worden zij dagelijks geconfronteerd met en architectuur- en ontwikkelvraagstukken rondom SharePoint 2007 en 2010. Gustavo Velez is Microsoft Most Valuable Professional in SharePoint.