KARTENMATERIAL UND MEHR MIT ESRI, TEIL 1

Schatzkarten in .NET

Features wie präzise Adressangaben oder das Routing zur Wegbeschreibung werden in Anwendungen immer wichtiger. Esri macht diese Funktionen auch für .NET verfügbar.

Seit der Einführung von Google Maps Anfang Februar 2005 hat sich viel getan. Und das nicht nur technologisch und inhaltlich, wie etwa in Gestalt besserer und detaillierterer Karten; auch bei den Nutzern von Anwendungen ist eine nicht zu unterschätzende Anspruchshaltung entstanden. Das hat zwar eine negative Konnotation, ist aber gar nicht so drastisch negativ gemeint. Google Maps hat es allerdings unbestritten geschafft, Anwendern in Windeseile zu zeigen, was mit Kartenmaterial und Kartendaten alles möglich ist. Wenn es um Adressdaten geht, ist eine Validierung der eingegebenen Daten bereits das absolute Minimum. Ebenso oft wird eine Karte verlangt, um zum Beispiel eine Adresse zu visualisieren oder eine Strecke von Wegpunkten zu berechnen. Das sind alles beliebte Funktionen, sodass immer mehr Webseiten, Apps und Co. diese Möglichkeiten anbieten.

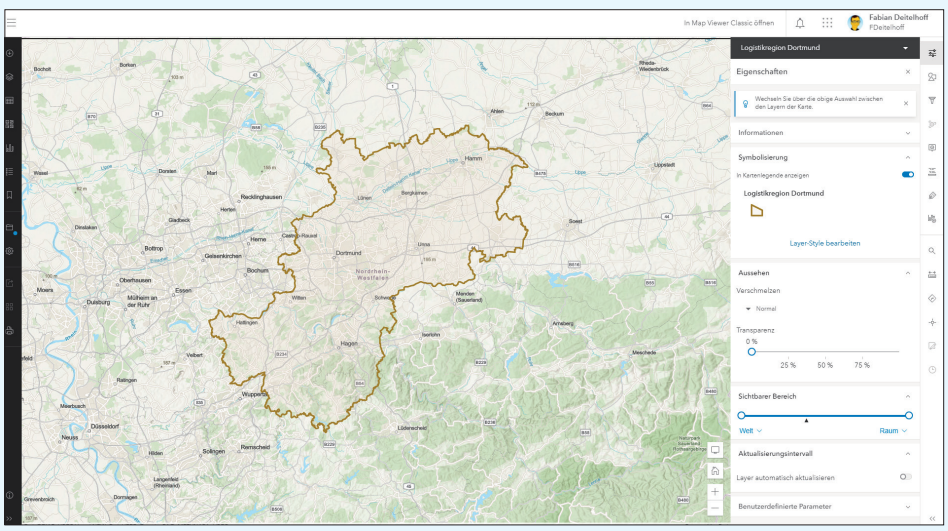
Wir Anwender sind daher verwöhnt, was den Umgang mit Kartenmaterial angeht. Eine verständliche Entwicklung, da diese Funktionen einen erheblichen Nutzen stiften, wenn sie richtig eingesetzt werden. Die Visualisierung einer Adresse kann verdeutlichen, was in der Nähe liegt, um bekannte Merkmale in Erinnerung zu rufen, sodass die neue Adresse plötzlich doch nicht so unbekannt ist wie zunächst gedacht. Und eine optimale Wegfindung ist in einer Welt, die sich immer schneller dreht, für viele zur Pflicht geworden.

Kartendaten, Kartenmaterial und alles, was dazugehört, in eine Anwendung einzubauen wird daher immer mehr zum Standard. Es muss aber nicht immer Google sein. Auch andere Anbieter stellen einen ganzen Fundus an Möglichkeiten bereit, um die eigenen Produkte kartentechnisch aufzuwerten. Esri mit seinen ArcGIS-Produkten ist ein solcher Anbieter, der in diesem Artikel näher vorgestellt wird. Aufgrund der Fülle an Möglichkeiten liegt der Fokus auf der .NET Runtime, eingebettet in eine Beispielanwendung. Andere Funktionen werden daher lediglich angerissen.

Was sind OOTB-Komponenten?

Um den Entwicklungsprozess zu beschleunigen und häufig genutzte Funktionalitäten zu bündeln, bietet Esri eine Reihe von OOTB-Komponenten an. Damit sind Out-of-the-Box-Komponenten gemeint, die direkt einsetzbar sind. Zu diesen gehören zum Beispiel der Map Viewer und der Vector Tile Style Editor.

Der Map Viewer ist eine interaktive Webanwendung, um Kartenmaterial für das Web und die eigene Anwendung zu erstellen. Das interaktive Tool erlaubt es beispielsweise, Karten zu erstellen, Layer auszuwählen und Daten zu visualisieren. Die Karten werden auf der ArcGIS-Plattform gespeichert und lassen sich von dort einbinden beziehungsweise abrufen. Der Map Viewer wurde vor Kurzem aktualisiert und befindet sich nicht mehr in der Betaphase. Der vorherige Map Viewer ist aber weiterhin – jetzt unter der Bezeichnung Map Viewer Classic – verfügbar. In den Einstellungen eines Accounts lässt sich zwischen beiden Versionen hin- und herschalten. In der Webanwendung stehen nicht nur Daten, zum Beispiel Layer, von ArcGIS Online zur Verfügung, sondern ebenso eigene und solche von der Community. Bild 5 zeigt das Beispiel einer Karte im Map Viewer mit eingezeichneter Logistikregion von Dortmund.



Eine Karte zur Logistikregion Dortmund im Map Viewer (Bild 5)

Was sind die Herausforderungen?

Über die Jahre ist die Anbindung von zum Beispiel Kartenmaterial immer einfacher geworden. Heutzutage muss die Antwort nicht immer direkt Google lauten, wenn Kartendaten zum Einsatz kommen sollen. Trotzdem sind die Anforderungen an die Anbieter dieser Materialien und Daten kontinuierlich gewachsen. Es reicht nicht mehr aus, nur eine Handvoll SDKs oder Schnittstellen anzubieten. In vielen Anwendungsszenarien müssen die Daten zum Beispiel in eine Webseite, eine Desktop-Anwendung und in mobile Apps integriert werden. Spiele sind ebenfalls ein wichtiger Faktor geworden, sodass die Kartendaten auch in Technologien wie Game Engines integrierbar sein müssen.

Das stellt viele Anforderungen sowohl an die Anbieter als auch an die Auswahl der korrekten Technologien. Nicht alles lässt sich über eine REST-Schnittstelle abbilden. Hin und wieder sind eigene SDKs sinnvoll oder notwendig. Den Funktionsumfang selbst zu implementieren ist schon bei sehr geringen Anforderungen praktisch nicht zu stemmen. Dafür ist die Komplexität zu hoch. Es gibt zwar kostenfreie Anbieter von Geodaten, zum Beispiel OpenStreetMap [1], die als Basis dienen können, dennoch ist es ein Mammutprojekt, Features wie die von Google Maps oder Esri selbst nachzubauen. In der Regel ist das nicht der eigene Fokus.

Eine mögliche Lösung, um der Komplexität und der Technologievielfalt zu begegnen, ist daher die Einbindung von vorhandenen Daten und Materialien über existierende APIs.

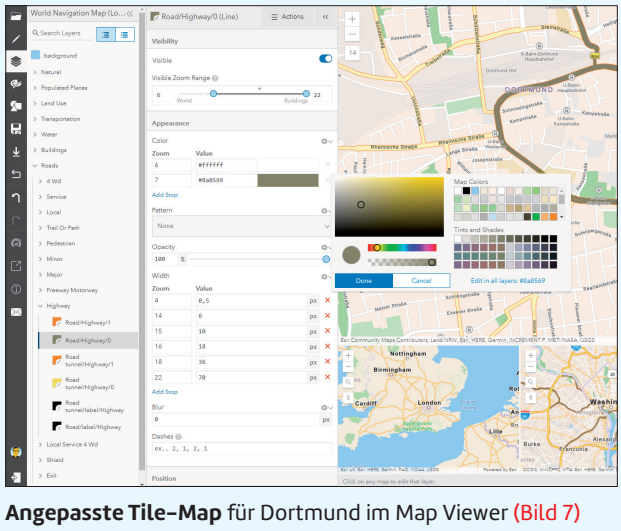
Was ist Esri?

Bevor sich der Artikel der ArcGIS .NET Runtime zuwendet, ein kurzer Überblick zu Esri [2]. Gegründet wurde das Unternehmen bereits im Jahr 1969 von Jack und Laura Dangermond. Der Hauptsitz des Unternehmens ESRI Inc. (Environmental Systems Research Institute) liegt in den USA. Das Motto von Esri ist „The Science of Where", was sehr gut passt, denn das Portfolio besteht aus Angeboten im Bereich Geoinformationssysteme (GIS). Unter dem Namen ArcGIS werden die Produktfamilien für alle Bereiche zusammengefasst. So umfasst zum Beispiel ArcGIS Desktop die Produkte für den Desktop, wie ArcGIS Pro. Im Server-Segment fallen unter die Produktfamilie ArcGIS Enterprise unter anderem zentrale Komponenten zur Datenhaltung und verschiedene Server für den Einsatz im eigenen Netzwerk. Mit ArcGIS Online respektive ArcGIS Platform werden die SaaS- und PaaS-Produkte aus der Cloud bezeichnet.

Neben dem Unternehmenssitz in den USA werden dort weitere zehn Handelsbüros und weltweit weitere 80 Handelsbüros unterhalten. Für die DACH-Region gibt es Esri Deutschland, Esri Österreich und Esri Schweiz. Mit den Angeboten und Dienstleistungen steht Esri unter anderem in Konkurrenz zu ▶

Verschiedene Darstellungsvarianten im Map Viewer (Bild 6)

Mit dem Vector Tile Style Editor lassen sich sogenannte Basemap-Styles, auch Basemap Layer Styles genannt, erzeugen. Darunter wird ein Satz an visuellen Eigenschaften, zum Beispiel die Füllfarbe, die Sichtebene und Schriftarten, zusammengefasst. Diese visuellen Elemente bestimmen, wie eine Kachel einer Karte angezeigt wird. Standard-Styles, die von Esri zur Verfügung gestellt werden, sind etwa die Straßenansicht oder die topografische Darstellung. Bild 6 zeigt ein Beispiel der zwei genannten Tile-Ansichten. Bild 7 zeigt dazu ein einfaches Beispiel mit angepassten Styles. Die Karte enthält einen Ausschnitt von Dortmund (Zentrum). Die angepasste Farbe verdeutlicht, dass die Hauptstraße verändert wurde. Auf diese Weise lassen sich komplett eigene Stile erzeugen, die dann wiederum in der eigenen Anwendung eingebunden werden. Dadurch entstehen sehr individuelle Kartenansichten, die den eigenen Anwendungsfall unterstützen oder zum Beispiel für das Branding der eigenen Marke genutzt werden können.



Angepasste Tile-Map für Dortmund im Map Viewer (Bild 7)

Hexagon, Precisely und Mapbox. Die Produkte von Esri bringen eine ganze Reihe von Features mit. Grob zusammengefasst sind das die folgenden:

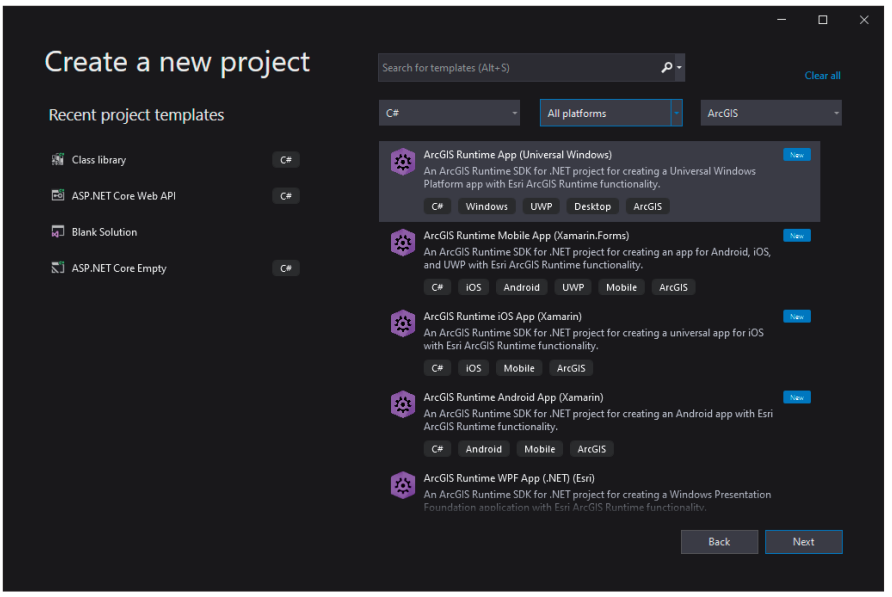
- **Basemap-Layer:** Es stehen Layer sowohl für 2D- als auch 3D-Kartenanwendungen zur Verfügung, darunter zum Beispiel Straßen, Satellitenbilder und topografische Daten. Diese lassen sich alle einzeln als Layer nutzen und beispielsweise mit einem eigenen Basemap-Layer anpassen.
- **Daten-Hosting:** Die ArcGIS-Plattform erlaubt das Speichern und Verwalten von Daten, zum Beispiel Kacheldaten zu Bildern beziehungsweise Vektoren. Diese Daten lassen sich in interaktiven Werkzeugen managen.
- **Daten-Visualisierung:** Aus den geografischen Daten können problemlos inhaltsstarke 2D- und 3D-Visualisierungen erstellt werden. Um Muster oder Beziehungen in den Daten zu visualisieren, steht ein Mapping-API zur Verfügung.
- **Geocoding und Suche:** Der Geocoding-Dienst ermöglicht die Suche nach Adressen, Unternehmen und Orten. Geografische Koordinaten lassen sich zum Beispiel zu Adressen umwandeln.
- **Routing:** Routen und Richtungen lassen sich auf Basis des Straßennetzes ermitteln. Diese Informationen stehen über den Routing-Service zur Verfügung. Auch Umkreissuchen sind darüber realisierbar.
- **Demografische Daten:** Der GeoEnrichment-Service erlaubt die Suche nach Informationen wie beispielsweise dem durchschnittlichen Einkommen, der Größe von Haushalten oder der Bevölkerungsdichte von Regionen.
- **Räumliche Analyse:** Die Analyse von räumlichen Zusammenhängen erlaubt Einblicke in versteckte Muster und Trends. Diese Features stehen als lokales API oder als serverseitige Plattform über den Spatial-Analysis-Service zur Verfügung.
- **Content Management:** Eine große Bandbreite von Inhalten, zum Beispiel Karten, Layer und Dateien, lassen sich über die ArcGIS-Plattform speichern, verwalten und über das API wieder abrufen. Auf diese Weise müssen die speziellen Daten und Formate nicht in der eigenen Anwendung vollständig verwaltet sein.
- **Offline-Funktionalitäten:** Dienste wie das Mapping, Geocoding oder die Berechnung von Routen funktionieren nativ auf Smartphones, Tables, Laptops oder Desktop-Systemen, und vor allem überall dort, wo die Netzwerkverbindung nicht sonderlich zuverlässig ist.

Diese umfassende Liste an Funktionen soll verdeutlichen, dass Esri mehr kann als Karten anzuzeigen und Orte zu suchen. Insbesondere die Abfrage von demografischen Daten und die umfassenden Möglichkeiten zur Datenspeicherung sowie die Offline-Funktionen stechen auf den ersten Blick nicht zwingend hervor. Im Folgenden legt der Artikel den Fokus auf die ArcGIS .NET Runtime, zeigt aber noch das weitere API-Angebot im zweiten Teil dieses Artikels.

Was ist die ArcGIS .NET Runtime?

Aus .NET-Sicht betrachtet ist die ArcGIS Runtime ein zentrales Kernstück für eine breite Palette an Features und Möglichkeiten. Über das SDK stehen beispielsweise Dienste zum Mapping und zur Analyse zur Verfügung. Unterstützung gibt es für die Plattformen Android, iOS und Windows.

Das API ermöglicht es, den Code zwischen den Plattformen zu tauschen, zum Beispiel über Xamarin Android, Xamarin iOS und Xamarin Forms. Ebenfalls unterstützt werden die Windows Presentation Foundation (WPF) und die Universal Windows Platform (UWP). Zu den Features zählen unter anderem:



Die Projektvorlagen zu ArcGIS in der Visual Studio Extension (Bild 1)

- **Karten:** Anzeigen, Messen von Entfernungen, Rotation und Grafiken wie Punkte, Linien und Polygone werden bei Karten unterstützt.
- **Szenen:** Die Anzeige von 3D-Daten, Analysen und die korrekte Höhe des Geländes sind möglich.
- **Layers und Daten:** Unterstützung für verschiedene Datenformate.
- **Dienste:** Basemap-Layer, Routing und Geocoding-Dienste stehen zur Verfügung.
- **Offline:** Zahlreiche Funktionen stehen auch offline zur Verfügung.
- **Queries:** Abfragen verschiedener Datenquellen in Queries und Suchen.
- **Routen und Richtung:** Berechnung von Routen, Entfernungen, Multi-Stop-Routen und eine Integration mit GPS.
- **Versorgungsnetze:** Diese Funktion erlaubt es, den Fluss von Ressourcen wie Gas, Wasser und Strom zu verfolgen und zu analysieren.

Diese Auflistung zeigt erneut, was Esri und ganz spezifisch die ArcGIS-Plattform alles unterstützt. Es hört also beim Anzeigen von Karten nicht auf, sondern fängt dort erst richtig an, spannend zu werden.

Beim Schreiben dieses Artikels ist Version 100.12.0 vom 25. August 2021 die aktuelle Version des ArcGIS Runtime API für .NET. An der Versionsnummer wird bereits deutlich, dass dieses SDK schon eine ganze Weile auf dem Markt verfügbar ist. Die detaillierten und dennoch übersichtlichen Release-Notes [3] verraten, was sich in den vergangenen Versionen so alles getan hat. Neben Fehlerbehebungen finden auch immer wieder neue Features den Weg in das SDK, und das übergreifend über alle Technologie-Plattformen. Neben der ArcGIS Runtime für .NET gibt es auch Versionen für Android, iOS, Java und Qt. Highlights der neuesten Version sind etwa das Feature Geotrigger, auch als Geofencing bekannt, und eine komplett runderneuerte API-Referenz [4] auf Grundlage von DocFX.

Bevor es um weitere Komponenten und konkrete Beispiele geht, ist die Lizenzierung anzusprechen, wenn das SDK mit der eigenen Anwendung ausgeliefert werden soll. Ein wichtiges Thema, da es einen Kostenfaktor darstellt. Vor der Veröffentlichung ist zu entscheiden, mit welchem Lizenzlevel die Laufzeitumgebung ausgeliefert werden soll. Zur Verfügung stehen die Abstufungen Lite, Basic, Standard und Advanced. Zudem existieren einige sogenannte Erweiterungs-Lizenzen, um zusätzliche Funktionalitäten hinzuzufügen. Dazu zählen zum Beispiel die Analyse von Daten, Informationen zu Versorgungsnetzen und eine Premium-Version von StreetMap, die hochaufgelöste Straßendaten für die Offline-Nutzung einschließt. Die vier Level bieten ebenfalls unterschiedlichste Möglichkeiten an. Sie sind aufeinander aufgebaut, sodass die Standard-Edition der Lizenz alles von Basic enthält und so weiter. Die Übersicht [5] der Lizenzen und deren Möglichkeiten erläutert sehr gut, worauf zu achten ist. Damit lässt sich der eigene Bedarf einschätzen und die Lizenz auswählen.

Die Demo-Anwendung

Für diesen Artikel wurde eine reale Anforderung in Form einer Demo-Anwendung [6] umgesetzt, um die Möglichkeiten von Esri zu zeigen. Diese Anforderung entstammt einer Anwendung zur Erfassung von Kunden, unter anderem mit Adressen, sowie Veranstaltungen, die ebenfalls Adressen besitzen. Diese Kunden, die im Moment primär aus Schulen aller Schulformen und weiteren, externen Bildungseinrichtungen bestehen, sollen auf einer Karte visualisiert werden. Dadurch lassen sich, so die Vermutung hinter der Anforderung, sehr gut Einzugsgebiete, räumliche Nähe und Routen visualisieren. Auch die Suche nach Adressen und deren gleichzeitige Visualisierung wären deutlich einfacher. Aus Sicht des Marketings ist es ebenfalls wichtig, diejenigen Schulen und Bildungseinrichtungen zu identifizieren, die noch keine Kunden sind, sich räumlich aber in der Nähe zu bereits existierenden Kunden befinden. Dadurch lassen sich im besten Fall Synergieeffekte nutzen. In zukünftigen Versionen ist dann das Routing von Strecken mit einem und mehreren Zielpunkten beziehungsweise Zwischenstopps geplant.

Umgesetzt werden soll also die Anzeige einer zuvor vorbereiteten Karte, um auf dieser Standorte von Kunden anzuzeigen. Die Demo-Anwendung ist mit WPF erstellt und liegt als Rider-Projekt vor. Die Basis bilden das .NET Framework in Version 4.7.2 und das NuGet-Paket *Esri.ArcGISRuntime.WPF*. Bei der Installation werden die Pakete *Esri.ArcGISRuntime* und *Esri.ArcGISRuntime.runtimes.win* direkt mitinstalliert. Die Daten der Kunden kommen aus einer REST-Schnittstelle des angesprochenen Verwaltungssystems, das sich gerade im Aufbau befindet. Diese Anwendung ist mit ASP.NET Core Web API in Version 5 erstellt.

Wer statt JetBrains Rider lieber Visual Studio einsetzen möchte, kann den Beispielen trotzdem folgen. Für Visual Studio existiert zusätzlich eine Visual Studio Extension (VSIX-Datei) mit Projektvorlagen und NuGet-Paketen für unterstützte Plattformen des ArcGIS Runtime .NET API. Durch die Installation der Erweiterung werden die Visual-Studio-Vorlagen hinzugefügt und eine lokale NuGet-Paketquelle konfiguriert. Die Pakete müssen daher nicht mehr heruntergeladen werden. Bild 1 zeigt dazu die Übersicht der neuen Projektvorlagen, um komfortabel loslegen zu können.

Vorbereitungen und Projektstruktur

Zur Vorbereitung eines Projekts müssen Fragen zur Sicherheit und Authentifizierung, Logins, API-Keys und der Lizenz geklärt sein.

Das Lizenzthema ist für die Demo-Anwendung recht einfach erklärt. Da die Anwendung momentan nicht ausgeliefert wird, ist die Auswahl einer der oben genannten konkreten Lizenzen nicht notwendig. Bei der Entwicklung lassen sich alle Funktionen problemlos nutzen.

Wichtig ist ein ArcGIS-Developer-Account [7], um nach dem Login das ArcGIS Runtime SDK herunterzuladen. Der Entwickler-Account ist nach einer einfachen Registrierung verfügbar. Für Entwicklungs- und Testzwecke bietet das SDK anschließend Zugriff auf alle Funktionen des API. Wichtig zu wissen ist, dass sich die eigene Anwendung im Entwickler-Modus etwas anders verhält. Das Wasserzeichen „Licensed For Developer Use Only" wird zum Beispiel auf Karten eingefügt und erscheint auf der Konsole als Ausgabe. Ein lokaler Server gibt die Meldung „This application is licensed for developer use only" auf der Konsole aus. ▶

Listing 1: ArcGISRuntimeEnvironment in C# aufsetzen

```
protected override void OnStartup(
    StartupEventArgs e)
{
  base.OnStartup(e);
  Esri.ArcGISRuntime.ArcGISRuntimeEnvironment
    .ApiKey
  = "AAPK565a27264a19496bb396bb...7zj8CNjPRfOSv6j2
    G5YfB";
}
```

Listing 2: Einbinden der MapView im XAML-Code

```
<Window.Resources>
  <models:CustomerMapViewModel
    x:Key="CustomerMapViewModel" />
</Window.Resources>
<Grid>
  <esri:MapView x:Name="MainMapView"
                Map="{Binding Map,
                Source={StaticResource
                CustomerMapViewModel}}" />
</Grid>
```

Für die Demo-Anwendung genügt zudem ein einfacher API-Key, um fertige Dienste zu nutzen. Diese Ready-to-use-Dienste sind eine Gruppe von ArcGIS-Platform-Diensten, die Kernfunktionalitäten zum Beispiel für die Erstellung von Kartierungsanwendungen bereitstellen, wie unter anderem Basemap-Layer, die Suche oder das Routing. Ein API-Key ist im Developer Dashboard nach dem Login zu erstellen, was problemlos funktioniert.

Authentifizierung bei der ArcGIS-Plattform

Für die Authentifizierung stehen verschiedene Möglichkeiten zur Verfügung. Es gibt den klassischen API-Schlüssel, der als permanentes Token dient und der Anwendung einen direkten Zugriff auf bestimmte Dienste bietet. Dann existiert noch die sogenannte ArcGIS-Identität, früher *Named User* genannt. Das ist ein kurzlebiges Token, erzeugt mittels OAuth 2.0, um der Anwendung den Zugriff auf Inhalte und Dienste zu geben, die mit einem existierenden ArcGIS-Benutzeraccount verknüpft sind. Die dritte Möglichkeit bilden die Application Credentials: ebenfalls ein kurzlebiges Token, erzeugt mittels OAuth 2.0, um der Anwendung Zugriff auf bestimmte Dienste zu geben. Welche Methode zum Einsatz kommt, hängt vom Anwendungsfall ab. Reichen die Ready-to-use-Dienste aus, ist ein API-Schlüssel völlig in Ordnung. Möchten Benutzer über die Anwendung eigene, private Daten ändern, ist eine ArcGIS-Identität notwendig.

Die Authentifizierung am Esri-System, in diesem Fall der ArcGIS-Plattform, ist für die Demo-Anwendung unkompliziert, da ein API-Key zum Einsatz kommt und keine ArcGIS-Identity oder Application Credentials, wie oben beschrieben. In der WPF-Anwendung wird dazu die *OnStartup*-Methode der App-Klasse genutzt, um dort den Schlüssel anzugeben und an die *ArcGISRuntimeEnvironment* weiterzureichen. Listing 1 zeigt den Code dazu. Damit ist die Authentifizierung abgeschlossen und die WPF-Anwendung bereit.

Statt eines globalen API-Keys lassen sich aber auch einzelne Schlüssel für bestimmte Bereiche beziehungsweise Komponenten setzen. Das funktioniert bei allen Klassen in der ArcGIS Runtime mit der implementierten Schnittstelle *IApiKeyResource*. Das erlaubt es zum Beispiel, deutlich feinere Nutzungsstatistiken über die Telemetrie zu bekommen und die Ressourcen, die eine Anwendung nutzt, besser zu steuern. Klassen, die diese Schnittstellen implementieren, sind beispielsweise *Basemap*, *LocatorTask* und *RouteTask*.

Kartenmaterial einbinden

Mittlerweile läuft die Demo-Anwendung und verbindet sich über den API-Schlüssel mit der ArcGIS-Plattform. Daher ist es Zeit für den nächsten Schritt: Wir brauchen eine Karte. Da auf ihr Informationen angezeigt werden sollen, zum Beispiel Standorte von Kunden, muss eine solche Karte zunächst in die Anwendung eingebunden und angezeigt werden.

Da die Demo-Anwendung mit WPF implementiert ist, wird das Entwurfsmuster Model-View-ViewModel (MVVM) umgesetzt, sodass zunächst ein Modell für die Karte erstellt wird. Das *CustomerMapViewModel* enthält einen Verweis auf die Map-Instanz und initialisiert diese Karte mit dem Basemap-Style *ArcGISStreets*, sodass Straßen zu sehen sind. Für die Visualisierung von Kunden und Routen, um zwei Themenbereiche aus den Anforderungen zu nennen, ist beispielsweise keine topologische Karte notwendig. Das Modell implementiert das Interface *INotifyPropertyChanged* und bekommt die *OnPropertyChanged*-Methode sowie eine öffentliche Eigenschaft für das *Map*-Objekt. Der Konstruktor initialisiert diese *Map*-Instanz, was wie folgt vonstatten geht:

```
private void SetupMap()
{
  Map = new Map(BasemapStyle.ArcGISStreets);
}
```

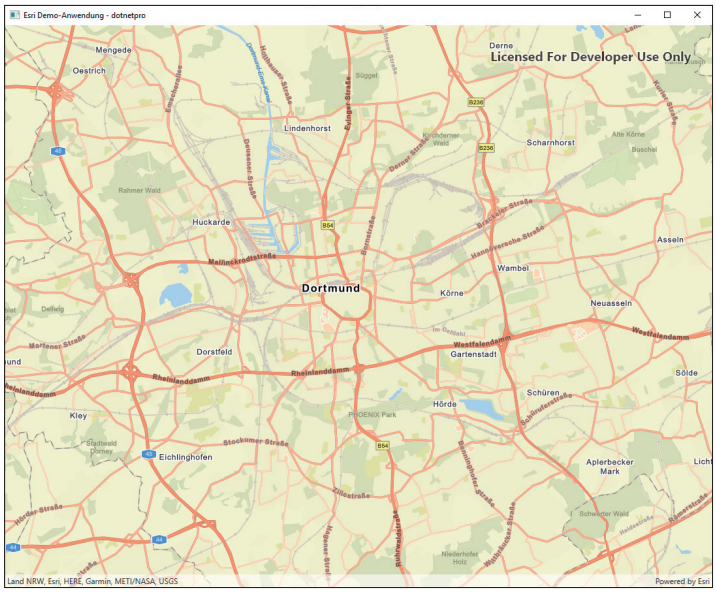
Danach ist das Modell bereits vollständig implementiert. Da die Karte im Hauptfenster der Anwendung angezeigt werden soll – für diese Demo ist das ausreichend –, ist noch das *Main-*

Listing 3: LocatorTask zum Auffinden einer Adresse als Zeichenkette

```
private async Task<GeocodeResult> LocateAddress(
    string address)
{
  var locatorTask = new LocatorTask(new Uri(
    "https://geocode-api.arcgis.com/arcgis/rest/
    services/World/GeocodeServer"));

  var results = await locatorTask.GeocodeAsync(
    address);

  return results?.FirstOrDefault();
}
```

Window anzupassen. Der XAML-Code dazu ist in Listing 2 zu sehen. Das *CustomerMapViewModel* wird als Ressource eingebunden und die *MapView* von Esri – hier wird die Karte dann tatsächlich visualisiert – im Grid eingebunden. Die Karte ist an das Modell gebunden und besitzt einen Namen, um auf die Instanz zugreifen zu können. Jetzt fehlt nur noch, einen zentralen Punkt zu setzen, für den die Karte initial angezeigt wird, beziehungsweise wohin sie automatisch zoomt. Das erledigt das folgende Code-Snippet, eingebettet im Konstruktor des Hauptfensters:

```
var mapCenterPoint
  = CoordinateFormatter.FromLatitudeLongitude(
    "51°30'51.2784''N 7°28'6.3444''E",
    SpatialReferences.Wgs84);

MainMapView.SetViewpoint(new Viewpoint(
  mapCenterPoint, 100000));
```



Die Karte mit fokussierter Region für Dortmund in der WPF-Demo-Anwendung (Bild 2)

Zunächst wird eine Koordinate erstellt, die aus der Angabe von Latitude und Longitude berechnet wird. Diese gibt das Zentrum von Dortmund an. Anschließend wird der erzeugte MapPoint genutzt, um den Viewpoint der Karte zu verändern. Das funktioniert problemlos, wie in Bild 2 zu sehen ist.

Die Demo-Anwendung hat jetzt eine Karte mit einem Standard-Basemap-Style von Esri, um Straßen anzuzeigen. Das WPF-Control zur Karte unterstützt die Standardoperationen, die Nutzer von einer solchen Karte erwarten. Ein Doppelklick zoomt an die entsprechende Stelle, das Mausrad verändert ebenfalls die Zoomstufe, und der angezeigte Ausschnitt kann durch Ziehen mit der linken Maustaste in alle Richtungen verändert werden. Das macht die Interaktion mit dem Kartenmaterial sehr komfortabel und einfach einzubinden.

Adresssuche und Darstellung durch Geocoding

Jetzt, wo die Demo-Anwendung eine gewünschte Karte anzeigen kann, ist der nächste Schritt, die Information in Form von Standorten mit einzubringen. Erst dann ist die Karte für die gewünschten Anwendungsfälle einsatzfähig und nützlich. Dazu ist eine Suche notwendig. Als Eingabe dient eine Adresse, denn die ist bei den Datensätzen wie Kunden, Veranstaltungen und Teilnehmenden dieser Veranstaltungen bekannt – besser als Koordinaten oder Ähnliches, die ebenfalls auf Basis einer Adresse zunächst umgewandelt werden müssten. Der Prozess, einen Standort zu einer Adresse zu matchen, wird Geocoding genannt. Dabei kommt ein sogenannter Locator zum Einsatz, der entweder lokal oder online zur Verfügung steht. Damit ist ein ArcGIS-Datensatz gemeint, der Adressinformationen und die Regeln für die Übersetzung von Ortsbeschreibungen – etwa Straßenadressen oder Ortsnamen – in räumlichen Daten speichert. Diese Daten lassen sich anschließend auf einer Karte anzeigen. Bei dem vorherigen Beispiel erfolgte das Geocoding noch manuell, indem der Standort von Dortmund in Form von Latitude und Longitude bei Google gesucht wurde. Automatisiert ist das verständlicherweise einfacher, zuverlässiger und bereitet weniger Aufwand. Das Reverse Geocoding funktioniert zudem ebenfalls. Wer einen Punkt kennt, kann dazu die Adresse oder einen Point of Interest (POI) suchen.

Für die Demo-Anwendung ist die Anzeige verschiedener Adressen und damit Punkte (Koordinaten) auf einer Karte notwendig. Im ersten Schritt wird eine Adresse gesucht und der Viewpoint der Karte dorthin verschoben. Das ist ein Test, ob die Adresssuche und die Anzeige des Resultats funktionieren. Listing 3 zeigt dazu die asynchrone *LocateAddress*-Methode. Die verwendete Klasse *LocatorTask* bekommt entweder einen URL für einen Online-Locator oder einen Pfad für eine lokale Version übergeben. In diesem Fall kommt das Angebot der ArcGIS-Plattform zum Einsatz. Der *GeocodeAsync*-Aufruf sucht zu einer Adresse im Textformat die Koordinaten und weiteren Daten heraus, wenn diese denn existieren. Testweise wird eine Suche über einen Button-Klick ausgelöst. ▶

Listing 4: Anzeige einer Adresse mit vorheriger Suche

```
private async void DisplayAddresses_OnClick(
    object sender, RoutedEventArgs e)
{
  var result = await LocateAddress(
    "Emil-Figge-Str. 80, 44227 Dortmund");

  if (result?.DisplayLocation != null)
  {
    await MainMapView.SetViewpointAsync(new
      Viewpoint(result.DisplayLocation, scale: 5000));
  }
}
```

Die Handler-Methode zeigt Listing 4. Die *LocateAddress*-Methode wird mit einer Adresse der TU Dortmund aufgerufen, und wenn es ein Ergebnis gibt, wird die *DisplayLocation* zur Anzeige genutzt, mit einer für Demo-Zwecke vernünftigen Zoom-Stufe. Zusätzlich existiert noch die *RouteLocation*, die für die Wegfindung genutzt wird. Die *DisplayLocation* ist genauer für die Anzeige auf einer Karte, zum Beispiel das Dach eines Hauses, während die *RouteLocation* die nächstgelegene Straße zurückgeben würde. Bild 3 zeigt das Ergebnis des Button-Klicks. Der asynchrone Aufruf von *SetViewpointAsync* nutzt eine nette Animation, damit die Position nicht plötzlich auf der Karte erscheint.

Die Suche nach einer Adresse lässt sich zudem mit Parametern anpassen. Diese Geocode-Parameter bestimmen zum Beispiel, wo gesucht werden soll, wie viele Resultate gewünscht sind und wie die Sprache der Ausgabe eingestellt sein soll. Um die Ausgabe auf Deutsch einzustellen, was in diesem Fall nur für Debug-Ausgaben eine Auswirkung hat, die Suche aber ebenfalls auf Deutschland zu beschränken, reicht folgendes Snippet aus:

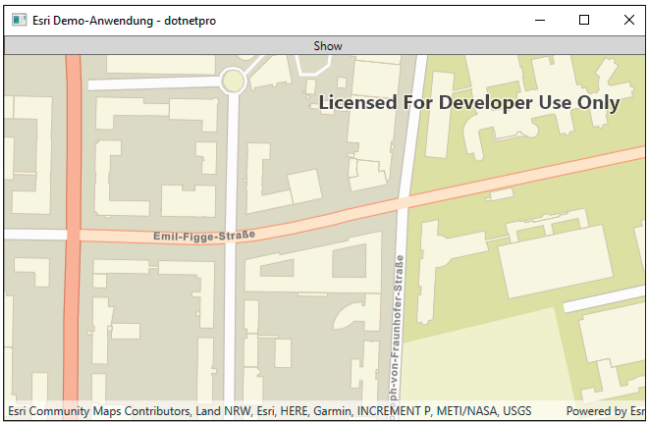
```
var geocodeParams = new GeocodeParameters
{
  CountryCode = "Germany",
  OutputLanguage = new System.Globalization.CultureInfo(
    "de-DE")
};
```

Eingebunden werden die Parameter in der *GeocodeAsync*-Methode. Insbesondere die Einschränkung der Suchresultate und des Suchortes wirken sich unter Umständen auf die Laufzeit der Suchanfrage aus.

Standorte visualisieren

Adressen suchen ist mit dem Beispiel aus dem vorherigen Abschnitt realisiert. Woher die Adressen kommen, spielt im aktuellen Fall noch keine Rolle. Eine externe Datenquelle ist aber problemlos denkbar. Diese Adressen müssen jetzt aber noch auf der Karte visualisiert sein. Ein Wechseln des Viewpoints zur Position hin ergibt nur dann Sinn, wenn ein Benutzer zum Beispiel einen entsprechenden Mausklick ausführt. Automatisiert alle Standorte nacheinander anzuzeigen und dort hinzuzoomen ist kein sinnvoller Anwendungsfall.

Um diese Art von Visualisierungen anzuzeigen, ist zur Karte ein Overlay hinzuzufügen. Dieses Graphics Overlay, ein



Eine Position, die über die Geocoding-Funktion gesucht und gefunden wurde (Bild 3)

Container für Grafiken wie Punkte, Linien und Polygone, zeigt visuelle Informationen temporär auf dem Client an. Es ist also keine Anreicherung der Karte mit Daten, die in der ArcGIS-Plattform dauerhaft gespeichert sind.

Um diese Funktionalität zu integrieren, sind zunächst Anpassungen am *CustomerMapViewModel* notwendig. Das Modell erhält zunächst eine neue Eigenschaft *GraphicsOverlays* vom Typ *GraphicsOverlayCollection*. Das zeigt, dass mehrere Overlays zu einer Karte hinzugefügt werden können; in diesem Fall ist allerdings nur eines notwendig. Die dazu implementierte Methode *CreateGraphics* ist in Listing 5 zu sehen. Dieses Overlay wird zunächst im Modell gespeichert. Konsumiert wird das Overlay als Binding im XAML-Code. Das Einbinden der Esri MapView ändert sich im *MainWindow* daher wie folgt:

```
<Esri:MapView x:Name="MainMapView"
              Map="{Binding Map, Source={StaticResource
                CustomerMapViewModel}}"
              GraphicsOverlays="{Binding GraphicsOverlays,
                Source={StaticResource
                CustomerMapViewModel}}" />
```

Im nächsten Schritt sollen Positionen an das ViewModel übergeben werden, um Koordinaten als Punkte von Standorten zu markieren. Dazu sind einige Anpassungen notwendig, da aktuell das *MainWindow* das *CustomerMapViewModel* nur als Ressource eingebunden hat. Von jetzt an wird

Listing 5: Erzeugen eines GraphicsOverlay zum Anzeigen von Markierungen auf einer Karte

```
private void CreateGraphics()
{
  var graphicsOverlay = new GraphicsOverlay();

  var overlays = new GraphicsOverlayCollection
  {
    graphicsOverlay
  };

  GraphicsOverlays = overlays;
}
```

Listing 6: Visualisierung eines Standorts auf der Karte

```
public void AddLocationPoint([CanBeNull] MapPoint
displayLocation)
{
  var markerSymbol = new SimpleMarkerSymbol
  {
    Style = SimpleMarkerSymbolStyle.Circle,
    Color = Color.LightBlue,
    Size = 25.0,

    Outline = new SimpleLineSymbol
    {
      Style = SimpleLineSymbolStyle.Solid,
      Color = Color.DarkGray,
      Width = 2.0
    }
  };

  var pointGraphic = new Graphic(displayLocation,
    markerSymbol);

  locationsGraphicsOverlay.Graphics.Add(
    pointGraphic);
}
```

dieses Modell in der Code-behind-Klasse instanziert und als *DataContext* an das WPF-Fenster gebunden:

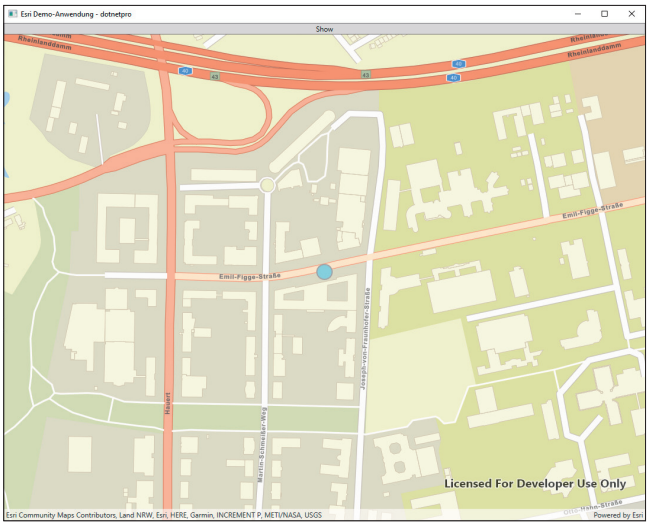
```
CustomerMapViewModel = new CustomerMapViewModel();
DataContext = _CustomerMapViewModel;
```

Daraufhin lässt sich die neue Methode verwenden, um einen Punkt für einen Standort hinzuzufügen:

```
CustomerMapViewModel.AddLocationPoint(
  result?.DisplayLocation);
```

Der Code, der diese Funktion implementiert, ist ein klein wenig umfangreicher und in Listing 6 abgebildet. Komplex ist der Code dennoch nicht. Zunächst ist ein Symbol für die Markierung notwendig. In diesem Fall reicht die Klasse *SimpleMarkerSymbol* aus. Dort lassen sich der Stil, die Farbe und die Größe festlegen. Dann noch eine Outline als Umrandung hinzufügen, und fertig ist das Symbol. Das wird zusammen mit der Position zu einem Punkt zusammengesetzt, sodass diese Grafik auf der Karte angezeigt werden kann. Bild 4 zeigt das Resultat in der laufenden Demo-Anwendung mit einer etwas größeres Zoom-Stufe. Ändert sich die Zoom-Stufe, so bleibt die Markierung stabil, was den Größenmaßstab betrifft. Herangezoomt ist der Punkt also nicht übergroß, sondern er verkleinert sich. Umgekehrt gilt dasselbe.

Auf diese Weise lassen sich auch andere Grafiken wie Linien und Polygone hinzufügen. Das Prinzip ist immer identisch, nur die Art und Weise, wie die Grafik zusammengebaut wird, verändert sich. Darüber hinaus stehen weitere Möglichkeiten zur Verfügung, diese Grafiken anzupassen, zum Beispiel über Tooltips.



Die Positionsmarkierung zur gefundenen Adresse (Bild 4)

Ausblick

Der aktuelle Stand der Demo-Anwendung ist, dass Standorte über eine Adresssuche angezeigt und visualisiert werden können. Wie sich markante Elemente auf einer Karte besonders hervorheben lassen – beispielsweise Bildungseinrichtungen und deren zugehörige Parkplätze – und wie sich Routen mithilfe von Feature Layern über die bestehende Ansicht legen lassen, wird der zweite Teil dieses Artikels zeigen. ■

[1] Website zum OpenStreetMap-Projekt, www.openstreetmap.de
[2] Website zu Esri, www.esri.de/de-de/home
[3] Release-Notes zur Version 100.12.0 der ArcGIS Runtime, www.dotnetpro.de/SL2202ArcGIS1
[4] Runderneuerte API-Referenz zur ArcGIS Runtime, www.dotnetpro.de/SL2202ArcGIS2
[5] Übersicht über die Möglichkeiten der verschiedenen Lizenzen, www.dotnetpro.de/SL2202ArcGIS3
[6] Beispielcode zur WPF-Demo-Anwendung auf GitHub, www.dotnetpro.de/SL2202ArcGIS4
[7] Seite zum Registrieren eines ArcGIS-Developer-Accounts, www.dotnetpro.de/SL2202ArcGIS5

Dr. Fabian Deitelhoff
arbeitet nach seiner Promotion als Innovation- und Transfermanager am Centrum für Entrepreneurship & Transfer an der TU Dortmund. Auch ist er als Autor, Dozent und Softwareentwickler im .NET- und Web-Umfeld tätig.
@FDeitelhoff

dnpCode A2202ArcGIS