Offlinefähigkeiten für Windows 8 Apps

Lädst du noch oder nutzt du schon?

(Alternative: Ich bin dann mal off!)

Dieser Artikel gibt einen Einblick, was bei der Entwicklung einer offlinefähigen Windows-8-App zu beachten ist und welche Möglichkeiten WinRT bietet. Ein Architekturbeispiel zeigt, wie auch die Performance einer App von Offlinedaten profitieren kann.

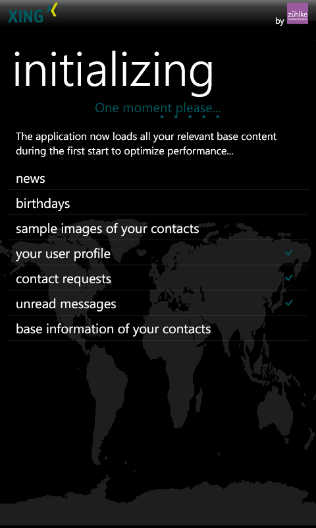
# Einleitung

Super! Die Idee für eine neue tolle Windows-8-App ist gefunden und somit ein wichtiger Schritt getan. Bevor man sich nun Hals über Kopf in die Entwicklung stürzt, lohnt es sich darüber nachzudenken, wodurch eine App für den Benutzer zur „Lieblings-App“ wird. Natürlich sollte sie ihren Zweck erfüllen, also genau die benötigen Anwendungsfälle abdecken. Aber das reicht noch lange nicht. Damit die App wirklich Spaß macht, muss sie schnell starten, sofort reagieren, nie abstürzen, intuitiv bedienbar sein, gut aussehen und immer funktionieren. Wir App-Benutzer sind sehr anspruchsvoll. Selbst Apps der großen Social-Media-Plattformen können diese Erwartungen nicht immer erfüllen und werden mit schlechten Bewertungen bestraft. Vielleicht kennen Sie das ja auch: Sie sitzen gerade im Zug und wollen sich die Neuigkeiten aus Ihrem Bekanntenkreis anschauen, müssen aber „gefühlte Stunden“ warten, bis überhaupt etwas auf dem Bildschirm angezeigt wird – wenn nicht gerade noch ein Tunnel dazwischen kommt. Große Anbieter können sich so etwas vielleicht leisten. Wenn man aber gerade erst in den Markt einsteigen will, muss der erste Wurf sitzen. Also verabschieden wir uns von der idealen Welt, in der wir immer online sind und Webdienste nie ausfallen oder überlastet sind und kümmern uns um die Offlinefähigkeit unserer Anwendung.

# Methodisches Vorgehen

Wie bei jeder guten Anwendung steht am Anfang der Entwicklung das Requirements- und Usability-Engineering. Auch im Bereich der Offlinefähigkeit sind diese Disziplinen unabdingbar. Sie sollten genau überlegen, welche Anwendungsfälle offline für Ihre App überhaupt Sinn haben und was im Offline-Modus angezeigt werden soll. Können Sie entscheiden, welche Informationen offline relevant sind oder sollte die Entscheidung dem Anwender überlassen werden? Eine App, die zum Beispiel eine Suche auf StackOverflow durchführt, kann offline nicht den vollen Funktionsumfang bieten. Es wäre aber vorstellbar, schon früher gefundene und markierte Diskussionen offline zu speichern und diese dem Nutzer zu präsentieren. Wahrscheinlich wären für diesen Anwendungsfall auch unterschiedliche Views für den Online- und Offline-Modus notwendig, so dass auch geklärt werden muss, wann man zwischen den Modi hin- und herwechselt. Kann das automatisch geschehen? Im Zweifelsfall sollten Sie dem Anwender die Kontrolle überlassen – den ärgert es nämlich, wenn seine App ständig automatisch ihre Views umschaltet. Sollte Ihre App für beide Modi dieselben Views verwenden, ist es dennoch wichtig über den Status der Internetverbindung zu informieren. Der Nutzer will natürlich wissen, ob er gerade über aktuelle oder ältere Daten verfügt.

Bei der Entwicklung der Xing-App fürs Windows Phone 7 sind auch wir durch einen Lernprozess gegangen. Der erste Entwurf sah vor, dass viele Entitäten, wie Kontakte, Nachrichten oder Neuigkeiten aus dem sozialen Netzwerk zwar offline gespeichert werden, aber nur für schon betrachtete Einträge. Bei internen Tests zeigte sich schnell, dass es für die Anwender vollkommen intransparent und frustrierend war, wenn einzelne Kontakte offline zur Verfügung standen, andere aber nicht. Die Lösung war möglichst alle Entitäten offline zu speichern. Nur zu welchem Zeitpunkt muss dies geschehen? Sollten alle Kontakte geladen und gespeichert werden, wenn man das erste Mal die Kontaktübersicht öffnet? Das Ergebnis wäre eine recht lange Wartezeit, was bei dem Anwender wiederum für Unmut sorgt. Am ehesten wird die Wartezeit beim ersten Start der Anwendung akzeptiert. Jedoch auch zu diesem Zeitpunkt muss der Anwender informiert werden, was die App gerade macht. In der Xing-App gibt es dafür einen Lade-Bildschirm, welcher genau darüber informiert, was gerade geladen wird und wie weit der Prozess schon fortgeschritten ist. (Abbildung 1)



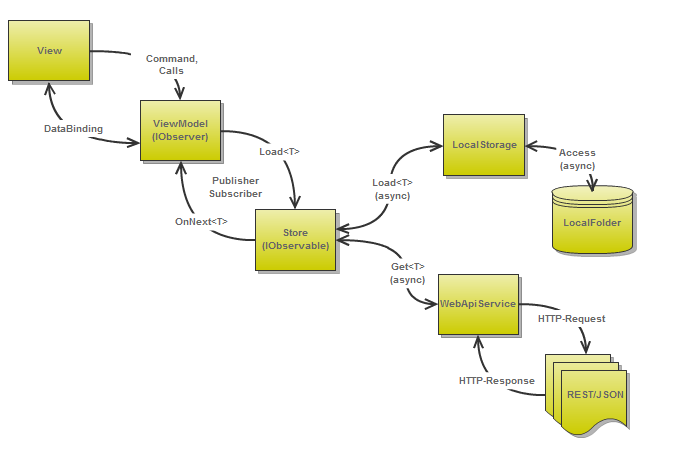
Aus dieser Vorgehensweise ergibt sich ein weiterer Vorteil: Beim zweiten Start der App sind die meisten Daten schon lokal abgelegt und können sofort angezeigt werden. Gleichzeitig sucht die Anwendung im Hintergrund online nach den neuesten Daten und präsentiert sie nachträglich dem Anwender.

Zusammengefasst sollte man sich bei der Entwicklung einer „Offline-First“-App folgende Fragen stellen:

* Welche Inhalte sollen offline gespeichert werden?-
* Erfolgt die Speicherung automatisch oder durch den Benutzer?
* Muss es unterschiedliche Views für den Online/Offline-Fall geben?
* Wie wird der Online-Status angezeigt?
* Findet der Wechsel zwischen Online- und Offline-Modus automatisch oder explizit durch den Benutzer statt?
* Zu welchem Zeitpunkt sollen die Daten für den Offline-Fall gespeichert werden?

# Architekturansatz für das Offline-First-Prinzip

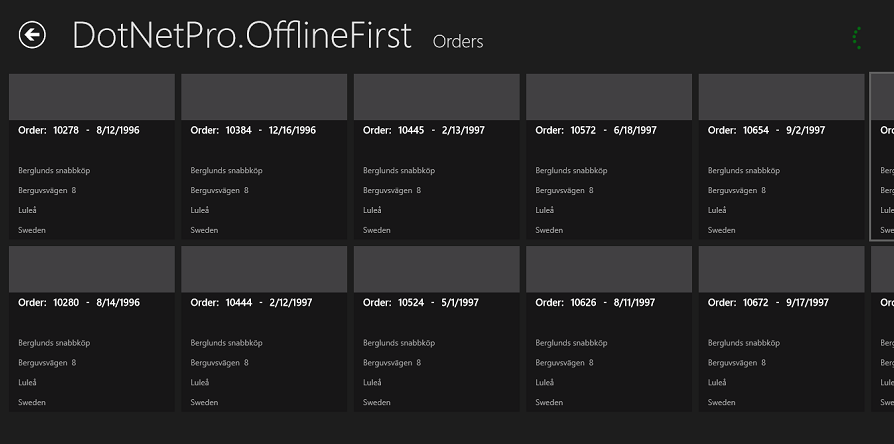
Im folgenden Abschnitt soll ein möglicher Architekturansatz für eine Offline-First Strategie aufgezeigt werden. Die zu dem Artikel mitgelieferte Beispiel-App zeigt Kunden und deren Bestellungen an. Als Datenbasis dient die gute alte Northwind-Datenbank, welche über einen Webservice angebunden wird. Die dabei verwendeten Klassen werden in Abbildung 2 gezeigt.



Die Beispielarchitektur ist nach dem für WPF-Anwendungen allgemein empfohlenen MVVM-Pattern umgesetzt. Die Store-Klasse übernimmt hierbei die Rolle des Models. Die ViewModel-Klasse selbst registriert sich als Observer über das von Microsoft in .NET 4 eingeführte IObservable-Interface für alle neuen Daten, die vom Store gelesen werden. Dadurch ist es für das ViewModel irrelevant, ob die Daten offline oder online geladen werden. Das ViewModel bekommt alle Daten durch den Aufruf von IObserver.MoveNext geliefert.

Die Klasse Store selbst fragt beim ersten Zugriff zuerst die möglicherweise schon offline gespeicherten Daten ab und liefert diese per Aufruf von OnNext erst einmal aus. Im zweiten Schritt erfolgt der Zugriff auf den WebApi-Service. Dabei wird für Customers und Orders ein jeweils eigener Store verwendet. Der ApplicationStore verwaltet den Zugriff auf die einzelnen Stores und kümmert sich unter anderem um die Speicherung und das Bereinigen von Offlinedaten.

Der Benutzer sieht dadurch die zuletzt gelesenen Daten relativ schnell und muss nicht warten, bis diese über den langsamen Webservice geladen werden. Wie in Abbildung 3 zu sehen ist, werden die offline geladenen Daten schon angezeigt. Dem Benutzer wird aber durch das ProgressRing-Steuerelement des WinRT-Frameworks suggeriert, dass Daten geladen werden.



# Anwendungsdaten lokal speichern

Es gilt nun zu überlegen, wann Daten lokal gespeichert werden sollten. Die Entscheidung kann je nach Anwendungsfall anders ausfallen. Es ist möglich, die Daten sofort nach dem Empfang lokal zu persistieren oder aber erst dann lokal zu speichern, wenn die Anwendung in den Ruhezustand geschickt wird. Windows-8-Apps unterliegen einem etwas anderen Lebenszyklus-Modell als man es von herkömmlichen Anwendungsprogrammen unter Windows gewohnt ist: Sobald der Benutzer zwischen zwei Windows-8-Apps wechselt, wird die vorher aktive App in den Suspended-Modus geschickt. Dabei bleiben zwar alle Daten im Speicher, aber die App bekommt diesen Vorgang durch das Abonnieren des Application.Suspending-Ereignisses mit und kann entsprechend darauf reagieren. In Listing 1 wird gezeigt, wie dies in der Beispiel-App umgesetzt ist.

private async void OnSuspending(object sender, SuspendingEventArgs e)

{

var deferral = e.SuspendingOperation.GetDeferral();

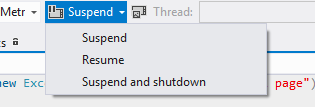
await Container.Resolve<ApplicationStore>().SaveAsync();

deferral.Complete();

}

In den Projektvorlagen, die Microsoft für die Erstellung von Windows-8-Apps mitliefert, ist dieser Eventhandler schon enthalten. Hinzugekommen ist nur die Zeile, in der die ApplicationStore.SaveAsync-Methode aufgerufen wird. In diesem Zusammenhang ist der Einsatz des Deferral Objektes wichtig. Da die SaveAsync-Methode asynchron ausgeführt wird, wartet die Laufzeitumgebung nicht, bis die Methode wirklich abgeschlossen ist. Das Programm könnte schon beendet werden, bevor der Speichervorgang durchgeführt wurde. Um das zu verhindern, fordert man über die SuspendingEventArgs ein SupendingDeferral-Objekt an, das man erst nach Ausführung der SaveAsync-Methode auf Complete setzt. Der WinRT-Runtime ist dadurch bekannt, dass hier erst asynchrone Methoden beendet werden müssen, bis die App selbst in den Suspended-Modus wechseln kann. Dieser Vorgang darf laut [1] allerdings maximal 5 Sekunden in Anspruch nehmen. Unter [2] ist in der MSDNDokumentation genauer erklärt, was bei Suspend und dem entsprechenden Gegenstück Resume zu beachten ist.

Um das Suspend-Ereignis und die entsprechende Ereignisbehandlung selbst in der Entwicklungsumgebung testen zu können, reicht es nicht, im Simulator von der zu testenden App zu einer anderen zu wechseln. Visual Studio stellt hierfür über die Toolbar Debug Location die in Abbildung 4 angezeigten Befehle zur Verfügung, um die Ereignisse explizit auszulösen.



Bevor gezeigt wird, wie die Beispiel-App ihre Daten offline speichert, sollten ein paar grundlegende Aspekte über die Möglichkeiten der lokalen Datenspeicherung in Windows-8-Apps erläutert werden.

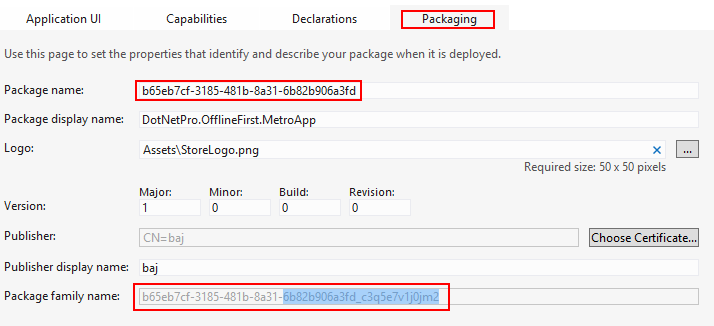
Ähnlich wie in Silverlight- oder Windows-Phone-Anwendungen kann man auch in Windows-8-Apps nicht an jeder beliebigen Stelle im Dateisystem speichern oder lesen.

Windows-8-Apps werden in einer Art Sandbox (App Container) ausgeführt. Jede App hat einen eigenen abgegrenzten Speicherbereich in den Anwendungsdaten des aktuellen Benutzerkontos. Das aus Silverlight als IsolatedStorage bekannte Gegenstück nennt sich unter Windows 8 LocalFolder und lässt sich innerhalb einer Windows-8-App über ApplicationData.Current.LocalFolder abfragen.

Ein Zugriff auf diesen Ordner liefert im aktuellen Beispiel folgenden Verzeichnisnamen zurück:

%LocalAppData%\Packages\**b65eb7cf-3185-481b-8a31-6b82b906a3fd\_c3q5e7v1j0jm2**\LocalState

Die lange Zeichenkette im Verzeichnisnamen ergibt sich zum Teil aus dem Package Namen, den man im Application-Manifest festlegen kann, wie dies in der Abbildung 5 zu sehen ist.



In diesem Verzeichnis sind nun die erforderlichen Lese- und natürlich auch Schreibrechte vorhanden.

In Listings 2 ist zu sehen, wie die übergebenen Daten als XML-Datei unter Verwendung des DataContractSerializers serialisiert und gespeichert werden können.

public async Task SaveAsync<T>(string key, T objectToSerialize, List<Type> knownTypes)

{

var localFolder = ApplicationData.Current.LocalFolder;

StorageFile file = await localFolder.CreateFileAsync(key + ".xml", CreationCollisionOption.ReplaceExisting);

IRandomAccessStream raStream = await file.OpenAsync(FileAccessMode.ReadWrite);

using (IOutputStream outStream = raStream.GetOutputStreamAt(0))

{

var serializer = new DataContractSerializer(objectToSerialize.GetType(), knownTypes);

serializer.WriteObject(outStream.AsStreamForWrite(), objectToSerialize);

await outStream.FlushAsync();

}

raStream.Dispose();

}

Zu beachten ist hierbei natürlich, dass die Daten als XML-Datei unverschlüsselt und im Klartext lesbar abgespeichert werden. Obwohl das Verzeichnis für die Beispiel-App explizit als Datenverzeichnis zur Verfügung gestellt wurde und andere Windows-8-Apps auf diesen Ordern nicht ohne weiteres zugreifen können, ist dieses Verzeichnis und damit die darin enthaltenen Dateien auf herkömmlichem Wege für den aktuellen Benutzer oder native Windows-Programme lesbar. Sollten die Daten sicherheitskritisch oder nur nach Authentifizierung im Programm lesbar sein, müssen sie entsprechend verschlüsselt werden.

Sollten für den Zugriff auf einen Webservice spezielle Zugriffsrechte erforderlich sein, stellt WinRT zum Lesen und Speichern dieser Authentifizierungsinformationen die Klasse PasswordCredential aus den Namensbereich Windows.Security.Credentials zur Verfügung. Diese werden allerdings nicht im lokalen Anwendungsbereich gespeichert, sondern in einem sicheren Bereich, der auch zwischen verschiedenen Geräten synchronisiert werden kann. Mehr Optionen zum Speichern vertraulicher Informationen in Windows-8-Apps sind unter [3] in der MSDN zu finden.

Weitere Anwendungseinstellungen können über ApplicationData.Current.LocalSettings als Schlüssel-Wert-Paare gespeichert werden. Sollten diese Einstellungen auf mehreren Windows-8-Geräten über die Microsoft Cloud synchronisiert werden, empfiehlt es sich, diese über ApplicationData.Current.RoamingSettings zu verwalten. Prädestiniert für solche Daten sind anwendungsspezifische Benutzereinstellungen, die der Anwender möglicherweise nicht auf jedem seiner Geräte von neuem vornehmen möchte.

Über Roaming-Ordner lassen sich im Übrigen weitere Daten abspeichern. Der Roaming-Ordner lässt sich über ApplicationData.Current.RoamingFolder abfragen. Unter [4] sind Empfehlungen von Microsoft zum Umgang mit Roaming-Daten zu finden. Ein wichtiger und zu beachtender Aspekt hierbei ist, dass die Größe des zur Synchronisation verfügenden Cloud-Speichers begrenzt ist. Diese kann über ApplicationData.Current.RoamingStorageQuota abgefragt werden.

# Lokale Anwendungsdaten lesen

Die Anwendungsdaten sind nun lokal gespeichert. Im nächsten Schritt wird gezeigt, wie auf diese Daten wieder zugegriffen werden kann. Beim späteren Abruf von aktuellen Daten über den Webservice ergibt sich schnell ein weiteres Problem. Die neuen Daten müssen mit den Offline-Daten synchronisiert werden. Je nach Anwendungsfall kann hierfür eine ganz unterschiedliche Strategie erforderlich sein.

In unserer Beispiel-App beschränken wir uns darauf, die im jeweiligen Store gespeicherten Daten einfach durch die vom Webservice zurückgelieferten zu ersetzen. In anderen Fällen wäre vielleicht ein Vergleich und das Zusammenführen der Daten notwendig.

In Listing 3 wird exemplarisch gezeigt, wie der CustomerStore zuerst den OfflineStore abfragt, um dem Benutzer möglichst schnell die zuletzt lokal abgespeicherten Customers anzuzeigen, bevor die Daten dann explizit über den WebApiService online geladen werden.

public async Task LoadAsync()

{

if (\_customers == null)

{

\_customers = await this.OfflineStore.LoadAsync<List<Customer>>("customers", this.KnownTypes);

this.OnNext(\_customers);

}

if (this.NetworkStatus.IsOnline)

{

var customers = new List<Customer>(await this.WebApiService.GetCustomersAsync());

this.OnNext(customers);

\_customers.Clear();

\_customers.AddRange(customers);

}

}

Durch den Aufruf von IObservable.OnNext werden hierbei dem ViewModel, das die Rolle des Observers übernimmt, die jeweiligen Daten zur Verfügung gestellt. Das ViewModel selbst muss erkennen, dass es die Daten zu den gleichen Customern eventuell zwei Mal bekommt.

In Listing 4 ist eine mögliche Implementierung abgebildet:

private void OnNext(IEnumerable<Customer> customers)

{

Dispatch.Action(() => CreateOrUpdateViewModels(customers));

}

private void CreateOrUpdateViewModels(IEnumerable<Customer> customers)

{

foreach (var customer in customers)

{

string customerId = customer.CustomerId;

var viewModel = (from vm in this.Customers where vm.CustomerId == customerId select vm).FirstOrDefault();

if (viewModel == null)

{

viewModel = new CustomerViewModel() { Customer = customer };

this.Customers.Add(viewModel);

}

else

{

viewModel.Customer = customer;

}

}

}

In Listing 3 ist auch zu sehen, dass der Zugriff auf den Webservice nur für den Fall durchgeführt wird, in dem auch eine Verbindung zum Internet besteht.

Um das feststellen zu können, stellt die WinRT-API über die Klasse NetworkInformation die Methode GetInternetConnectionProfile zur Verfügung. Im Online-Zustand gibt diese Methode ein entsprechendes Connection-Objekt zurück. Über dieses Profil kann dann über GetNetworkConnectionLevel abgefragt werden, ob aktuell eine Verbindung zum Internet besteht.

In Listing 5 wird die entsprechende Implementierung von NetworkStatus.IsOnline gezeigt.

public bool IsOnline

{

get

{

return (NetworkInformation.GetInternetConnectionProfile() != null &&

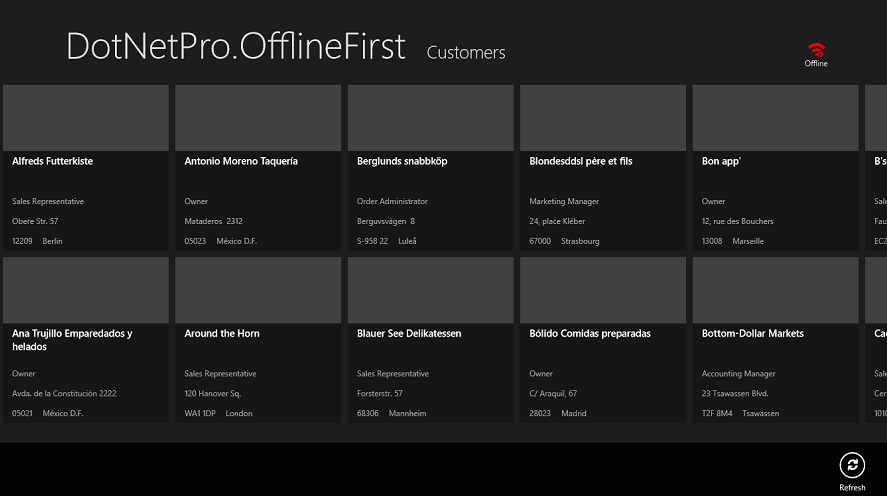
NetworkInformation.GetInternetConnectionProfile().GetNetworkConnectivityLevel() == NetworkConnectivityLevel.InternetAccess);

}

}

Über die aktuelle Geschwindigkeit und Qualität der Internetverbindung und ob der Webservice selbst gerade online ist, kann die NetworkInformation-Klasse natürlich keine Auskunft geben. Dieser Zustand muss separat behandelt werden.

Die NetworkInformation-Klasse teilt über das Ereignis NetworkStatusChanged relativ zeitnah mit, dass sich der Status geändert hat. Dadurch kann dem Benutzer visuell angezeigt werden, ob gerade eine Internetverbindung besteht (siehe Abbildung 5).



# Fazit

Windows 8 ist dafür konzipiert, auf mobilen Geräten wie Tablets oder Smartphones zu laufen. Da diese Geräte zwangsläufig nicht immer online sind, benötigen Sie für eine gute App eine Strategie, wie Sie mit diesem Offline-Zustand umgehen.

Über die WinRT-API ist es dabei möglich, Daten in lokalen oder Roaming-Ordnern zu speichern und dem Benutzer für einen späteren Offlinezugriff zur Verfügung zu stellen. Im Sinne des Offline-First Ansatzes werden die lokal gespeicherten Daten nicht nur genutzt, um ohne Internetverbindung lauffähig zu sein, sondern auch um Performance zu gewinnen. Während der Benutzer schon die Daten aus der letzten Online-Sitzung sieht, wird der aktuelle Stand über eine langsame Verbindung im Hintergrund abgefragt.

Zusammen mit einer guten Usability, die den Anwender nicht bevormundet und gut über den aktuellen Status der App informiert, ist der Grundstein für eine neue Top-App gelegt. Der nächste Tunnel kann kommen!

[1] <http://msdn.microsoft.com/en-us/library/windows/apps/xaml/hh994640.aspx>

[2] <http://msdn.microsoft.com/en-us/library/windows/apps/hh465088.aspx>

[3] <http://msdn.microsoft.com/en-us/library/windows/apps/xaml/br229572.aspx>

[4] <http://msdn.microsoft.com/en-us/library/windows/apps/hh465094.aspx>

Jan Baer und Jörg Heinichen sind bei der Zühlke Engineering GmbH als Software-Ingenieure mit den Schwerpunkten Microsoft-Technologien, Web- und Mobile-Anwendungen tätig. Gerne stehen sie für Fragen unter [jan.baer@zuehlke.com](mailto:jan.baer@zuehlke.com) und [joerg.heinichen@zuehlke.com](mailto:joerg.heinichen@zuehlke.com) zur Verfügung.