Dieudonné Foko is developer bij Macaw

Nooit meer verdwalen met MapPoint

POSITIE EN ROUTE BEPALEN MET MAPPOINT WEBSERVICES

Google Maps en MSN Virtual Earth hebben mapping en GIS (Geographic Information Systems) weer op de kaart gezet. De Microsoft MapPoint Product Group is een vrij onbekende groep, maar bestaat intussen al meer dan negen jaar en werkt aan cartografie en spatial technology. Producten die deze groep heeft voortgebracht, zijn onder andere Auto Route, MapPoint 2002, MapPoint.NET, Streets and Trips, Pocket Streets, MSN Mobile en zeer recent MSN Visual Earth. Naast deze producten biedt deze groep ook een aantal diensten in de vorm van webservices. Deze webservices zijn alleraardigst en daarom zullen we ze in dit artikel eens nader te bekijken.

De MapPoint-productfamilie bestaat grofweg uit drie onderdelen: MapPoint 2004, MapPoint Location Server en MapPoint Web Service. Voor alledrie de MapPoint- producten zijn API's beschikbaar. Afhankelijk van de situatie en de behoefte kun je voor een van de drie kiezen. Een factor die de keuze direct bepaalt is het feit of jouw device wel of niet verbonden is met internet. Wanneer er een internetverbinding is, kun je de webservice van MapPoint gebruiken. Het voordeel hiervan is dat de locatie-informatie en kaarten volledig up-to-date zijn.

MapPoint 2004

Van MapPoint 2004 bestaan er twee versies: een Noord-Amerikaanse en een Europese versie. Microsoft MapPoint 2004 is een stand-alone product voor het maken van plattegrondjes, routekaartjes en het analyseren van demografische informatie. Het is namelijk mogelijk om in MapPoint een Excel-spreadsheet in te lezen met verkoopcijfers per postcodegebied. MapPoint kan deze cijfers grafisch weergeven op de kaart, zodat je kunt zien in welke regio het meest wordt verkocht. MapPoint bevat zelfs al interessante demografische informatie van aantal Europese landen. Zo is voor Nederland onder meer het aantal inwoners per postcodegebied, de gemiddelde leeftijd, gemiddeld jaarinkomen, het aantal personen per huishouden beschikbaar, allemaal ingedeeld per postcode of provincie. Een onderdeel van MapPoint is het programma Microsoft Pocket Streets voor op de Windows Mobile Pocket PC of Smartphone. Met behulp van MapPoint kun je zelf kaartjes maken voor Pocket Streets. Wanneer je een GPS-ontvanger aansluit op jouw computer of Pocket PC zal MapPoint of Pocket Streets de actuele positie op de kaart weergeven. Het is vrij eenvoudig om in MapPoint te programmeren. Om de verschillende functies van MapPoint aan te roepen is een ActiveX-control beschikbaar. Deze kun je downloaden van het MapPoint development center.

MapPoint Location Server

Met de MapPoint Location Server is het mogelijk om met bijvoorbeeld een Pocket PC of Smartphone real-time positie-informatie op te vragen van mobile providers en deze te combineren met informatie van de MapPoint Web Service. MapPoint Location Server is bedoeld voor fleet-management, asset-tracking, servicediensten en verkoopteams.

MapPoint Web Service

De MapPoint Web Service biedt onder andere navigatie-instructies, afstandberekening, routekaartjes, points of interest, proximity-searches, geocoding en reverse geocoding. Deze service is ontwikkeld en wordt gehost door Microsoft. De dienst wordt al geruime tijd gebruikt door grote internationale organisaties en er wordt dan ook een uptime van 99.9% gegarandeerd. De kaarten komen van NavTech en Tele Atlas en worden constant bijgewerkt. Dekking van deze mappingservice is voor heel Noord-Amerika, grote delen van Europa en sinds 2002 is er voor België en Nederland 100% dekking. Op vrij eenvoudige wijze kunnen we zelf geografische informatie toevoegen aan eigen applicaties. Mogelijke scenario's zijn bijvoorbeeld het aanbieden van een gepersonaliseerde routekaart op een website. In dit artikel zullen wij verder in gaan op de MapPoint Web Service. Ik zal in eerste instantie de vier belangrijke MapPoint Web Services API's beschrijven. Vervolgens doorlopen we met behulp van codevoorbeelden de stappen die nodig zijn voor het hele proces van zoeken, berekenen en renderen en het tonen van de kaarten op een site. Ik zal ook toelichten hoe je een omleiding op een specifieke weg kunt implementeren. Laten we beginnen met de API's van MapPoint Web Service.

MapPoint Web Service API's

De API's van de MapPoint Web Services zijn gegroepeerd in vier verschillende services.

- Common Service, die gebruikt wordt om algemene metadata van de webservice op te vragen. Deze meta-informatie betreft de DataSource, het versienummer en de landspecifieke informatie.
- Zoekservice, die geraadpleegd wordt om bijvoorbeeld een plaats, adres, postcode of straat op te zoeken. Je kunt ook zoeken met andere zoekcriteria zoals de breedtegraad (latitude) of lengtegraad (longitude) van een bepaald punt.
- Route Service. Deze service wordt aangeroepen om een route tussen twee eindpunten te berekenen.
- Render Service, die gebruikt wordt om een kaart met de bijbehorende symbolen weer te geven.

In de rest van dit artikel zal ik demonstreren hoe je gebruik kunt maken van de vier MapPoint-webservices om een routekaart te implementeren.

```
1. const string DataSource = "MapPoint.EU";
2. const string Culture = "nl-NL";
3. private CommonServiceSoap commonService;
4. private FindServiceSoap findService;
5. // declaration omitted for render and route soap services
6. private void InitializeMPNetConnection()
7. {
8. NetworkCredential ourCredentials = new NetworkCredential(
   ConfigurationSettings.AppSettings["MPUser"],
   ConfigurationSettings.AppSettings["MPPass"]);
9. CultureInfo culture = new CultureInfo();
10.  CountryRegionContext context = new CountryRegionContext();
11.  DistanceUnit distance = new DistanceUnit();
12.  //common service initialization
13.  UserInfoHeader headerCommon = new UserInfoHeader();
14.  headerCommon.Culture = culture;
15.  haderCommon.Context = context;
16.  headerCommon.DefaultDistanceUnit = distance;
17.  commonService = new CommonServiceSoap();
18.  commonService.Credentials = ourCredentials;
19.  commonService.PreAuthenticate = true;
20.  commonService.UserInfoHeaderValue = headerCommon;
21.  //findservice initialization
22.  UserInfoFindHeader headerFind = new UserInfoFindHeader();
23.  headerFind.Culture = culture;
24.  headerFind.Context = context;
25.  headerFind.DefaultDistanceUnit = distance;
26.  findService = new FindServiceSoap();
27.  findService.Credentials = ourCredentials;
28.  findService.PreAuthenticate = true;
29.  findService.UserInfoFindHeaderValue = headerFind;
30.  //initialization omitted for the render and route services
31.  }
```

Codevoorbeeld 1. De webservices initialiseren

Stap 1: Initialiseren van de webservices

Voordat ik de initialisatiecode in codevoorbeeld 1 toelicht moeten we eerst een paar webreferenties toevoegen aan ons project. Er zijn twee URL's die toegang geven tot de MapPoint-webservices:

- een voor de staging/developmentomgeving
  http://findv3.staging.mappoint.net/Find-30/Common.asmx
- een voor de productieomgeving (niet gratis)
  http://findv3.mappoint.net/Find-30/Common.asmx

Om gebruik te maken van de developmentomgeving is een account nodig. Zie daarvoor de 'Licentie'-sectie van dit artikel. De initialisatiestap bestaat uit het creëren van de variabelen en de juiste waarde er aan toekennen. Op regel 1 van codevoorbeeld 1 zie je dat de stringwaarde 'MapPoint.EU' wordt toegekend aan de DataSource-constant. De MapPoint Web Service biedt op dit moment 14 datasources, bovendien kun je zelf eigen datasources uploaden. Voorbeelden van andere DataSources zijn: 'MapPoint.NA' (voor kaarten uit Noord-Amerika), 'MapPoint.World' (voor kaarten van de wereld), 'MapPoint.Icons' (voor de pictogrammen die op een kaart getoond worden), 'MapPointNAICS.NA' (voor alle hotels, restaurants, enzovoort). Elke datasource biedt een of meer mogelijkheden (CanFindNearby, CanDrawMaps, CanFindPlaces, CanRoute, CanFindAddress of HasIcons). Er bestaat zelfs een datasource voor de maan: 'MapPoint.Moon'. Daarmee kan je kaarten van de maan opvragen! Voorlopig beperken we ons tot routekaartjes met Nederlandstalige instructies en aanwijzingen; dat wordt aangegeven in codevoorbeeld 1 op regel 2.

De CommonServiceSoap en de FindServiceSoap zijn twee van de vier soap-objecten die de MapPoint Webservices implementeren. Elk object bevat de benodigde methodes die wij later zullen aanroepen om allerlei queries op de webservice uit te voeren. Wat we eerst voor elk soapobject moeten doen, is het instellen van de waarden die de property van het soap-object heeft. Bijvoorbeeld in codevoorbeeld 1 op regel 17 t/m 20 zie je dat een instantie van het CommonServiceSoap-object wordt gemaakt en vervolgens wordt er voor die instantie de waarden van de properties ingesteld zoals de Credentials, PreAuthenticate en UserHeaderInfoValue. De Credentials-property bevat informatie uit het account die gemachtigd is om de MapPoint Webservice te consumeren (zie regel 8, codevoorbeeld 1). De UserHeaderInfoValue encapsuleert de cultuurinstellingen (bijvoorbeeld 'nl-NL', 'fr-FR', etc), de context en de afstandeenheden (bijvoorbeeld 'kilometers' of 'miles'). De UserHeaderInfoValue is gedefinieerd op regel 13 t/m 16. De andere twee MapPoint Web Services (RouteServiceSoap en RenderServiceSoap) kun je op hetzelfde manier initialiseren zoals in codevoorbeeld 1 is gedaan.

Stap 2: Zoeken naar de juiste kaart met adresgegevens.

De FindServiceSoap van MapPoint biedt de mogelijkheid om op diverse manieren te zoeken. Je kunt zoeken op EntityID. Hierbij wordt gezocht op een (arbitraire) waarde die gekoppeld is aan een gebied of aan een land. Je kunt ook op property zoeken. Hierbij gebruik je de FindByProperty(…)-methode van de FindServiceSoap. De FindByProperty(…)-methode neemt een FindByPropertySpecification-object dat bijvoorbeeld de latitude- en longitude-coördinaten bevat als parameter. Verder kun je ook zoeken naar alle plaatsen die zich binnen een bepaalde straal of afstand van een referentiepunt bevinden. Hiervoor gebruik je de FindNearBy(…)-methode. Denk bijvoorbeeld aan het zoeken naar alle dichtstbijzijnde filialen van een winkel. Ten slotte biedt de MapPoint Web Service ook methoden om per adres te zoeken. Omdat

```
1. public FindResults foundResults;
2. public double topScore;
3. public void ExecuteSearchByFormattedAddress(string formattedAddress)
4. {
5.    topScore = 0;
6.    foundResults = null;
7.    FindRange myFindRange = new FindRange();
8.    myFindRange.Count = 10;
9.    myFindRange.StartIndex = 0;
10.
11.   FindOptions myFindOptions = new FindOptions();
12.   myFindOptions.ThresholdScore = 0;
13.   myFindOptions.Range = myFindRange;
14.
15.   FindService.UserInfoFindHeaderValue.Culture.Name = Culture;
16.
17.   if (formattedAddress.Length > 0)
18.   {
19.     formattedAddress += ", Nederland";
20.     Address address = new Address();
21.     address.FormattedAddress = formattedAddress;
22.     FindAddressSpecification findSpec = new
23.     FindAddressSpecification();
24.     findSpec.InputAddress = address;
25.     findSpec.DataSourceName = DataSource;
26.     findSpec.Options = myFindOptions;
27.     try
28.     {
29.       foundResults =
30.       FindService.FindAddress(findSpec);
31.     }
32.       catch(System.Exception ex)
33.     {
34.       ///find address method has timed out.
35.     }
36.     if (foundResults != null &&   foundResults.Results.Length >0)
37.     {
38.       topScore = foundResults.TopScore;
39.     }
40.   }
41. }
```

Codevoorbeeld 2. Het vinden van een locatie

```
1. public void GetLocationMap(string placeName, int ImageWidth,
     int ImageHeight)
2. {
3.    ExecuteSearchByFormattedAddress(placeName);
4.
5.    ViewByHeightWidth[] myViews = new ViewByHeightWidth[1];
6.    myViews[0] = foundResults.Results[0].FoundLocation.BestMapView.
        ByHeightW idth;
7.    Location startLocation = foundResults.Results[0].FoundLocation;
8.    MapOptions mapOptions = new MapOptions();
9.    mapOptions.ReturnType = MapReturnType.ReturnUrl;
10.   mapOptions.Format = new ImageFormat();
11.   //set the width and height to whatever size the map is on the page
12.   mapOptions.Format.Width = ImageWidth;
13.   mapOptions.Format.Height = ImageHeight;
14.   mapOptions.Zoom = 1.8;
15.   Pushpin[] myPushpins = new Pushpin[1];
16.   myPushpins[0] = new Pushpin();
17.   myPushpins[0].IconDataSource = "MapPoint.Icons";
18.   myPushpins[0].IconName = "29";
19.   string[] values = placeName.Split(new char[] { ',' });
20.   myPushpins[0].Label = string.Format("{0}{1}{2}", values[0],
        '\n',values[1]);
21.   myPushpins[0].LatLong = startLocation.LatLong;
22.   MapSpecification mapSpec = new MapSpecification();
23.   mapSpec.DataSourceName = DataSource;
24.   mapSpec.Pushpins = myPushpins;
25.   mapSpec.Options = mapOptions;
26.   mapSpec.Views = myViews;
27.   MapImage[] mapImages;
28.   try
29.   {
30.      mapImages = renderService.GetMap(mapSpec);
31.      //code omitted
32.   }
33.   catch(Exception e)
34.   {
35.      //write exception to log file
36.   }
37. }
```

Codevoorbeeld 3. Map rendering

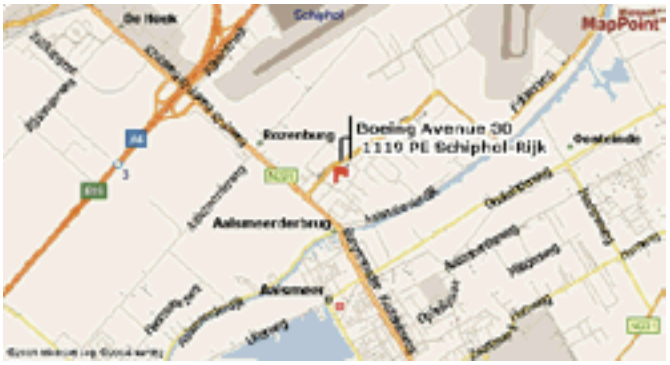
ik er vanuit ga dat de invoerparameters een adres zullen zijn, gaan wij verder in op het zoeken per adres. De methode die de zoekactie uitvoert, staat in codevoorbeeld 2. Op regel 1 en 2 worden twee variabelen gedeclareerd: foundResults en topScore. De eerste variabele gebruiken we om het zoekresultaat op te slaan, terwijl de andere variabele wordt gebruikt om het zoekresultaat vast te leggen dat de hoogste score heeft opgeleverd; het meest nauwkeurige zoekresultaat dus. De topScore-variabele kan je ook gebruiken om te controleren of een zoekcriterium ambigu is.
De belangrijkste regel uit het codefragment 2 is regel 29 waar de methode FindAddress(…) aangeroepen wordt. De methode neemt als parameter een adresspecificatieobject (FindAddressSpecification) en hij retourneert een FindResults-object. Ik kom later (in codevoorbeeld 3) terug op de eigenschappen van een FindResults-object. Maar nu eerst het FindAddressSpecification-object. Een object dat een specificatie is van een adres bevat eigenschappen zoals een InputAddress (zie regel 24), de datasource en de zoekopties (om bijvoorbeeld het minimale en maximale aantal zoekresultaten te definiëren). De InputAddress is de enige property van de FindAddressSpecification die niet mag ontbreken. Deze bevat namelijk een string die gelijk staat aan een geformatteerd adres. Zo'n string, die als paramater wordt meegegeven, kan bijvoorbeeld een waarde zijn als `"Boeing Avenue 30,1119 PE Schiphol-Rijk, Nederland"` of `",1119 PE,Nederland"`. Nadat de ExecuteSearchByFormattedAddress(...)-methode is verwerkt, wordt het beste zoekresultaat opgeslagen. Dit resultaat gaan we in de volgende stap gebruiken.

Stap 3: Renderen van de kaart die bij het zoekresultaat hoort

Voordat je de kaarten die bij een zoekresultaat horen kunt laten renderen, moet je eerst bepalen in welke view die kaarten getoond worden. Daarna moet je aangeven of de kaart als een image geretourneerd wordt in plaats van een URL die naar de image verwijst. Ten derde kan je ook specificeren in welk formaat de kaart wordt opgeleverd. En als laatste bepaal je welke (pushpins) icoontjes op de kaart getekend worden. Om de bovengenoemde vier aandachtpunten verder toe te lichten duiken we in de body van de GetLocationMap(...)-methode die in codevoorbeeld 3 staat. Op regel 3 staat de aanroep naar de ExecuteSearchByFormattedAddress(…)-methode die ik eerder heb toegelicht. De aanroep op regel 3 zorgt ervoor dat de findResults- variable (die onder andere een array van FindResult bevat) wordt ingevuld.
Om te bepalen in welke view ons kaartje getoond zal worden, biedt de MapView-class vier verschillende viewtypes.

- ViewByScale: waarbij de view afhangt van de schaal en het middenpunt
- ViewByBoundingRectangle: de view die is gebaseerd de latitude en longitude van twee punten van rechthoek, namelijk de noordoost- en zuidwest-hoeken
- ViewByBoundingLocation: de view die bepaald wordt, is afhankelijk van een (verzameling) locatie(s) die de kaart moet bevatten
- ViewByHeightWidth: waarbij de view is gekoppeld aan een bepaalde hoogte en breedte. Wij gebruiken het ViewByHeightWidth-object op regel 4 en 5.

Het locatieobject (Location) dat bij een FoundResult hoort, gebruiken we later om bijvoorbeeld een pushpin op die locatie te tekenen door gebruik te maken van de latitude- of longitude-property van het locatieobject. Een locatieobject wordt in regel 6 gedefinieerd en verder gebruikt op regel 21. De returntype van de map is de volgende variabele die gedefinieerd moet worden. Zoals ik al eerder heb vermeld, zijn er meer mogelijkheden voor de returntype van een kaart. Je kunt er bijvoorbeeld voor kiezen om de kaart als image te retourneren, of de URI's (ook secure URL) van de image zelf. In dit geval zou je een Image-object kunnen maken en de ImageSource-property laten verwijzen naar de URL. Voor beide keuzen (ReturnImage en ReturnURL) geldt dat je een MapOptions-object moet maken en vervolgens zet je de ReturnType-property van MapOptions op een van de waarden uit de enumeratie van MapReturnType(zie regel 9). Het formaat van de afbeelding is een andere property van de MapOptions die gezet moet worden. Op regel 10 t/m 13 zie je dat een nieuw formaat van de afbeelding wordt gedefinieerd. Vervolgens worden de breedte en de hoogte van het formaat gezet aan de waarden die in de parameters zijn doorgespeeld.
Als laatste wil ik een pushpin-icoontje met tekstuele aanwijzingen op een kaart tekenen. Daarvoor declareren we een array die voorlopig uit een Pushpin-object bestaat (zie regel 15, codevoorbeeld 3). De IconDataSource, de IconName, het Label en het punt waarop het pushpin-icoontje moet verschijnen, zijn achtereenvolgens gedefinieerd op regel l7 t/m 21. De IconName die we gebruiken



Afbeelding 1. Kaartje met locatieaanduiding

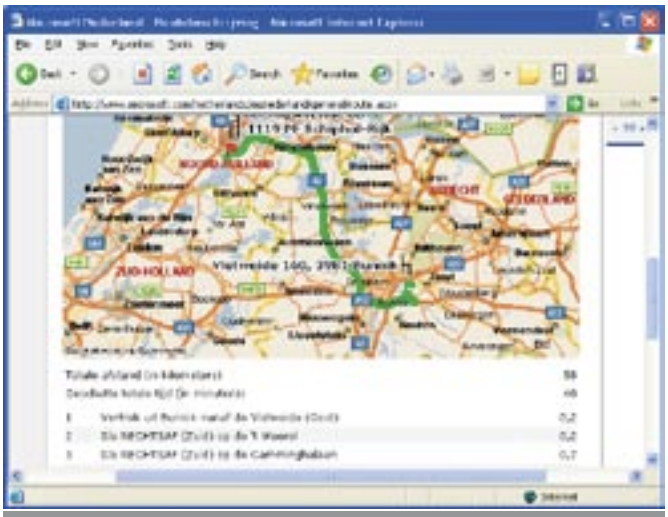
is nummer '29' die hardgecodeerd is aan een rood vlaggetje. Het Label is de displaytekst die naast de pushpin zal verschijnen en in ons geval nemen wij daarvoor gewoon de straat- en plaatsnaam uit het adres. Wij hebben al de view, URL en het formaat van de kaart, plus de pushpin voor op de kaart gedefinieerd. We hoeven alleen nog maar een object te creëren dat al die informatie zal encapsuleren en doorspelen aan de RederSoapService. Dit gebeurt op regel 22 t/m 26. De GetMap(…)-methode rendert en retourneert een array van kaartjes (MapImage) die voldoen aan de specificatie die als parameter is doorgeven. Zo'n specificatie, die dus bijvoorbeeld de gewenste view, of URL kan zijn, wordt aangeboden in de vorm van een MapSpecification-object. Afbeelding 1 is het eerste kaartje uit de array van MapImage. Zoals je kunt zien op afbeelding 1 is er geen route op het kaartje getekend. Die komt in de volgende stap.

Stap 4: Routekaart berekenen

Als je een routekaart wilt tekenen, dan zal je eerst moeten aangeven hoe die routekaart opgebouwd moet worden. In codevoorbeeld 4, regel 28 zie je hoe een route die voldoet aan de routespecificatie die op regel 23 gedefinieerd staat, berekend wordt. Een RouteSpecification-object bevat informatie zoals het aantal segmenten van een route en de routeopties (bijvoorbeeld snelste of kortste route) voor elk segment. Een belangrijke property van een segment is de Waypoint. Een Waypoint bepaalt het start- of eindpunt of de tussenstop van een route. Bij de twee segmenten die wie hebben gedefinieerd, horen twee aparte Waypoint-objecten: een voor de beginlocatie en de andere voor de eindlocatie (zie regel 12 t/m 20). Een andere property van een Route-object, die wel belangrijk is om te noemen, is de Itinerary (reisbeschrijving) die alleen gegenereerd wordt op het moment dat de methode CalculateRoute(..) van de

```
1. public void CalculateRouteByBeginLocation(string address,
     string beginFormattedAddress, string routeType, int ImageWidth,
     int ImageHeight, double zoomfactor)
2. {
3.   //code omitted.
4.   //fill the foundResults object as shown in Listing 2
5.   Location endLocation =   foundResults.Results[0].FoundLocation;
6.   Location startLocation =    foundResults.Results[0].FoundLocation;
7.   //code fragment omitted
8.   Pushpin[] myPushpins = new Pushpin[2];
9.   //initialize pushing as shown in Listing 3
10.    myPushpins[0].LatLong = startLocation.LatLong;
11.  myPushpins[1].LatLong = endLocation.LatLong;
12.
13.  SegmentSpecification[] routeSegmentsSpec = new SegmentSpecification[2];
14.  routeSegmentsSpec[0] = new SegmentSpecification();
15.  routeSegmentsSpec[0].Waypoint = new Waypoint();
16.  routeSegmentsSpec[0].Waypoint.Name = startLocation.Address.PrimaryCity;
17.  routeSegmentsSpec[0].Waypoint.Location = startLocation;
18.  routeSegmentsSpec[1] = new SegmentSpecification();
19.  routeSegmentsSpec[1].Waypoint = new Waypoint();
20.  routeSegmentsSpec[1].Waypoint.Name = endLocation.Address.PrimaryCity;
21.  routeSegmentsSpec[1].Waypoint.Location = endLocation;
22.  SegmentOptions segmentOptions = new SegmentOptions();
23.  segmentOptions.Preference = SegmentPreference.Shortest;
24.  RouteSpecification routeSpec = new RouteSpecification();
25.  routeSpec.DataSourceName = DataSource;
26.  routeSpec.Segments = routeSegmentsSpec;
27.  routeSpec.Segments[0].Options = segmentOptions;
28.
29.  Route myRoute = routeService.CalculateRoute(routeSpec);
30.  MapSpecification mapSpec = new MapSpecification();
31.  mapSpec.Route = myRoute;
32.  //set other properties for mapSpec such as pushpins
33.  //datasource and options. See Listing 3.
34.  MapImage[] mapImages = renderService.GetMap(mapSpec);
35.  //code omitted.
36.  }
37. }
```

Codevoorbeeld 4. Route berekenen



Afbeelding 2. Kaartje met route

RouteService wordt aangeroepen. Een itinerary-object bevat routeinformatie die wij nodig hebben zoals de totale reistijd, de totale routeafstand, en de rij-instructies per segment.
Op afbeelding 2 zie je het eindresultaat van de routeberekening. Je kunt de applicatie ook live zien: http://www.microsoft.com/netherlands/msnederland/general/route.aspx. Ik heb een paar ASP.NET-usercontrols op een aspx-pagina gesleept en de properties daarvan gezet. De usercontrols die ik heb gebruikt zijn: een Image, Labels en een Repeater. De ImageUrl-propery is gelijk gezet aan URL van de MapImage, de Labels zijn gebruikt voor de afstand en reistijd. De DataSource-property van de Repeater is gezet aan het Segment-array van een Route en voor elk segment lezen we de rij-instructies af.

Makkelijk in gebruik

In dit artikel heb ik laten zien hoe je op een makkelijke manier gebruikmaakt van de MapPoint-webservice om een routekaart op een website te tonen. De route-informatie klopt niet altijd. Bijvoorbeeld in het geval van een omleiding of file. Dit komt omdat de MapPoint webservice helaas nog geen 'avoid areas' kent zoals de cd-romversie van MapPoint. Als work-around kun je vrij gemakkelijk een omleiding langs een bepaalde weg of rondom een gebied implementeren door een extra Waypoint te definiëren en vervolgens af te dwingen dat een route langs dat waypoint gaat. Een andere beperking van de MapPoint-webservice is dat er naast autowegen geen andere transporttypes ondersteund wordt (bijvoorbeeld de fiets; er zijn geen fietspaden, waterwegen of spoorwegen opgenomen). In dat geval zou je andere MapPoint Server-producten kunnen combineren om het gewenste resultaat te krijgen. Maar hoe dan ook, je kunt in principe alles realiseren met MapPoint-producten.

Dieudonné Foko (dieudonne.foko@macaw.nl) is developer bij Macaw (www.macaw.nl). Hij heeft aan de Universiteit Utrecht informatica gestudeerd. Dieudonné heeft in het verleden voor Microsoft en de belastingdienst gewerkt voor welke laatste hij een minicompiler voor de OCL-taal heeft ontwikkeld. Zijn interesses zijn codeperformance, security, architectuur en compilertools. Dieudonné is Microsoft Certified Professional.

Nuttige internetadressen

MapPoint development center: http://msdn.microsoft.com/mappoint/
MapPoint trainingen : http://msdn.microsoft.com/mappoint/mappointtrain/
MapPoint multi-platform assistance center: http://go.mappoint.net/mappointmpac/
MapPoint Web Service examples: http://demo.mappoint.net/fourthcoffeecompany
MapPoint Web Service SDK 4.0: http://go.microsoft.com/fwlink?LinkID=23505
MapPoint Location Server SDK: http://msdn.microsoft.com/library/en-us/mslocservsdk10/mels10_sdk_introduction.asp
Voorbeeld route per afslag: http://msdn.microsoft.com//library/en-us/dnmapnet30/html/mwscombine.asp