Unified-E Windows Application Adapter
Benutzerhandbuch

Windows-Applikationen als Endpunkt verwenden und Datenpunkte konfigurieren

Software-Version 3.1.0.0, zuletzt aktualisiert: Juli 2025

Herausgeber: Unified-E AG, Winterthur, Schweiz

Inhalt

1 Allgemeines ... 3
1.1 Kommunikation ... 3
1.2 Prinzipieller Ablauf ... 3
2 LabVIEW Beispiel ... 3
2.1 .NET Bibliothek einbinden ... 4
2.2 LabVIEW VI erstellen ... 4
2.3 Bedienfeld erstellen ... 5
2.4 Blockdiagramm erstellen ... 5
2.5 App-Projekt erstellen ... 6
2.5.1 Endpunkt und Datenpunkte konfigurieren ... 6
2.5.2 Datenpunkte in Ansicht verwenden ... 6
2.6 App mit App Manager veröffentlichen ... 8
3 .NET Beispiel mit C# ... 8
3.1 Einbinden der .NET Bibliothek ... 8
3.2 Bereitstellen der Datenpunkte im Windows-Programm ... 9
3.2.1 Variante 1: Anbindung mit dem ApplicationDataPoints-Objekt ... 9
3.2.2 Variante 2: Anbindung mit dem DataPointRequestListener-Objekt ... 12
4 Adapter-Parameter ... 14
5 Adressierung von Datenpunkten im App-Designer ... 15

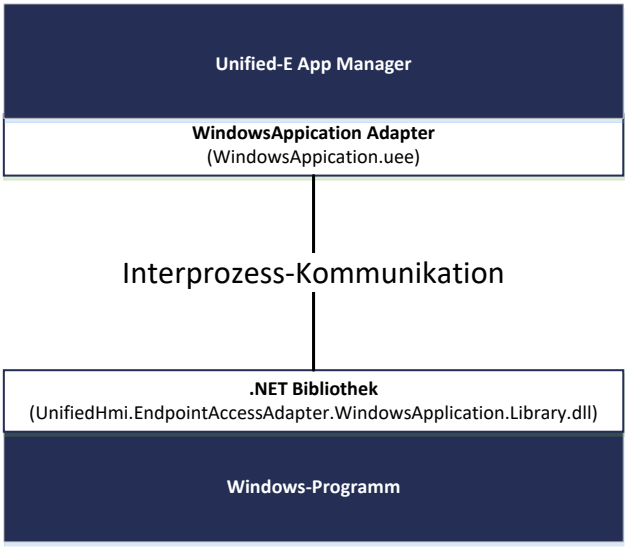
1 Allgemeines

Mit dem Windows Application Adapter ist es möglich, dass ein Windows-Programm (z. B. Unternehmens-Software, LabView, MES-System) als Unified-E Endpunkt fungiert und Datenpunkte für die Bedien-App bereitstellt.

1.1 Kommunikation

Das Windows-Programm muss definieren, welche Datenpunkte für Unified-E veröffentlicht werden und diese Datenpunkte entsprechend auslesen bzw. schreiben. Dafür muss im Windows-Programm eine .NET-Bibliothek eingebunden werden, welche die Hantierung mit Datenpunkten unterstützt und die Kommunikation zwischen dem lokalen Unified-E Dienst (App-Manager) und des Windows Programms ermöglicht.

Da der App-Manager mit dem Windows-Programm mittels Interprozess-Kommunikation kommuniziert, müssen beide Programme auf demselben Rechner installiert sein.



1.2 Prinzipieller Ablauf

1. Das Windows-Programm mit Hilfe der .NET Bibliothek erweitern, damit die Datenpunkte zugänglich werden.
2. Die App mit dem Unified-E App Designer erstellen. Es werden die vom Windows-Programm veröffentlichten Datenpunkte genutzt.
3. Das App-Paket im Unified-E App Manager installieren. Die berechtigten Benutzer können sich anschliessend für die installierte App registrieren.

2 LabVIEW Beispiel

Der Windows Application Adapter kann auch für LabVIEW verwendet werden. Im Moment muss dafür noch direkt die .NET-Bibliothek verwendet werden mit Hilfe der LabVIEW-internen .NET Mechanismen.

Die .NET Bibliothek ist für die CLR 4.0 entwickelt und kann daher für LabVIEW 2013 oder neuere Versionen verwendet werden.

Es wird vorausgesetzt, dass Sie bereits mit den Grundlagen von Unified-E vertraut sind. Eine Beschreiben ist unter www.unified-e.com zu finden oder in den Handbüchern/Hilfe des Unified-E App Designers bzw. Unified-E App Managers.

Die Handbücher sind online zu finden:

http://unified-e.com/en/Downloads/S

2.1 .NET Bibliothek einbinden

Die .NET Bibliothek ist im Windows Application Adapter enthalten. Daher muss zuerst der Adapter auf dem Computer installiert werden mit Hilfe des Unified-E App Designers oder Unified-E App Managers (siehe Handbuch).

Nach der Adapterinstallation kann die .NET Bibliothek wie folgt lokalisiert werden.

Verzeichnis:

C:\Program Files (x86)\Unified-E\App Designer\Endpoint Adapters\Windows Application Adapter <Version>

DLL-Name:

UnifiedHmi.EndpointAccessAdapter.WindowsApplication.Library.dll.

Wir empfehlen die DLL-Datei in den LabVIEW-Projektordner zu kopieren. Es muss sichergestellt werden, dass die DLL mit dem VI-Paket ausgeliefert wird.

2.2 LabVIEW VI erstellen

Wie bereits oben beschrieben muss ein Endpunkt wie eine LabVIEW-VI die Datenpunkte veröffentlichen, um mit dem Smartphone Daten austauschen zu können. Dies ist vergleichbar mit dem "Shared variables"-Konzept von LabVIEW.

Beispielprogramm:

Eine LabVIEW-VI veröffentlicht einen Text mit dem Datenpunktnamen "Datapoint1" und einen numerischen Wert mit dem Datenpunktnamen "Datapoint2". Im Folgenden wird immer auf dieses Beispiel Bezug genommen.

2.3 Bedienfeld erstellen

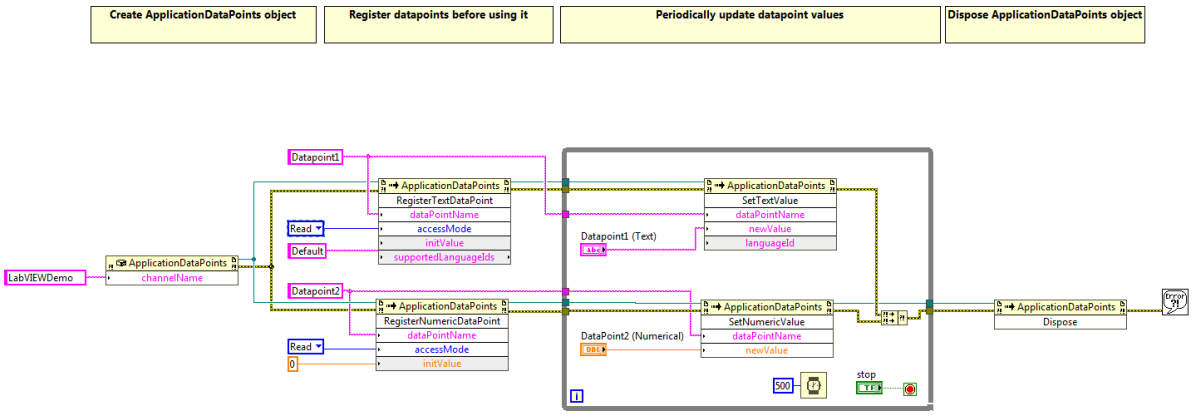
Für Demozwecke werden direkt die Werte der veröffentlichten Datenpunkte eingegeben. In den meisten Fällen sind Datenpunktewerte eingelesene Sensorwerte, z. B. die aktuelle Temperatur oder der Druck.

2.4 Blockdiagramm erstellen

Das Blockdiagramm enthält den Code, um die im Bedienfeld eingegebenen Werte zu veröffentlichen. Diese werden dann den Unified-E Programmen bereitgestellt und letztendlich in der Bedienapp angezeigt.

Prinzipieller Ablauf:

1. ApplicationDataPoints-Objekt erzeugen und einen beliebigen Kanalnamen setzen; dieser muss auf dem Computer für alle Unified-E Anwendungen eindeutig sein.
2. Alle Datenpunkte registrieren
3. Sicherstellen, dass die Datenpunktwerte aktuell sind, z. B. indem sie periodisch auf den neusten Wert gesetzt werden
4. Die .NET-Referenz wieder schliessen, sobald sie nicht mehr benötigt wird (bzw. Dispose-Methode aufrufen am ApplicationDataPoints-Objekt)



2.5 App-Projekt erstellen

2.5.1 Endpunkt und Datenpunkte konfigurieren

Endpunkt-Adapter wählen:

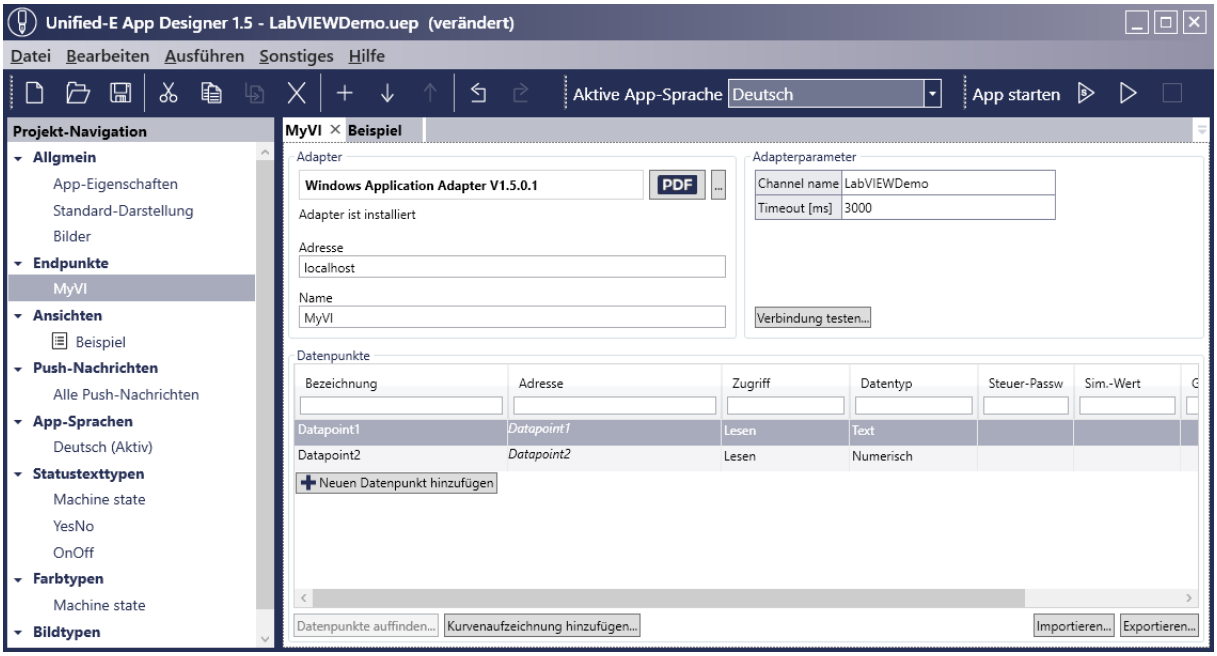
1. Datenpunkt-Editor öffnen
2. Den "Windows Application" Adapter wählen, ggf. installieren, falls dieser noch nicht aufgelistet wird.
3. Den Kanalnamen (Channel name) setzen: Dieser muss mit dem Kanalnamen im LabVIEW-Blockdiagramm übereinstimmen.

Für das Beispiel wird "LabVIEWDemo" als Kanalname verwendet.

Datenpunkte konfigurieren:

Es muss sichergestellt sein, dass alle definierten Datenpunkte auch im Blockdiagramm mit dem richtigen Namen, Typ und Zugriff registriert sind.

Für das Beispiel werden zwei Datenpunkte konfiguriert – passend zu den registrierten Datenpunkten (siehe Bild).

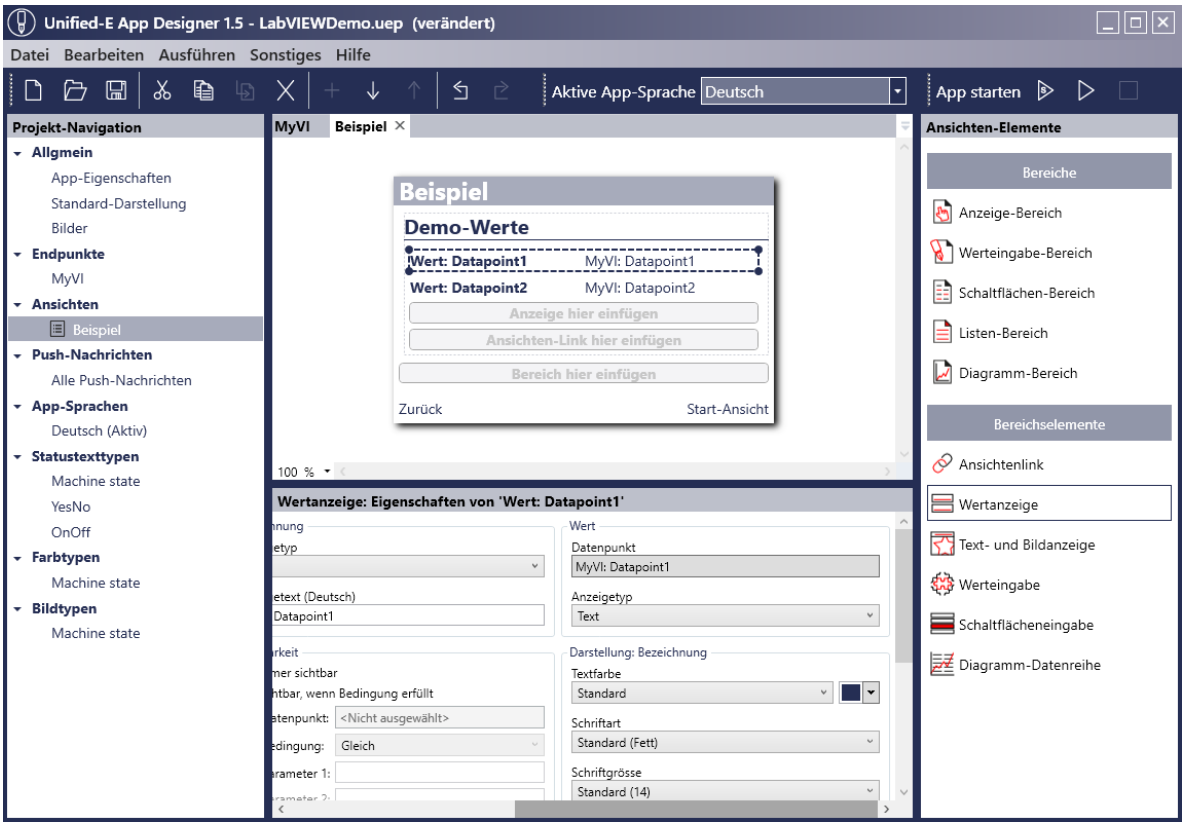


2.5.2 Datenpunkte in Ansicht verwenden

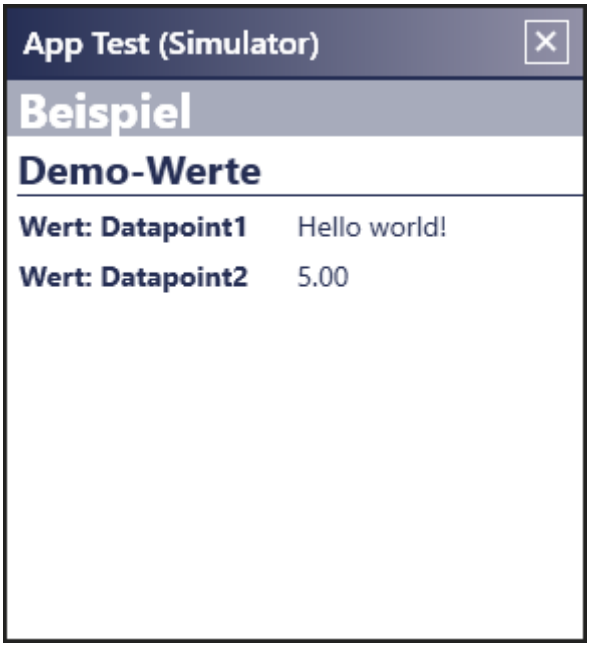
Nachdem die Datenpunkte definiert sind können diese in der Ansichtenkonfiguration oder auch für Push-Nachrichten verwendet werden.

Ansicht für das LabVIEW-Beispiel erstellen:

1. Anzeige-Bereich hinzufügen (mit Drag&Drop)
2. Zwei Wertanzeige-Elemente dem Anzeige-Bereich hinzufügen
3. Die Eigenschaften der Wert-Anzeige setzen; der Wert muss mit dem entsprechenden Datenpunkt verknüpft werden (siehe Eigenschaften-Bereich im Bild unten)



Nachdem die Ansicht konfiguriert wurde kann diese mit dem Simulator getestet werden (F5-Taste):



2.6 App mit App Manager veröffentlichen

Nachdem die Bedienapp konfiguriert ist, muss diese mit dem Unified-E App Manager veröffentlicht werden, um die App auf das Smartphone zu bekommen.

1. App-Paket im Unified-E App Designer erstellen (Menü Ausführen -> App Paket erstellen).
2. App-Paket im Unified-E App Manager installieren
3. Jetzt können die berechtigten Benutzer registriert werden mit Hilfe der Smartphone-App "Unified-" aus dem Store.

3 .NET Beispiel mit C#

3.1 Einbinden der .NET Bibliothek

Die .NET-Bibliothek wird beim Installieren des Windows Application Adapters in folgendes Verzeichnis kopiert:

C:\Program Files (x86)\Common Files\Unified-E\Endpoint Adapters\Windows Application Adapter <Version>

DLL-Name:

UnifiedHmi.EndpointAccessAdapter.Application.Library.dll.

Es handelt sich um eine .NET-Assembly (derzeit für .NET Runtime 4.0 bzw. höher, weitere auf Anfrage).

Die .NET-Bibliothek ist im Windows-Programm zu referenzieren und auszuliefern.

3.2 Bereitstellen der Datenpunkte im Windows-Programm

Im Folgenden wird bei Methodensignaturen bzw. Beispielcodes immer C# verwendet.

Grundsätzlich gibt es zwei Varianten, wie man Datenpunkte im Windows-Programm bereitstellen kann:

- Anbindung mit dem ApplicationDataPoints-Objekt
- Anbindung mit dem DataPointRequestListener-Objekt

Wichtig: Sie können nicht beide Objekte gleichzeitig mit dem selben Kanal-Namen verwenden!

Für beide Varianten ist Beispielcode verfügbar (C#-Code App Designer Projekt):

C:\Program Files (x86)\Common Files\Unified-E\Endpoint Adapters\Windows Application Adapter <Version>\CsharpSample.zip

3.2.1 Variante 1: Anbindung mit dem ApplicationDataPoints-Objekt

Datenpunkte sind die Schnittstelle zwischen dem Windows-Programm und Unified-E. Die zu veröffentlichten Datenpunkten des Windows-Programms müssen zuerst registriert und schliesslich regelmässig aktualisiert bzw. ausgelesen werden. Dafür wird in der .NET-Bibliothek die Klasse *ApplicationDataPoints* bereitgestellt.

3.2.1.1 ApplicationDataPoints-Objekt erstellen

Syntax:

```
public ApplicationDataPoints(string channelName);
```

Der Parameter "channelName" identifiziert den Kommunikationskanal zwischen Adapter und Bibliothek eindeutig. Dieser muss identisch sein mit dem "channel name" beim Adapter-Parameter und kann frei vergeben werden.

Allgemein:

Unified-E kennt vier Datentypen:

- YesNo: Ja/Nein
- Numeric: 64-Bit Gleitpunktzahl
- Text: Zeichenkette
- DataTable: Tabelle mit Zeichenketten pro Zelle

Für alle Typen gibt es entsprechende Get-, Set- und Register-Methoden. Im Folgenden werden beispielhaft die Methoden des Datentyps "Numeric" vorgestellt.

Alle Get- und Set-Methoden haben als optionalen Parameter "languageOrUser". Falls der Endpunkt mit Datenpunkt-Kontext "User" oder "Language" konfiguriert ist, dann können Sie mit diesem Parameter den benutzer- bzw. sprachspezifischen Wert lesen oder auch schreiben. Wenn languageOrUser den Wert "null" hat, dann gilt dieser Wert für alle Sprachen bzw. Benutzer.

3.2.1.2 Datenpunkt registrieren

Bevor Datenpunkt-Werte gesetzt oder ausgelesen werden können, müssen diese registriert werden.

Syntax:

```
public void RegisterNumericDataPoint(string dataPointName, DataPointAccessMode
accessMode, double initialValue, bool forceGlobal);
```

dataPointName: Der Datenpunkt-Name, dieser ist dann auch im App-Designer in der Datenpunkt-Tabelle in der Name- bzw. Adress-Spalte zu verwenden.

accessMode: Bei "Read" darf dieser Datenpunkt nur gelesen werden in einer Bedien-App, d. h. das Windows-Programm aktualisiert diesen Wert regelmässig. Bei "ReadAndWrite" darf die Bedien-App den Datenpunkt zum Steuern setzen, z. B. um einen Prozess zu starten oder zu stoppen. Das Windows-Programm hat diesen Datenpunkt daher zyklisch auszulesen und je nach Wert entsprechende Aktionen vorzunehmen.

initialValue: Der Start-Wert des Datenpunktes.

forceGlobal: Wenn true, dann wird der Datenpunkt immer wie ein globaler Datenpunkt behandelt, auch wenn der Datenpunk-Kontext bei den Adapterparametern auf "User" gesetzt ist.

3.2.1.3 Datenpunkt-Wert aktualisieren

Syntax:

```
public void SetNumericValue(string dataPointName, double newValue);
```

dataPointName: Der Name des zu aktualisierenden Datenpunktes.

newValue: Der neue Wert des Datenpunktes.

3.2.1.4 Datenpunkt-Wert auslesen

Dies ist erforderlich, wenn eine Bedien-App auch Datenpunkt-Werte setzen darf (DataPointAccessMode.ReadAndWrite).

Syntax:

```
public double GetNumericValue(string dataPointName);
```

dataPointName: Der Name des auszulesenden Datenpunktes.

Rückgabewert: Der aktuelle Datenpunkt-Wert, dieser kann vom Windows-Programm selbst oder auch von der Bedien-App (via App-Manager) gesetzt worden sein.

3.2.1.5 Benachrichtigung bei Änderung von Datenpunkt-Werten

Wenn der Bedien-App Benutzer einen Datenpunkt-Wert ändert, dann wird das Ereignis DataPointsExternallyModified ausgelöst. Damit ist es nicht erforderlich, alle Datenpunkt-Werte zyklisch auszuwerten.

3.2.1.6 Benachrichtung bei Wert-Anfragen

Mit Hilfe des Ereignisses UpdateDataPointsRequested muss der Endpunkt nicht zyklisch die Datenpunkt-Werte aktualisieren, sondern lediglich auf Anfrage.

Bei einer Datenpunkt-Wert Anfrage wird das Ereignis UpdateDataPointsRequested ausgelöst. Damit hat der Endpunkt die Möglichkeit, die anfragenden Datenpunkte zu aktualisieren, so dass anschliessend die die neusten Werte an die Clients zurückgegeben werden.

3.2.1.7 ApplicationDataPoints-Objekt freigeben

Beim Beenden des Windows-Programms muss die Dispose-Methode aufgerufen werden, damit belegte System-Resourcen wieder freigegeben werden.

3.2.1.8 Beispiel: Ablaufsteuerung im Windows-Programm

```
private static void Program()
{
    // -------------- INIT ------------------

    // Step 1: Create ApplicationDataPoints
    ApplicationDataPoints applicationDataPoints = new ApplicationDataPoints("ReadOnlyLoopSample");

    // Step 2: Register all datapoints which should be published
    applicationDataPoints.RegisterNumericDataPoint("Device1.Temperature", DataPointAccessMode.Read, 0);
    applicationDataPoints.RegisterNumericDataPoint("Device1.Pressure", DataPointAccessMode.Read, 0);
    applicationDataPoints.RegisterNumericDataPoint("Device2.Temperature", DataPointAccessMode.Read, 0);
    applicationDataPoints.RegisterNumericDataPoint("Device2.Pressure", DataPointAccessMode.Read, 0);
```

```
            // -------------- LOOP ------------------

            while (!m_IsFinished)
            {
                // do work...
                Thread.Sleep(50);

                // read sensor data
                double device1Temperature = GetDevice1Temperature();
                double device1Pressure = GetDevice1Pressure();
                double device2Temperature = GetDevice2Temperature();
                double device2Pressure = GetDevice2Pressure();

                // Step 3: Update datapoint values so the Application Adapter can retrieve the current values
                applicationDataPoints.SetNumericValue("Device1.Temperature", device1Temperature);
                applicationDataPoints.SetNumericValue("Device1.Pressure", device1Pressure);
                applicationDataPoints.SetNumericValue("Device2.Temperature", device2Temperature);
                applicationDataPoints.SetNumericValue("Device2.Pressure", device2Pressure);
            }


            // -------------- EXIT ------------------

            // Step 4: Dispose object
            applicationDataPoints.Dispose();
        }
```

3.2.2 Variante 2: Anbindung mit dem DataPointRequestListener-Objekt

Bei dieser Variante wird für jede Anfrage des App Manager bzw. App Designers ein Ereignis ausgelöst, welches im Windows-Programm behandelt werden muss.

Vorteil:

Es müssen nur Datenpunkt-Werte ermittelt werden (z. B. über eine Datenbank), wenn tatsächlich eine Anfrage eingetroffen ist.

Nachteil:

Der App Manager (und damit auch der Smartphone-Benutzer) müssen unter Umständen länger warten, weil erst bei der Anfrage die Wert-Ermittlung erfolgt.

3.2.2.1 DataPointRequestListener-Objekt erzeugen

Das DataPointRequestListener-Objekt befindet sich him folgenden Namespace:

UnifiedHmi.EndpointAccessAdapter.WindowsApplication.Library.NativeAccess

3.2.2.2 Behandlung von Anfragen

Für die Behandlung von Anfragen stehen folgende Ereignisse zur Verfügung

- *ReadValuesRequested*
- *WriteValuesRequested*
- *ReadDataTablesRequested*

Eine Verwendung der Ereignisse finden Sie im Beispielcode unter

C:\Program Files (x86)\Common Files\Unified-E\Endpoint Adapters\Windows Application Adapter <Version>\CsharpSample.zip

3.2.2.3 Optimierung der Tabellenzugriffe

Um zu vermeiden, dass bei jeder Anfrage die vollständige Tabelle zurückgeliefert werden muss, wurde eine «ModificationId» eingeführt.

Der Programmierer sollte bei Tabellenänderungen immer die ModificationId ändern. - Typischerweise ist diese Id ein laufender Zähler oder Zeitstempel. Diese Id ist dann bei jedem ReadDataTablesRequested-Aufruf zu setzen.

Bei jeder ReadDataTablesRequested-Anfrage wird die LastModificationId mitgeliefert. Wenn diese identisch ist mit der aktuellen ModificationId, dann muss die Tabelle nicht als Value gesetzt werden, da keine Änderungen vorgenommen wurden.

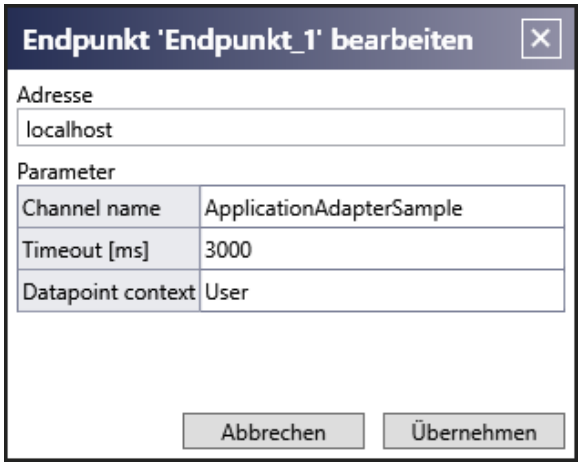
Beispiel:

```
        private void m_DataPointRequestListener_ReadDataTablesRequested(object sender,
DataTableDataPointsEventArgs e)
        {
            string user = GetMappedUser(e.User);

            foreach (var dataPoint in e.DataPoints)
            {
                DataTableValue value = (DataTableValue)GetVariable(user,
dataPoint.Address);
                if (value != null)
                {
                    // consider no changes
                    dataPoint.ModificationId = value.ModificationId;
                    if (dataPoint.LastModificationId != value.ModificationId)
                    {
                        // value must only be set on changes
```

```
                    dataPoint.Value = value.Value;
                }
            }
        }
    }
```

4 Adapter-Parameter



Adresse:

Dort muss derzeit immer localhost eingetragen sein.

Channel name:

Der eindeutige Name für den Kommunikationskanal. Dieser muss mit dem channelName im ApplicationDataPoints-Konstruktor übereinstimmen.

Timeout [ms]:

Timeout-Wert.

Datapoint context:

Folgende Datenpunkt-Kontexte werden unterstützt.

Global:

Alle Benutzer bekommen beim Anfragen eines bestimmten Datenpunktes denselben Wert geliefert. Dies ist üblich bei Datenpunkten, die Maschinenzustände repräsentieren.

User:

Unterschiedliche Benutzer können beim Anfragen eines bestimmten Datenpunktes unterschiedliche Werte erhalten. Beispiele:

- Diagramm- oder Listenanzeige: Benutzer wollen nur ihren eingestellten Bereich sehen, der ebenfalls über benutzerspezifische Datenpunkte konfiguriert ist.
- Aktuelle Produktionsaufträge des App-Benutzers.
- Leistungskennzahlen des App-Benutzers.

Language:

Alle Benutzer einer Sprache erhalten denselben Datenpunkt-Wert für einen bestimmten Datenpunkt. Dieser Kontext macht dann Sinn, wenn Datenpunkte vom Typ „Text“ mehrsprachig sein sollen.

5 Adressierung von Datenpunkten im App-Designer

Datenpunkte werden in der Datenpunkt-Tabelle definiert (siehe unten).

Die Adresse ist der Datenpunkt-Name, der vom Windows-Programm beim Registrieren vergeben wird. Falls keine Adresse in der Datenpunkt-Tabelle eingetragen wird, dann wird die Datenpunkt-Bezeichnung für die Adresse verwendet.