Peter ter Braake en Peter Lemmens zijn respectievelijk productspecialist Microsoft SQL Server en Microsoft CRM bij Compu'Train.

Introductie in Reporting Services

WAAROM EEN RAPPORTAGE-'TOOL' INTERESSANT IS VOOR ONTWIKKELAARS

Al enkele jaren is Microsoft met SQL Server één van de grote spelers op de databasemarkt. Gedreven door de concurrenten en in grote mate door de wensen van de klanten is Microsoft SQL Server in relatief korte tijd uitgegroeid tot een volwaardige en zeer betrouwbare enterprise database. Tot nu toe ontbreekt het echter aan een rapportageomgeving die de verzamelde informatie op een veilige en efficiënte manier door een hele organisatie beschikbaar kan maken. SQL Server-gebruikers maken deze opmerking zo vaak in de SQL wishlist, dat Microsoft in 2003 besluit om niet langer te wachten totdat SQL Server 2005 (Yukon) uitkomt, maar reeds nu Reporting Services uit te brengen voor SQL Server 2000. Dit past volledig bij de aandacht die Microsoft de laatste tijd besteedt aan BI (Business Intelligence).

In het eerste deel van dit artikel zullen we een overzicht geven van Reporting Services. Waar krijgen we functioneel gezien mee te maken als we rapporten gaan inzetten binnen een organisatie en wat zijn de componenten waaruit het Reporting Services-platform is opgebouwd? In het tweede deel gaan we praktisch in op het benaderen van de rapporten vanuit code. Gezien het gemak waarmee we met wizards en bekende drag-and-drop-functionaliteit rapporten kunnen maken in Visual Studio, laten we het maken van de rapporten zelf buiten beschouwing.

Reporting lifecycle

Reporting Services is een serverplatform dat de volledige rapport-lifecycle ondersteunt. Tegenwoordig onderscheiden we drie fases (authoring, management en delivery) in deze lifecycle. We geven op alle drie de fases een korte toelichting.

De eerste fase is het ontwerpen en maken van de rapporten (report authoring). In deze fase bepalen we voor een rapport de te gebruiken gegevensbron, de query die we nodig hebben (welke velden uit de database gaan we tonen) en de lay-out van het rapport. Deze informatie wordt binnen Reporting Services opgeslagen als een RDL-bestand. RDL staat voor Report Definition Language. RDL is een op XML gebaseerde beschrijving van een rapportdefinitie en is een door Microsoft gedefinieerde open standaard voor het beschrijven van rapporten (zie www.microsoft.com/sql/reporting/techinfo/rdlspec.asp). Microsoft hoopt via deze open standaard de onderlinge uitwisseling van verschillende rapporten en ontwikkelomgevingen te stimuleren. In een RDL-bestand wordt de gegevensbron van het rapport beschreven (connection informatie en query informatie), de lay-out van het rapport (grafiek, draaitabel, kolommen enz.) en ten slotte overige eigenschappen zoals de voor het rapport gedefinieerde parameters. Bij Reporting Services levert Microsoft een add-in voor Visual Studio 2003 .NET waarmee report-projecten kunnen worden gemaakt en waarmee we dus RDL-bestanden maken die op de Report Server gepubliceerd kunnen worden. Gezien het open karakter is het uiteraard ook mogelijk zelf via een eigen tool of editor RDL-bestanden te genereren.

De tweede fase in de lifecycle is het rapportmanagement. Een belangrijk aspect van het managen van een rapport is wat we report execution noemen. Op welke wijze gaan we de benodigde gegevens uit de database ophalen? Rapporten die veel gegevens bevatten en rapporten die erg frequent worden opgehaald, kunnen een gevaar vormen voor de performance van de onderliggende gegevensbron. Om dit te omzeilen maken we gebruik van caching. Bij caching geven we aan dat de opgehaalde gegevens een bepaalde tijd in het geheugen van de Report Server moeten blijven. Als iemand dat rapport binnen die tijd opvraagt, hoeft de database niet opnieuw benaderd te worden maar komen de gegevens direct uit de cache. Een andere optie is periodiek een snapshot maken op basis waarvan het rapport gegenereerd wordt. Een tweede belangrijk aspect van rapportmanagement is beveiliging. Reporting Services gaat uit van role-based security, waarbij groepen en gebruikers aan rollen worden toegevoegd, die bepaalde taken kunnen/mogen uitvoeren. Report Manager is een in C# geschreven ASP.NET-applicatie die we gebruiken om alle management gerelateerde zaken te regelen.

De laatste fase van de rapport lifecycle is het bij de mensen krijgen van het rapport (report delivery). Een gebruiker kan zelf het initiatief nemen een rapport te genereren (on demand delivery of pull-mechanisme genoemd). Aangezien Reporting Services draait binnen Internet Information Server (IIS) is het mogelijk om via een browser een rapport te benaderen. Er zijn twee virtuele directories waar we naar toe kunnen browsen: de Reports virtual directory die gebruikt wordt door de Report Manager-applicatie en de ReportServer virtual directory van de Reporting Service zelf. Daarnaast kunnen we de Reporting Service als webservice benaderen. Later in dit artikel komen we op beide manieren gedetailleerder terug.

Naast het hierboven beschreven pull-mechanisme ondersteunt Reporting Services ook een push-mechanisme. Als een rapport de login-gegevens om een databron te benaderen zelf opslaat, kunnen we zogeheten 'subscriptions' maken. Daarna bestaat er de mogelijkheid om op basis van bijvoorbeeld veranderende data, of periodiek, een rapport te genereren. Dit rapport kan vervolgens als e-mail bijlage verstuurd worden of naar een file server gekopieerd worden. Reporting Services maakt gebruik van de SQL Agent om dit push-mechanisme te ondersteunen. Naast het afleveren van rapporten bij de gebruiker bepalen we in deze derde fase van de rapport-lifecycle ook het formaat waarin we een rapport willen ontvangen. Er wordt een aantal formaten ondersteund zoals HTML, XML, Excel, PDF.

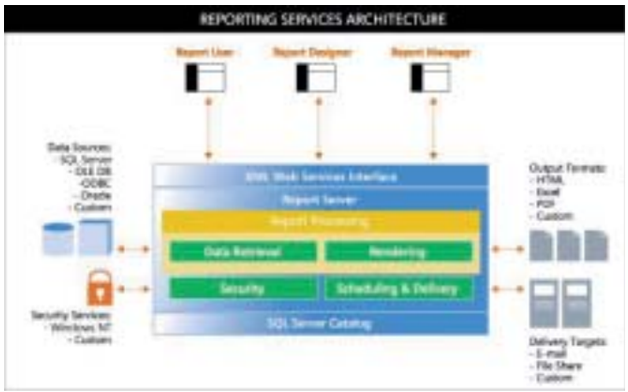
Architectuur

In afbeelding 1 staat een overzicht van de architectuur van Reporting Services. Aan de basis staat SQL Server waarin alle informatie zoals gebruikers, groepen, schedules en dergelijke wordt opgeslagen. Meerdere Reporting servers kunnen samen gebruikmaken van een enkele ReportServer database. Tijdens de installatie van Reporting Services worden op de aangegeven SQL Server twee nieuwe databases gemaakt; ReportServer en ReportServerTempDB. De ReportServerTempDB is, zoals de naam al aangeeft, een database met slechts tijdelijke informatie. De ReportServer database is bedoeld om metadata op te slaan, zoals gegevens over data sources, users, policies, subscriptions etc. Daarmee kunnen we de webapplicaties volledig stateless maken. Uiteraard is daarmee niet gezegd dat alleen SQL Server gebruikt kan worden om rapporten te genereren. De metadata van elk rapport wordt weliswaar in SQL Server opgeslagen, de echte data waaruit het rapport wordt opgesteld kan naast SQL Server uit een Oracle database komen of uit elke mogelijke ODBC- of OLE DB-gegevensbron.

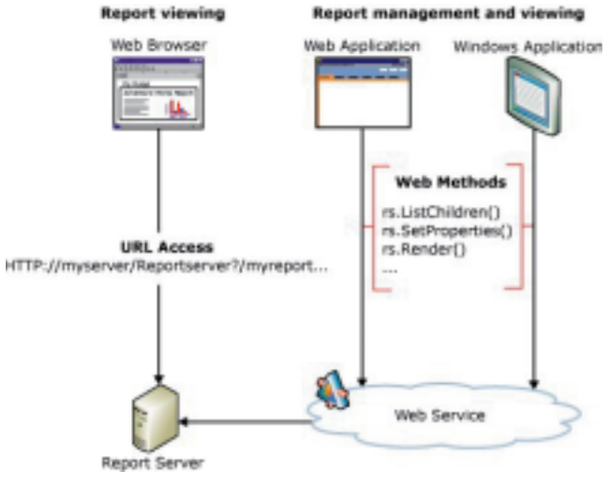
Het voornaamste onderdeel is uiteraard de Report Server zelf, een in C# geschreven webservice die uit een aantal componenten is opgebouwd. De Report Server verwerkt alle rapportaanvragen en gebruikt de vier in afbeelding 1 getoonde componenten (Scheduling and Delivery, Rendering, Data Retrieval, Security) om het rapport te genereren. De 'Data Retrieval'-module bevat zoals gezegd de code om ODBC- en OLE DB-gegevensbronnen te benaderen. Het spreekt bijna vanzelf dat er modules in zitten om rechtstreeks SQL Server- en Oracle-databases te benaderen. De overige componenten zorgen voor het gewenste outputformaat van het rapport getoond, de manier waarop het rapport bij de gebruiker gebracht moet worden en hoe de beveiliging geregeld is.

Bij alle vier de componenten staat, naast het rijtje met ingebouwde mogelijkheden, het woord 'custom'. Het mooie van de modulaire structuur van Reporting Services en de gepubliceerde managed code API's is, dat we bij alle modules onze eigen uitbreidingen kunnen schrijven. Is er sprake van een eigen databaseformaat of een bestandsformaat waarin het rapport afgeleverd moet worden (bijvoorbeeld als ZIP-bestand), dan kunnen we een .NET assembly schrijven die juist dat doet. Via het RSReportServer.config bestand maken we het bestaan van deze assembly bekend aan de Reporting Service die vervolgens in staat is de toegevoegde functionaliteit te gebruiken. Aan het einde van dit artikel komen we hier nog kort op terug.

Bij installatie van de Reporting Service worden twee belangrijke applicaties meegeïnstalleerd. De eerste is de Report Designer uit afbeelding 1. Dit is een add-in voor Visual Studio .NET 2003 waarmee we zoals gezegd RDL-bestanden genereren die vervolgens via de webservice interface gepubliceerd kunnen worden op de Report Server. Report Manager is de webbased beheertool die met name in fase 2 van de rapport lifecycle van belang is.



Afbeelding 1.



Afbeelding 2.

Programmeren

Afbeelding 2 laat twee manieren zien hoe we Reporting Services benaderen. De gemakkelijkste manier is via 'direct URL Access'. De tweede manier is door gebruik te maken van het feit dat Reporting Services is geschreven als webservice. Zoals bekend kunnen we via HTTP en SOAP (Simple Object Access Protocol) een webservice benaderen. Dat maakt het dus mogelijk om op vrij eenvoudige wijze de volledige functionaliteit van de Reporting Service in onze eigen applicaties in te bouwen. In het plaatje niet getoond, maar de moeite van het vermelden wel waard, is dat er ook een WMI (Windows Management Instrumentation) provider beschikbaar is voor Reporting Services. Dit maakt de Reporting Service o.a. toegankelijk voor Windows Script Host.

URL Access

URL Access is de efficiëntste, maar tevens de meest beperkte manier om Reporting Services te benaderen. Vanuit een ASP.NET-applicatie is het zo simpel als een hyperlink op een webpagina zetten, maar ook vanuit andere types applicaties kunnen we de server via de juiste URL benaderen. Zoals eerder gezegd kunnen we zowel naar de virtual directory van de Report Server browsen als naar de directory die de Report Manager gebruikt. Voorafgaand aan de gewenste virtual directory nemen we uiteraard de servernaam op. We kunnen dus beide onderstaande links gebruiken, ervan uitgaande dat we IIS en Reporting Services op de lokale machine hebben geïnstalleerd. Bovendien hebben we de default namen van de virtual directories aangehouden tijdens installatie.

http://localhost/ReportServer
http://localhost/reports

De eerste URL leidt tot niets anders dan een lijst van beschikbare items op de server. Items kunnen naast rapporten bijvoorbeeld ook gedeelde gegevensbronnen zijn. De tweede URL start de Report Manager waardoor alle functionaliteit van de Report Manager voorhanden komt.

Als de voorbeeldrapporten van Reporting Services zijn geïnstalleerd, zien we in beide bovenstaande gevallen een folder met de naam SampleReports. Om een lijst te krijgen van alle items die zich in deze folder bevinden, moeten we enkele parameters aan de URL toevoegen. We moeten er dan wel rekening mee houden dat er twee soorten parameters zijn. De eerste zijn parameters die we aan de Report Server willen meegeven: bijvoorbeeld welk rapport we willen raadplegen, in welk formaat we het willen zien en eventueel de credentials (login-gegevens) om de gegevensbron te benaderen. Daarnaast zijn er parameters (rapportparameters) die definiëren welke gegevens we naar boven willen halen. Bijvoorbeeld uit welke categorie we producten willen zien. Voor de eerste

soort parameters zijn speciale prefixen gedefinieerd. Een lijst met mogelijkheden staan op de MSDN site (http://msdn.microsoft.com/library/default.asp?url=/library/en-us/RSPROG/htm/rsp_prog_urlaccess_374y.asp) of in BOL (Books Online).

Om de lijst van beschikbare items in de folder SampleReports op te vragen maken we gebruik van de Command parameter met de prefix rs. Deze parameter geven we de waarde ListChildren. Dit levert de volgende URL op:

http://localhost/ReportServer?/SampleReports&rs:Command=ListChildren

We krijgen nu een lijst van rapporten en één algemeen gedefinieerde (gedeelde) datasource. Om nu een rapport daadwerkelijk op te halen geven we de Command parameter de waarde Render. In onderstaande voorbeeldlink geven we ook nog de rapportparameters ReportYear, ReportMonth en EmpID een waarde mee om het overzicht van werknemer 29 voor februari 2004 te krijgen:

http://localhost/ReportServer?/SampleReports/Employee Sales Summary&rs:Command=Render&ReportYear=2004&ReportMonth=2&EmpID=29

De volgende URL tenslotte bevat het productoverzicht van AdventureWorks (de nieuwe voorbeelddatabase van Microsoft) als PDF. Als we deze URL opgeven, krijgen we automatisch een download-dialoog die vraagt waar we het PDF-bestand willen opslaan.

http://localhost/ReportServer?/SampleReports/Product Catalog&rs:Command=Render&rs:Format=PDF

De webservice benaderen vanuit .NET

Zoals eerder gemeld maken we ook gebruik van het feit dat Reporting Services als webservice geïmplementeerd is waardoor de functionaliteit vanuit een .NET-applicatie kan worden benaderd. We doen dat hier in twee stappen. We maken een Windows form met daarop een listbox control (lstReports). Bij het opstarten vragen we via de webservice een lijst op van alle op de server gepubliceerde rapporten. Klikken op een rapport haalt dit rapport vervolgens op waarna een Excel-bestand ontstaat.

Vanuit Visual Studio maken we een nieuw project. We werken dit voorbeeld uit met een VB.NET Windows-applicatie, maar uiteraard is C# of ASP.NET ook mogelijk. De belangrijkste stap is het aanmaken van een web reference die verwijst naar onze Reporting Service webservice (naar de Web Service Description Language om precies te zijn). Het mooie van Visual Studio is dat deze een proxy class creëert met als naam de naam die wij aan de web reference geven. Afbeelding 3 laat de 'Add Web Reference'

Afbeelding 1.

```
Private Sub Form1_Load(ByVal sender As Object, _
  ByVal e As System.EventArgs) Handles MyBase.Load

  ' Maak een ReportingService object en zet windows authentication aan.
  Dim rs As New ReportProxy.ReportingService
  rs.Credentials = System.Net.CredentialCache.DefaultCredentials

  'Maak en vul een array met rapport elementen
  Dim elementen As ReportProxy.CatalogItem() = Nothing
  elementen = rs.ListChildren("/", True)

  'Toon alleen rapporten in een listbox
  If Not (elementen Is Nothing) Then
      Dim ci As ReportProxy.CatalogItem
      For Each ci In elementen
          If ci.Type = ReportProxy.ItemTypeEnum.Report Then
                lstReports.Items.Add(ci.Path)
  End If
  Next ci
  End If

End Sub
```

Listing 1.

dialoog zien. In dit voorbeeld gaan we op zoek naar de webservice op de computer w2k-peter, beschreven door de ReportService WSDL. Nadat we op "Add Reference" klikken, hebben we dus een class genaamd ReportProxy (let op de naam in het tekstvak "Web reference name"). Deze class is de SOAP proxy waarin de klasse ReportingService zit.

De code in listing 1 instantieert als eerste een nieuw object, genaamd rs, van onze ReportingService class. Voordat we de nodige methods kunnen aanroepen, moeten we de authenticatie regelen. Als we dat niet doen, krijgen we de foutmelding "The request failed with HTTP status 401: Access Denied". Hoewel ReportingService geen eigen authenticatie doet (het vertrouwt hiervoor op IIS) hebben we maar twee opties: basic authentication en windows authentication. In de code kiezen we voor de veiligste van de twee en gebruiken we het .NET framework om de Windows credentials van de gebruiker op te halen en te koppelen aan de credentials eigenschap van ons ReportingService-object.

Zoals te verwachten heeft de ReportingService class veel properties en methods die we via de BOL of de object browser kunnen bekijken. In de voorbeeldcode maken we gebruik van de method ListChildren, die ook aangeroepen wordt door de ListChildren command die we bij URL access hebben gezien. Deze method geeft een array van CatalogItem terug. Een CatalogItem-object representeert een object in de Report Server waarvan we via de property Type het soort object kunnen uitvragen (Unknown, Folder, Report, Resource, LinkedReport of DataSource). Uiteindelijk filtert de code van listing 1 alle rapporten uit het array en zet die, inclusief het volledige pad, in de listbox control lstReports.

In listing 2 gebruiken we de method Render van de ReportingService class. Net zoals we bij de method ListChildren zien, geldt ook hier dat de Render parameter die we eerder bij URL access gebruikten niets anders doet dan deze method aanroepen. De method heeft 12 parameters, via welke we onder andere kunnen meegeven met welke credentials de onderliggende datasource moet worden benaderd. Ook bevat het de parameters die nodig zijn om dit rapport op te halen en uiteraard informatie welk rapport we willen ophalen en in welk formaat. De method Render geeft een byte array terug die we vervolgens via een FileStream-object in een bestand kunnen zetten.
Uiteraard is er een hoop aan te merken op deze (sterk vereen-

```
Private Sub lstReports_SelectedIndexChanged(ByVal sender _
  As System.Object, ByVal e As System.EventArgs) _
  Handles lstFolders.SelectedIndexChanged

  ' Maak een ReportingService object en zet windows authentication aan.
  Dim rs As New ReportProxy.ReportingService
  rs.Credentials = System.Net.CredentialCache.DefaultCredentials

  ' Haal het geselecteerde rapport op als EXCEL bestand.
  Dim rapport As Byte() = Nothing
  rapport = rs.Render(lstReports.SelectedItem, "EXCEL", Nothing, _
                  Nothing, Nothing, Nothing, Nothing, _
                  Nothing, Nothing, Nothing, Nothing)

  ' Schrijf het rapport weg op de c schijf.
  Dim stream As FileStream = _
               File.Create("c:\rapport.xls", rapport.Length)
  stream.Write(rapport, 0, rapport.Length)
  stream.Close()
End Sub
```

Listing 2.

voudigde) code. Het eerste wat opvalt is het ontbreken van goede foutafhandeling. Meer functioneel gezien zijn er een aantal problemen te verwachten door alle niet zinvol ingevulde parameters. De vijfde parameter van de method Render, genaamd Parameters()), moet bijvoorbeeld een array van ParameterValue-objecten zijn. Elk rapport waarvoor rapportparameters zijn gedefinieerd genereert een foutmelding als we het met de code van listing 2 proberen op te vragen. Deze foutmelding ontstaat doordat we de vijfde parameter van de method Render niet hebben ingevuld. De method GetReportParameters van het ReportingService-object kan ons helpen de nodige informatie te verzamelen. De method geeft namelijk een array van ReportParameter-objecten terug. Na aanroep van de method weten we dus welke rapportparameters er zijn en welke daarvan geen defaultwaarde hebben. Voor die rapportparameters moeten wij dus waarden meegeven.

Een ander aspect waarmee rekening moet worden gehouden is de onderliggende authenticatie die de database uitvoert. De code in listing 2 gaat er van uit dat alle credentials zijn opgenomen in de bij het rapport gedefinieerde datasource. Als dat niet het geval is zullen wij aan de method Render de te gebruiken credentials moeten meegeven. In de voorgaande voorbeelden is te zien hoe we de Reporting Service kunnen benaderen, zowel via URL's als via code. Gebruiken we Visual Studio, dan genereert deze een proxy class die alle moeilijke SOAP-details afhandelt die nodig zijn om de webservice te benaderen. We kunnen uiteraard andere ontwikkelomgevingen gebruiken als we maar een manier hebben om via het SOAP-protocol de webservice te benaderen. Het belangrijkste is de Reporting Service-objectbibliotheek te leren kennen.

Ook de meegeleverde rs.exe utility is de moeite van het vermelden waard. Via deze tool kunnen vanuit de commandline scriptjes uitgevoerd worden die het beheer van de server automatiseert en (dus) vereenvoudigt. Ook in dit geval verzorgt de utility de afhandeling van het SOAP-protocol en kunnen wij ons richten op de functionele kant van het verhaal.

Uitbreidbaarheid

Naast het gebruik van de bestaande functionaliteit is het ook mogelijk (en volgens Microsoft zelfs de bedoeling) dat third party vendors uitbreidingen gaan maken. Uiteraard kunnen wij dat dan ook voor ons zelf gaan doen. Zoals we eerder in het artikel al hebben opgemerkt, is de uitbreidbaarheid van Reporting Services te

```
...
<Extensions>
  <Delivery>
    <Extension Name="Report Server FileShare" ...
    ...
    </Extension>
    <Extension Name="Report Server Email" ...
    ...
    </Extension>
    <Extension Name="NULL" .../>
  </Delivery>

  <Render>
    <Extension Name="XML" Type="Microsoft.ReportingServices.Rendering..
    <Extension Name="NULL" Type=...
    <Extension Name="CSV" Type=...
    <Extension Name="IMAGE" Type=...
    <Extension Name="PDF" Type=...
    <Extension Name="HTML4.0" Type=...
    <Extension Name="HTML3.2" Type=...
    <Extension Name="MHTML" Type=...
    <Extension Name="EXCEL" Type=...
    ...
  </Render>

  <Data>
    <Extension Name="SQL" Type=...
    <Extension Name="OLEDB" Type=...
    <Extension Name="ORACLE" Type=...
    <Extension Name="ODBC" Type=...
  </Data>

  <Security>
    <Extension Name="Windows" Type=...
  </Security>

  <Authentication>
    <Extension Name="Windows" Type=...
  </Authentication>

  <EventProcessing>
    <Extension Name="SnapShot Extension" Type=...
      <Event>
        <Type>ReportHistorySnapshotCreated</Type>
      </Event>
    </Extension>
    <Extension Name="Timed Subscription Extension" Type=...
      <Event>
        <Type>TimedSubscription</Type>
      </Event>
    </Extension>
    <Extension Name="Cache Update Extension" Type=...
      <Event>
        <Type>SnapshotUpdated</Type>
      </Event>
    </Extension>
  </EventProcessing>
</Extensions>
```

Listing 3.

danken aan zowel de modulaire opbouw als aan het feit dat we, met gebruik van het .NET framework, de managed code API's kunnen gebruiken. Daarmee schrijven we assemblies waarmee we de vier modules zoals in afbeelding 1 (Scheduling and Delivery, Rendering, Data Retrieval, Security) naar behoeven kunnen uitbreiden. Het voert binnen het kader van dit artikel te ver om dit proces volledig uit te werken, maar we willen toch kort schetsen hoe dat verloopt.

Als eerste moeten we naar het RSReportServer.config bestand kijken. Dit bestand staat standaard in de folder C:\Program Files\Microsoft SQL Server\MSSQL\Reporting Services\ReportServer. Uit listing 3 volgt de structuur van dit bestand en valt de modulaire opbouw van de Reporting Service goed te herleiden. We zien een Extensions-element met daarbinnen onder andere Render- en Delivery-elementen. Als we bijvoorbeeld een printer delivery extension willen maken, moeten we binnen het Delivery-element een nieuw Extension-element maken. Een zelf geschreven assembly die deze printerextensie implementeert, maakt gebruik van de Microsoft.ReportingServices.Interfaces.dll die in de Reporting Services bin folder staat. In deze dll vinden we onder andere de IExtension en de IDeliveryExtension interface die we moeten implementeren. Als de voorbeelden van Reporting Services zijn meegenomen tijdens de installatie, staat er een uitgewerkt voorbeeld tussen in de samples folder. Ten slotte voegen we een Extension-element toe aan het RSWebApplication.config-bestand om de nieuwe functionaliteit ook terug te vinden in de Report Manager. Om het geheel te laten werken regelen we nog de code access security door onze assembly 'Full Trust' permissie te geven in het rssvrpolicy.config-bestand.

Conclusie

De kracht van Reporting Services is de breedheid van het platform. Alle fases in de levensloop van een rapport worden ondersteund. Door de open architectuur is alle functionaliteit vanuit elke gewenste omgeving te benaderen. De uitbreidbaarheid garandeert dat we nog jaren voort kunnen met dit product. Het groeit mee met de veranderende wensen die de toekomst brengt.

Gratis Reporting Services voor SQL Server 2000 gebruikers

Gebruikers van SQL Server 2000 kunnen direct profiteren van Reporting Services zonder bijkomende licentiekosten. U kunt de volledige versie van Reporting Services op cdrom bestellen (€ 10,- verzendkosten)
http://www.microsoft.com/netherlands/servers/sql/reporting/bestel/default.asp

Nuttige links

- www.microsoft.com/sql/reporting/techinfo/rdlspec.asp
- http://msdn.microsoft.com/library/default.asp?url=/library/en-us/RSPROG/htm/rsp_prog_urlaccess_374y.asp
- http://msdn.microsoft.com/sql

Peter ter Braake en Peter Lemmens zijn respectievelijk productspecialist Microsoft SQL Server en Microsoft CRM bij Compu'Train, de grootste CPLS (voorheen CTEC Gold) in Nederland en marktleider op het gebied van SQL Server trainingen. Peter ter Braake is bereikbaar via pbraake@computrain.nl, www.computrain.nl

POWERTOYS FOR VISUAL STUDIO .NET 2003

PowerToys for Visual Studio .NET 2003 is een verzameling handige hulpprogramma's en tools die Microsoft heeft ontwikkeld als aanvulling op Visual Studio. Met deze eerste release van de PowerToys kunt u verborgen registerinstellingen van Visual Studio aanpassen, commentaar toevoegen en helpbestanden bouwen, bestanden laden vanaf de command-line en aangepaste layouts definiëren voor het IDE-venster. Bij alle releases van PowerToys for Visual Studio .NET 2003 wordt de broncode geleverd, zodat u deze naar hartelust kunt wijzigen.

VSTweak

VSTweak maakt het mogelijk een aantal van de minder bekende opties en instellingen van Visual Studio aan te passen. Met VSTweak kunt u het volgende doen:

* De instellingen voor Advanced Dynamic Help aanpassen
* Importeren en exporteren van eigen keyboard-mappings
* Herkennen van custom bestandsextensies in Visual Studio .NET
* De MRU-lijst (Most Recently Used) met onlangs gebruikte bestanden en projecten bewerken en wissen
* Bekijken en bewerken van aliassen voor het opdrachtvenster

VSEdit

Elke keer als u een bestand probeert te openen wordt een nieuwe Visual Studio-sessie geopend. Dat kan irritant zijn. VSEdit is een utility voor de command-line waarmee u een of meer bestanden kunt laden in de actieve Visual Studio .NET 2003-sessie. Geen gedoe meer met extra vensters die worden geopend. Gewoon laden en programmeren.

VB Commenter

Het toevoegen van commentaar aan uw Visual Basic-broncode is nu makkelijker dan ooit te voren. Na installatie van deze PowerToy typt u eenvoudigweg ''' in de Visual Basic code-editor en drukt u op Enter, waarna de juiste XML-commentaartags verschijnen. U kunt zelfs een XML-bestand maken van alle commentaar in een bepaalde class library en dit zo gebruiken in de PowerToy Custom Help Builder. Op die manier maakt u in een handomdraai een professioneel Help-pakket ter ondersteuning van uw Visual Basic-oplossing.

VSWindowManager

Met VSWindowManager kunt u zelf vensterlay-outs ontwerpen. Deze PowerToy voegt een extra menu-item toe aan het menu Windows van Visual Studio, waarmee u uw favoriete vensterindeling kunt opslaan en laden. Wanneer u van de codelay-out overschakelt op de designlay-out, worden automatisch de opgeslagen vensterposities geladen.

Custom Help Builder

Met deze PowerToy kan elke Basic- of Visual C#-programmeur een eigen Help-file maken van een class library. Uw aangepaste Help wordt volledig geïntegreerd in Visual Studio .NET 2003, zodat uw onderwerpen beschikbaar zijn vanuit de dynamische Help, F1, de zoekfunctie, de Help-index en de Helpinhoudsopgave.

VSCMDShell Window

U wilt het Visual Studio Commands Window en uw externe CMD.exe-proces vanuit één venster bedienen? Genoeg van het gedoe met de shell-command vanuit Visual Studio? Dan is dit toolvenster iets voor u!

VSMouseBindings

Uw muis heeft vijf knoppen? Waarom zou u niet aan elke muisknop een Visual Studio-opdracht toewijzen?

Meer Power Toys zijn te vinden op

http://www.gotdotnet.com/team/ide/

Meewerken aan een van de Power Toys kan natuurlijk. Voor elke project is er een eigen workspace ingericht met source-control, bugtracker, documentatie en message-board.