Teun Duynstee is werkzaam als lead developer bij Macaw

Rapporteren over SharePoint-lijsten met een Reporting Services Extension

DE KOPPELING VAN WINDOWS SHAREPOINT SERVICES EN MICROSOFT REPORTING SERVICES

In het kader van Microsoft's inspanningen om de Business Intelligence breed beschikbaar te maken werd in 2004 Microsoft Reporting Services geïntroduceerd. Deze reporting engine is gratis voor klanten die al een SQL Server 2000-licentie hebben. Het stelt organisaties in staat WYSIWYG-rapporten op te stellen over verschillende bronnen zoals OLEDB-databases en OLAP-cubes. Reporting Services biedt een scala aan doelformaten (zoals HTML, PDF en Excel) en verschillende aflevermethoden, zoals een webinterface of geschedulede e-mails. Voor het ontwerpen en testen van nieuwe rapporten wordt gebruik gemaakt van Visual Studio. Dit maakt Reporting Services voor de typische Microsoft-ontwikkelaar een herkenbare omgeving waarin in weinig tijd indrukwekkende rapportages kunnen worden ontwikkeld.

Nu zijn er natuurlijk meer data in organisaties dan de data in relationele databases. In veel organisaties worden bijvoorbeeld SharePoint-lijsten gebruikt om bedrijfskritische processen in bij te houden. Bijvoorbeeld door projectteams die de status en prioriteit van alle deeltaken van het project in een Tasks-lijst bijhouden. Ook daarover zou je graag willen kunnen rapporteren vanuit Reporting Services, zodat het management in een oogopslag de voortgang van de taken kan zien in bijvoorbeeld een grafiek. Gelukkig is Reporting Services voorzien van een uitbreidbare architectuur. We kunnen dus in deze behoefte voorzien door een zogenaamde Reporting Services Extension te schrijven. Hoe je dat aanpakt is het onderwerp van dit artikel.

Uitbreidbaarheid SharePoint

Een extension op Reporting Services, die kan rapporteren over data uit SharePoint-lijsten, moet in ieder geval zelf in staat zijn de benodigde data uit SharePoint op te halen. Dat is niet zo moeilijk, omdat Windows SharePoint Services (WSS) voorzien is van een heel scala aan SOAP-webservices. In ons geval zijn we geïnteresseerd in de SOAP service Lists.asmx. Deze service bestaat voor iedere WSS-site onder de url http://servername/sites/sitename/_vti_bin/Lists.asmx. In deze service zijn methodes beschikbaar zoals GetListCollection (geeft alle beschikbare lijsten in de site terug), GetList (geeft informatie over de lijst en al zijn kolommen) en GetListItems (geeft de content terug). Al deze methodes hebben als resultaat een XmlNode-object (in de WSDL is dit een <xsd:any /> type). Het is in de praktijk vaak nog een hele klus om uit deze XML precies de gewenste informatie terug te vinden, maar op zich is alle lijstinformatie via SOAP-webservices op te vragen.

Uitbreidbaarheid Reporting Services

In de architectuur van Reporting Services is op een aantal plaatsen rekening gehouden met de behoefte om de functionaliteit uit te breiden. Dit gebeurt volgens een plug-in model. De uitbreiding daarbij is een assembly met daarin types die bepaalde interfaces implementeren. Reporting Services gebruikt van binnen alleen deze interfaces. Welk type wordt geïnstantieerd hangt af van de configuratie en de keuze van de gebruiker.

Reporting Services kent voor vier verschillende onderdelen een plug-in:

- Data Processing Extension voor het rapporteren over andere databronnen; deze plug-in gaan we in dit artikel gebruiken.
- Delivery Extension om een nieuwe manier van afleveren van geschedulede rapporten te creëren; de standaard mogelijkheden zijn e-mail en fileshare.
- Rendering Extension voor het genereren van rapporten in nieuwe documentformaten. Er zijn standaard renderers beschikbaar voor bijvoorbeeld HTML 3.2, HTML 4.0, TIFF-afbeeldingen, PDF-documenten, Excel en XML. Maar wellicht is in sommige organisaties behoefte aan afwijkende formaten. RTF en Microsoft Word zijn bijvoorbeeld opvallende afwezigen.
- Security Extension voor het koppelen van de Role Based Security van Reporting Services aan een eigen authenticatie/autorisatiesysteem, zoals een LDAP-service. Standaard valideert Reporting Services alle user credentials tegen een windows-domein. Het spreekt vanzelf dat het zelf bouwen van een security-extension veiligheidsrisico's met zich meebrengt. Pas hiermee op.

De werkwijze

Voor het rapporteren over SharePoint-lijsten zullen we een Data Processing Extension schrijven. Voordat we in de code duiken, kijken we eerst naar de manier waarop de Reporting Services-engine onze extension gebruikt. Het gebruikte patroon lijkt sprekend op dat van DataReaders in ADO.NET. Een Data Procesing Extension bestaat uit classes die de volgende interfaces implementeren (alle in de Microsoft.ReportingServices.DataProcessing): IDbConnection, IDbCommand en IDataReader. Daarnaast bestaat er nog een aantal extra interfaces dat geïmplementeerd kan worden wanneer nodig, maar deze drie interfaces vormen de kern.

Zoals te zien is in afbeelding 1, creëert de Reporting Services-engine een instantie van IDbConnection. Op deze class wordt een aantal properties gezet (bijvoorbeeld ConnectionString) waarna de methodes Open en CreateCommand worden aangeroepen. Het Connection-object geeft dan een instantie terug die IDbCommand implementeert. Dit Command-object beschikt over CommandType- en CommandText-properties en een ExecuteReader-methode. Deze methode geeft een object terug dat IDataReader implementeert. Dit object lijkt sterk op een ADO.NET DataReader. Deze forward-only reader gebruikt Reporting Services vervolgens om alle te rapporteren data uit de bron te trekken.

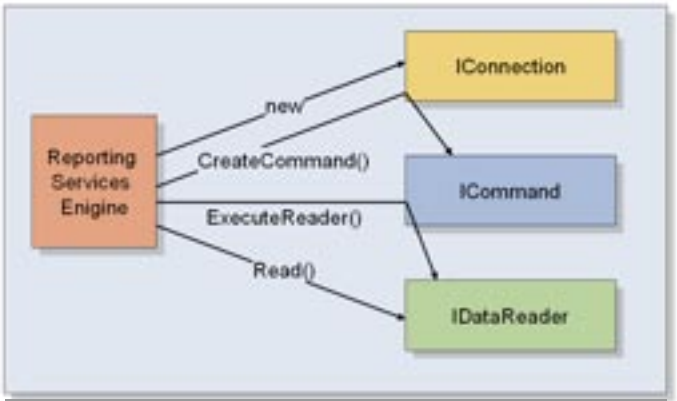
We zullen nu de relevante stukjes code bekijken uit elk van de centrale classes in onze extension. De volledige werkende broncode van deze extension kan worden gedownload vanaf www.microsoft.nl/netmagazine9

Connection

De Connection-class implementeert IDbConnection en is voor Reporting Services het startpunt bij het gebruik van onze extensie. Van de properties en methodes uit IDbConnection wordt een aantal niet ondersteund. Daarom zal BeginTransaction altijd een NotImplementedException gooien en geeft de ConnectionTimeout-property altijd 0 terug. Transacties en time-outs zijn voor onze connectie niet relevant, omdat we geen echte databaseconnectie hebben. Daarom gebeurt er ook zo weinig in de Open- en Close-methodes van Connection; zie codevoorbeeld 1.

Wel belangrijk is de property ConnectionString. De connection string kan voor deze extension eigenlijk maar één gegeven bevatten: de URL van de WSS-site waaruit de data moeten worden opgehaald. Om uitbreiding later mogelijk te maken hanteren we voor de connection string toch het ingewikkelde formaat van de OLEDB connection string: NAME1=VALUE1;NAME2=VALUE2. De WssSiteUrl-property wordt gebruikt om de juiste waarde uit de connection string te lezen, die opgeslagen wordt in een HashTable; codevoorbeeld 2.

Naast de IDbConnection-interface implementeert de Connection-class ook nog de IDbConnectionExtension interface. Deze interface bestaat uit vier properties die met security en login-gegevens te maken hebben: Impersonate, IntegratedSecurity, UserName en Password. Door deze interface te implementeren geven we aan dat we in staat zijn om op basis van de gegevens die in deze properties worden ingevuld een bijbehorende connectie op te bouwen. In ons geval roepen we de webservices aan en geven we de login-gegevens op dat moment door aan de WSS-webserver. De properties zelf doen niets met de informatie. De informatie wordt later gebruikt door het Command-object. De CreateCommand-methode doet niets anders dan een instantie van het juiste Command teruggeven; codevoorbeeld 3.



Afbeelding 1. Collaboration diag voor RS, idbconnection, idbcommand en idatareader.

De LocalizedName-property geeft ons de mogelijkheid te bepalen hoe onze extensie in het lijstje met mogelijke bronnen verschijnt. In ons geval geven we altijd de tekst “SharePoint Lists” terug.

```
private System.Data.ConnectionState _state =
        System.Data.ConnectionState.Closed;

public System.Data.ConnectionState State
{
   get {return _state;}
}

public void Open()
{
   _state = System.Data.ConnectionState.Open;
   return;
}

public void Close()
{
   _state = System.Data.ConnectionState.Closed;
   return;
}
```

Codevoorbeeld 1.

```
private string _connString;
private Hashtable _connStringParts = new Hashtable();

public string ConnectionString
{
   get
   {return _connString;}
   set
   {
      _connString = value;
      string[] parts = _connString.Split(';');
      _connStringParts = new Hashtable();
      foreach(string part in parts)
      {
         if(part.IndexOf("=") < 1)
         {
            throw new ArgumentException(String.Format(
               "The part {0} is not a valid part for a
               connection string", part));
         }
         _connStringParts.Add(
            part.Substring(0, part.IndexOf("=")).ToLower(),
            part.Substring(part.IndexOf("=") + 1)
            );
      }
   }
}

public string WssSiteUrl
{
   get
   {
      if(_connStringParts.ContainsKey("site"))
      {
         string site = (string)_connStringParts["site"];
         if(!site.EndsWith("/")) site = site + "/";
         return site;
      }
      return "";
   }
}
```

Codevoorbeeld 2.

```
public IDbCommand CreateCommand()
{
   Command result = new Command(this);
   return result;
}
```
Codevoorbeeld 3.

```
private CommandType _type = CommandType.Text;

public CommandType CommandType
{
   get { return _type; }
   set
   {
      if (value != CommandType.Text) throw new NotSupportedException();
      _type = value;
   }
}
```
Codevoorbeeld 4.

```
public IDataReader ExecuteReader(CommandBehavior behavior)
{
   data.DataTable table = GetDataTableFromWSS((behavior ==
      CommandBehavior.SchemaOnly));

   return new DataReader(this._connection, table, _parameters);
}
```
Codevoorbeeld 5.

Command

Het Command-object, dat wordt teruggegeven door de CreateCommand–methode, is feitelijk de kern van onze oplossing. Het object is in staat om op basis van het Connection-object en de naam van een SharePoint-lijst een ADO.NET DataTable te vullen met de benodigde data. Vervolgens kan een DataReader-object worden geïnstantieerd dat de data uit deze DataTable aanbiedt aan de Reporting Services-engine.

De belangrijkste properties van de IDbCommand-interface zijn CommandText en CommandType. CommandText bevat bij het rapporteren over een relationele database een SQL-query. In onze extensie gebruiken we de CommandText-property om de naam van de lijst door te geven. We ondersteunen van de drie CommandType-waardes alleen CommandType.Text; codevoorbeeld 4. Onze oplossing ondersteunt geen parameters en transacties. De property Parameters geeft dus een lege collectie terug en de Transaction-property is altijd null, terwijl de CreateParameter-methode een NotImplementedException gooit. De CommandTimeout-property is in deze oplossing hardcoded op 30 seconden.

Het echte werk van Command gebeurt wanneer de methode ExecuteReader wordt aangeroepen. Deze roept GetDataTableFromWSS aan en geeft de resulterende DataTable mee aan een nieuwe DataReader; codevoorbeeld 5. De GetDataTableFromWSS-methode is dus het echte werkpaard van de oplossing. Codevoorbeeld 6 toont de volledige code van deze methode. Er wordt eerst een instance van de proxy class voor de webservice geïnstantieerd. Hierop wordt een aantal gegevens ingevuld op basis van de Connection. Daarna wordt de GetListCollection-methode aangeroepen. In codevoorbeeld 7 staat een fragment van de XML die deze service teruggeeft. Het is ons er vooral om te doen om de Name van de lijst te achterhalen. Dat is een Guid die we bij alle volgende webserviceaanroepen nodig hebben.

Met deze unieke naam wordt vervolgens de webservice GetList aangeroepen. De resulterende XML bevat informatie over naam en type van alle kolommen in de lijst; codevoorbeeld 8. Voor elke WSS-kolom voegen we een nieuwe kolom toe aan onze DataTable

```
private data.DataTable GetDataTableFromWSS(bool onlySchema)
{
   WssSite.Lists.Lists lists = new WssSite.Lists.Lists();
   lists.Url = _connection.WssSiteUrl + "_vti_bin/Lists.asmx";
   lists.Credentials = GetCredential(_connection);
   lists.PreAuthenticate = true;
   lists.Timeout = this.CommandTimeout;

   XmlNode ListCollectionNode = lists.GetListCollection();
   XmlElement List = (XmlElement)ListCollectionNode.SelectSingleNode(
      String.Format("wss:List[@Title='{0}']", this.CommandText),
         NameSpaceMgr);
   if(List == null)
   {
      throw new ArgumentException(
         String.Format("The list {0} could not be found in the site {1}",
         this.CommandText, _connection.WssSiteUrl));
   }
   string TechListName = List.GetAttribute("Name");
   data.DataTable result = new data.DataTable("list");

   XmlNode ListInfoNode = lists.GetList(TechListName);
   StringBuilder fieldRefs = new StringBuilder();

   Hashtable DisplayNames = new Hashtable();
   foreach(XmlElement Field in
      ListInfoNode.SelectNodes("wss:Fields/wss:Field", NameSpaceMgr))
   {
      string FieldName = Field.GetAttribute("Name");
      string FieldDisplayName = Field.GetAttribute("DisplayName");

      if(result.Columns.Contains(FieldDisplayName))
      {
         FieldDisplayName = FieldDisplayName + " (" + FieldName + ")";
   }
      result.Columns.Add(FieldDisplayName, TypeFromField(Field));
      fieldRefs.AppendFormat("<FieldRef Name=\"{0}\"/>", FieldName);
      DisplayNames.Add(FieldDisplayName, FieldName);
   }

   if(onlySchema)return result;

   XmlElement fields = ListInfoNode.OwnerDocument.CreateElement("ViewFields");
   fields.InnerXml = fieldRefs.ToString();

   XmlNode ItemsNode = lists.GetListItems(TechListName, "", null,
      fields, "1000000", null);

   // Lookup fields always start with the numeric ID, then ;# and then
   // the string representation.
   // We are normally only interested in the name, so we strip the ID.
   System.Text.RegularExpressions.Regex CheckLookup =
      new System.Text.RegularExpressions.Regex("^\\d+;#");

   foreach(XmlElement Item in ItemsNode.SelectNodes("rs:data/z:row",
      NameSpaceMgr))
   {
      data.DataRow newRow = result.NewRow();
      foreach(data.DataColumn col in result.Columns)
      {
         if(Item.HasAttribute("ows_" + (string)DisplayNames[col.ColumnName]))
         {
            string val = Item.GetAttribute("ows_" +
                  (string)DisplayNames[col.ColumnName]);
            if(CheckLookup.IsMatch((string)val))
            {
               string valString = val as String;
               val = valString.Substring(valString.IndexOf("#") + 1);
```

```
            }
            // Assigning a string to a field that expects numbers or
            // datetime values will implicitly convert them
            newRow[col] = val;
          }
        }
        result.Rows.Add(newRow);
    }
    return result;
}
```

Codevoorbeeld 6.

```
<Lists xmlns="http://schemas.microsoft.com/sharepoint/soap/">
  <List
    DefaultViewUrl="/sites/kd/work/Lists/Announcements/AllItems.aspx"
    ID="{FB328912-52A1-4E00-88C8-5388559437E8}"
    Title="Announcements"
    Description="Use the Announcements list to ..."
    ImageUrl="/_layouts/images/itann.gif"
    Name="{FB328912-52A1-4E00-88C8-5388559437E8}"
    ...many="" more="" attributes...=""
  />
  <List ...
</Lists>
```

Codevoorbeeld 7.

met dezelfde naam. Hierbij speelt één lastig detail: WSS kent voor iedere kolom een Name en een DisplayName. Als een gebruiker in SharePoint een nieuwe kolom aanmaakt zijn deze twee gelijk, maar de DisplayName mag nog gewijzigd worden terwijl de Name vaststaat. In de UI van SharePoint zie je altijd alleen de DisplayName. Het is dus van belang dat wij ook in de designer deze naamgeving volgen. We gebruiken dus de DisplayName als ColumnName in onze DataTable en onthouden welke WSS Name er hoort bij welke ColomnName (in een HashTable). DisplayNames die meermalen voorkomen maken we uniek door de Name er achter te zetten. We bouwen bovendien in een StringBuilder een stukje XML op dat we later nodig hebben om aan te geven dat we **alle velden** willen terugkrijgen.

Als we alleen het schema hoeven terug te geven - dit gebeurt in de designer - zijn we nu klaar, maar meestal is het juist om de data te doen. Die gaan we nu in de DataTable laden met behulp van de webservice-methode GetListItems. Als je deze methode aanroept krijg je een stuk XML terug zoals in codevoorbeeld 9. Merk op dat alle velden van de lijst zijn verwerkt tot attributen met de Name (niet DisplayName) van het veld, geprefixed met "ows_". We lopen door de z:row-elementen heen en kijken voor elk van de kolommen in onze DataTable of er een attribuut voor op de row aanwezig is. De naam van het attribuut bepalen we met de vaste prefix en de Name, die we in een Hashtable hadden bewaard voor elke kolomnaam. De string die wordt teruggegeven door de GetAttribute-methode kan zonder casten en converteren in het veld van de DataTable gestopt worden, want de DataTable voert intern altijd nog een Convert uit.

Wel veranderen we voor de leesbaarheid van de rapporten de waarde van de velden die als type een Choice-lijstje of een verwijzing naar een andere SharePoint-lijst hebben. Voor deze velden krijgen we een waarde terug van de vorm "36;#Teun Duynstee". Het numerieke ID komt eerst, dan een ; en een # waarna een leesbare weergave volgt. Voor rapportages zijn we eigenlijk zelden geïnteresseerd in ID's; we willen namen afbeelden. Daarom strippen we met een regular expression de ID en separator van de waarde voordat we hem in de DataTable stoppen. De gevulde DataTable wordt vervolgens meegegeven aan een DataReader, die wordt teruggegeven als resultaat van ExecuteReader.

DataReader

De DataReader is een vrij eenvoudige class. Het is niet veel meer dan een wrapper om de DataTable die het Command-object heeft gevuld. Intern wordt een Enumerator geïnstantieerd, die wordt gebruikt in de CurrentRow-property om de huidige rij terug te geven; zie codevoorbeeld 10. Deze CurrentRow-property wordt vervolgens gebruikt door alle methodes die in IDataReader gedefinieerd worden zoals GetFieldType, GetValue, et cetera; zie codevoorbeeld 11.

Het uitrollen van de oplossing

Als plug-ins eenmaal gebouwd zijn, kunnen ze eenvoudig geregistreerd worden door ze toe te voegen in een config-bestand. Dit moet op twee plaatsen gebeuren: voor de designer op de machine waar rapporten worden gebouwd en voor de service op de machine waar de rapporten uiteindelijk geproduceerd zullen worden. De dll die gebuild is, moet ook naar twee plaatsen gekopieerd worden. De stappen voor deployment zijn (de directorypaden zijn volgens de default-installatie):

- Kopieer de dll naar de folders C:\Program Files\Microsoft SQL Server\80\Tools\Report Designer en C:\Program Files\Microsoft SQL Server\MSSQL\Reporting Services\ReportServer\bin.
- Open met notepad de configbestanden C:\Program Files\Microsoft SQL Server\MSSQL\Reporting Services\ReportServer\RSReportServer.config en C:\Program Files\Microsoft SQL Server\80\Tools\Report Designer\RSReportServer.config. In beide bestanden staat een blokje met data-extensions. Hierin staan al de extensies voor SQL, OLEDB, ORACLE en ODBC). Hieraan voegen we onze extensie toe. Het resultaat daarvan is te zien in codevoorbeeld 12.

Op een machine waarop geen Visual Studio geïnstalleerd staat, bestaat de Report Designer-directory uiteraard niet.

```
<List >
  <Fields>
    <Field Type="Text" Name="Title" DisplayName="Title"
    Required="TRUE" FromBaseType="TRUE" ColName="nvarchar3" />
    <Field Name="Case" FromBaseType="FALSE" Type="Lookup"
      List="{BE7960D6-D604-4799-B34F-8FAA67A8392C}"
      ShowField="Case_x0020_Number" DisplayName="Requirement"
      Description="Case this workitem belongs to" Required="TRUE"
      ColName="int4" />
    ...
  </Fields>
</List>
```

Codevoorbeeld 8.

```
<listitems xmlns:s="uuid:BDC6E3F0-6DA3-11d1-A2A3-00AA00C14882"
  xmlns:dt="uuid:C2F41010-65B3-11d1-A29F-00AA00C14882"
  xmlns:rs="urn:schemas-microsoft-com:rowset"
  xmlns:z="#RowsetSchema"
  xmlns="http://schemas.microsoft.com/sharepoint/soap/">

  <rs:data ItemCount="4">
    <z:row ows_ID="1" ows_Title="Interesting task"
    ows_Modified="2004-02-25 00:14:40"
    ows_Created="2004-02-25 00:14:40"
    ows_Author="36;#Teun Duynstee"
    ows_Editor="36;#Teun Duynstee"
    ows_owshiddenversion="1"
    ows_Attachments="0"
    ows__ModerationStatus="0"
    ows_Status="Not Started" ows_AssignedTo="36;#Teun Duynstee"
    ...many more attributes...  />
    <z:row .../>
  </rs:data>
</listitems>
```

Codevoorbeeld 9.

```
IEnumerator TableEnum = null;

public bool Read()
{
  if(TableEnum == null)
  {
    TableEnum = _table.Rows.GetEnumerator();
  }
  return TableEnum.MoveNext();
}

protected DataRow CurrentRow
{
  get
  {
    return (DataRow)TableEnum.Current;
  }
}
```

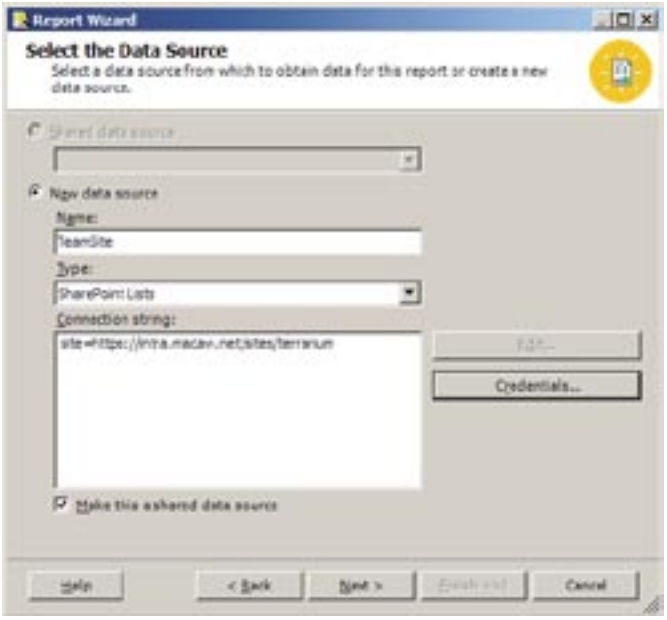
Codevoorbeeld 10.

```
public int FieldCount
{
  get { return _table.Columns.Count; }
}

public string GetName(int i)
{
  return _table.Columns[i].ColumnName;
}

public Type GetFieldType(int i)
{
  // Return the actual Type class for the data type.
  return _table.Columns[i].DataType;
}

public Object GetValue(int i)
{
  return CurrentRow[i];
}

public int GetOrdinal(string name)
{
  foreach(DataColumn col in _table.Columns)
  {
    if(col.ColumnName == name)
    {
      return col.Ordinal;
    }
  }
  throw new IndexOutOfRangeException("There is no field with
    the name " + name);
}
```

Codevoorbeeld 11.

```
<Extensions>
  <Data>
    <Extension Name="SQL" Type="Microsoft.ReportingServices… " />
    <Extension Name="OLEDB" Type="Microsoft.ReportingServices…"/>
    <Extension Name="ORACLE" Type="Microsoft.ReportingServices… "/>
    <Extension Name="ODBC" Type="Microsoft.ReportingServices…"/>
    <Extension Name="SPSLIST"
    Type="Macaw.ReportingServices.SharepointListExtension.Connection,
      Macaw.ReportingServices.SharepointListExtension"/>
  </Data>
</Extensions>
```

Codevoorbeeld 12.



Afbeelding 2. In de eerste wizard-stap wordt de verbinding geconfigureerd.

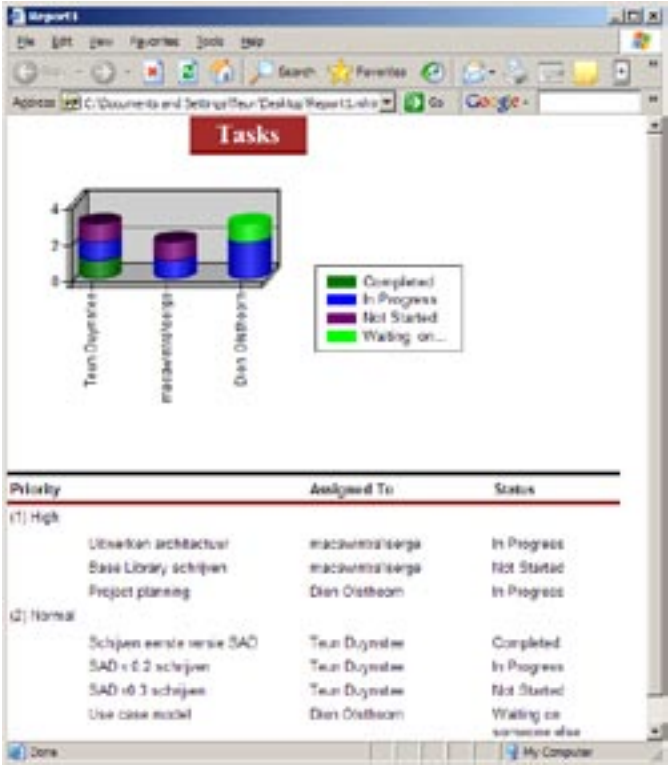
Resultaten

Nu we de extensie hebben geïnstalleerd, zullen we proberen een rapport te creëren op basis van de informatie in een teamsite. We nemen als bron in dit geval de lijst Tasks, die altijd standaard wordt aangemaakt in een project teamsite. In Visual Studio.NET 2003 maken we met de Report Project Wizard een nieuwe project aan.

In de wizard zien we de nieuwe functionaliteit reeds verschijnen. In afbeelding 2 is de wizard-stap te zien waarin de datasource wordt geconfigureerd. In het lijstje met datasource-types is nu ook de mogelijkheid 'SharePoint Lists' verschenen. De connection string bestaat uit slechts één element: de URL van de teamsite. Met de Credentials-knop kan het user-account worden ingevoerd waarmee de data uit WSS moet worden opgehaald. In het scherm erna kan de Command Text ingevoerd worden. We voeren daar 'Tasks' in, de naam van de lijst waarop we een query uitvoeren. Vervolgens komen we in het scherm van afbeelding 3.

In de beschikbare velden zijn alle velden opgenomen die de Tasks-lijst kent. Dit zijn er veel meer dan de velden die normaal gesproken getoond worden. Ook alle onzichtbare velden en calculated

Afbeelding 3. Automatisch worden alle velden (inclusief calculated fields) van de geselecteerde lijst aangeboden.



Afbeelding 4. Met klikken en slepen kunnen aantrekkelijke rapporten gemaakt worden op basis van de WSS Teamsite.

fields worden getoond. Deze zijn dus ook beschikbaar om over te rapporteren. Vervolgens wordt het rapport in elkaar geklikt zoals we dat altijd doen en kunnen we eenvoudig een resultaat krijgen zoals in afbeelding 4.

Conclusie

Doordat zowel Windows SharePoint Services en MS Reporting Services met de mogelijkheid van uitbreiding zijn ontworpen, is het niet zo moeilijk om de beide producten aan elkaar te koppelen. Er zijn al webparts beschikbaar om Reporting Services-rapporten in WSS-sites te tonen. Met de in dit artikel gepresenteerde techniek is het dus ook mogelijk om vanuit Reporting Services de data in de lijsten van WSS te benaderen. Hoe vaker organisaties hun primaire processen - geheel of gedeeltelijk - organiseren via teamsites, hoe belangrijker het zal worden om ook rapportages over de vorderingen te krijgen.

Teun Duynstee is als Lead Developer werkzaam bij Macaw. Hij is geïnteresseerd in het doorgronden van nieuwe technologie, vooral als er een concrete toepassing in het verschiet ligt. Nieuwe mogelijkheden met anderen bespreken en uitleggen (in schrift of bij het koffiezetapparaat) gaat nooit vervelen. Antwoorden op vragen naar aanleiding van dit artikel zal de auteur behandelen op zijn weblog: http://blog.duynstee.com

Nuttige internetadressen

In nummer 5 van .NET magazine stond een introductie in MS Reporting Services: http://www.microsoft.nl/netmagazine5

MSDN documentatie over het uitbreiden van Reporting Services: http://msdn.microsoft.com/library/en-us/RSPROG/htm/rsp_prog_extend_install_0ur7.asp

Om Reporting Services rapporten te tonen binnen een WSS site: http://www.gotdotnet.com/workspaces/workspace.aspx?id=176e78a1-4d90-4860-9bd8-da93dffa8fd1

Artikel over het benaderen van WSS over SOAP: http://www.developer.com/tech/article.php/3104621