

# **Unified-E Windows Application Adapter User Manual**

## **Configure Siemens PLC Endpoints and Datapoints**

Software version 3.1.0.0, last updated: July 2025

Publisher: Unified-E AG, Winterthur, Switzerland



## Content

<b>1</b>	<b>General.....</b>	<b>3</b>
1.1	Communication .....	3
1.2	Basic Workflow .....	3
<b>2</b>	<b>LabVIEW Example.....</b>	<b>3</b>
2.1	How to get the .NET Library .....	4
2.2	Create the LabVIEW VI .....	4
2.2.1	Create the Front Panel .....	4
2.2.2	Create the Block Diagram .....	5
2.3	Create Unified-E App Project .....	5
2.3.1	Configure the Endpoint and Datapoints .....	5
2.3.2	Datapoint Usage within View .....	6
2.4	App Deployment via App Manager .....	8
<b>3</b>	<b>.NET Example with C#.....</b>	<b>8</b>
3.1	Referencing the .NET Library .....	8
3.2	Providing Datapoints in the Windows Program.....	9
3.2.1	Option 1: Interfacing with the ApplicationDataPoints Object .....	9
3.2.2	Option 2: Interfacing with the DataPointRequestListener Object .....	12
<b>4</b>	<b>Adapter Parameters .....</b>	<b>14</b>
<b>5</b>	<b>Addressing Datapoints in the App Designer.....</b>	<b>15</b>

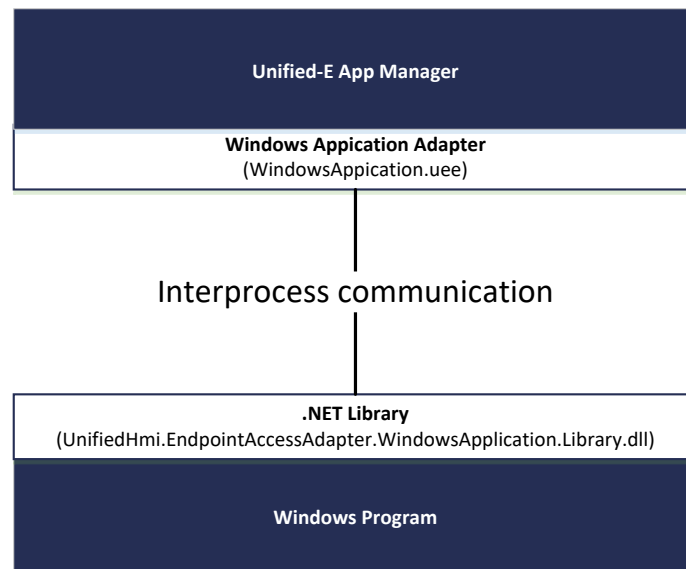
# 1 General

The Windows Application Adapter allows to connect to a windows program (e. g. enterprise software, LabVIEW, MES system). The Windows program acts as an endpoint exposing datapoints for an operator app.

## 1.1 Communication

The windows program must decide which datapoints need to be published to become accessible via Unified-E. Therefore, a .NET library must be referenced in the windows program which allows reading and writing datapoint values. The library also handles the communication between App Manager and the windows program.

Communication is done via an interprocess communication mechanism which means both the App Manager and the windows program must run on the same computer.



## 1.2 Basic Workflow

1. Enable the Windows program to publish the required datapoints by using the .NET library.
2. Design the operator app in the Unified-E App Designer by using the published datapoints. Finally create an app package.
3. Install the app package on the Unified-E App Manager and allow granted users to register for operation/monitoring.

# 2 LabVIEW Example

Since LabVIEW also targets the Windows platform you can use the Windows Application adapter within LabVIEW. Up to now you must use the methods from the .NET library directly by using the LabVIEW .NET mechanisms.

The .NET library uses the CLR 4.0 and so can be used from LabVIEW 2013 and higher.

It is assumed you are already familiar with the basics of Unified-E. It is described on [www.unified-e.com](http://www.unified-e.com) or the user manuals of the Unified-E App Designer and Unified-E App Manager. You will find the user manuals online in the "Downloads" section:

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

## 2.1 How to get the .NET Library

The .NET library (assembly) is included in the Windows Application adapter. So, install the adapter first on the computer with the Unified-E App Designer or Unified-E App Manager (see program documentation).

After adapter installation, you will get the .NET library as follows.

Directory:

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

DLL Name:

*UnifiedHmi.EndpointAccessAdapter.WindowsApplication.Library.dll.*

We recommend copying this DLL file into your LabVIEW project folder. Make sure it gets deployed with your VI package.

## 2.2 Create the LabVIEW VI

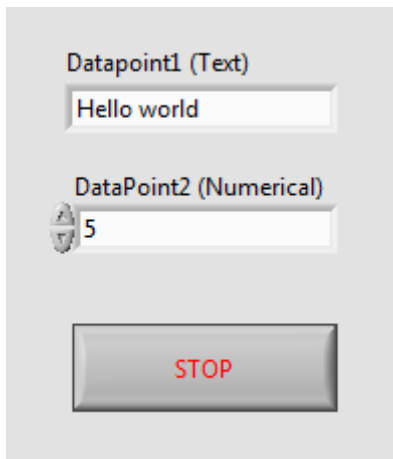
As described above an endpoint like a LabVIEW VI needs to publish the datapoints to exchange data with the Smartphone. For LabVIEW users, a datapoint is like a shared variable.

### Example application:

A LabVIEW VI will publish a string value called Datapoint1 and a numerical value called Datapoint2. The user may modify these values via controls.

### 2.2.1 Create the Front Panel

For illustration purposes the front panel is used for modifying the published datapoints. In typical programs the datapoint values might be sensor values e. g. the current temperature or pressure.

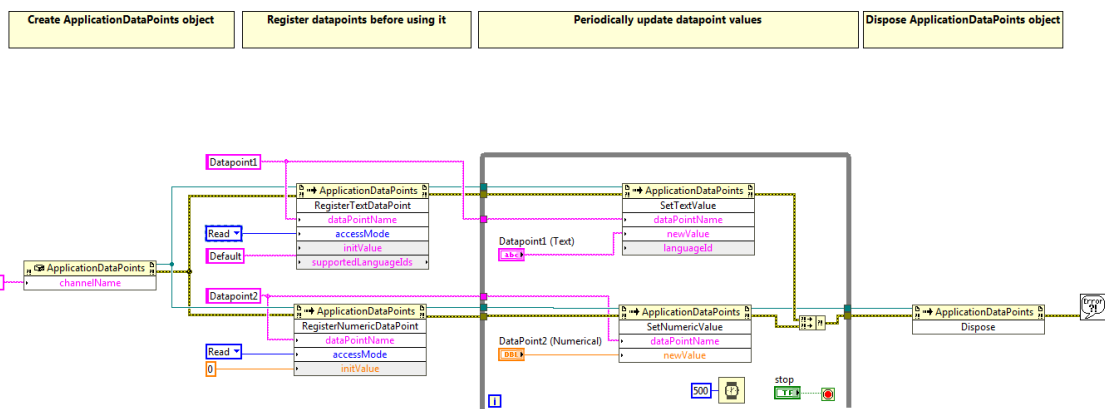


## 2.2.2 Create the Block Diagram

The block diagram contains the code to publish the entered values above for the Unified-E programs (and finally for the Smartphone operator app).

Basically, the control flow is as follows:

1. Create the ApplicationDataPoints object and set an arbitrary channel name which is unique on the computer for Unified-E
2. Register all datapoints
3. Make sure the datapoint values are up to date, e. g. by periodically setting the current value
4. Close the reference or call the Dispose method on the ApplicationDataPoints object, if it is no longer in use



## 2.3 Create Unified-E App Project

### 2.3.1 Configure the Endpoint and Datapoints

Set the endpoint:

1. Open the Endpoint/Datapoint editor
2. Select the "Windows Application" adapter, install it first if required

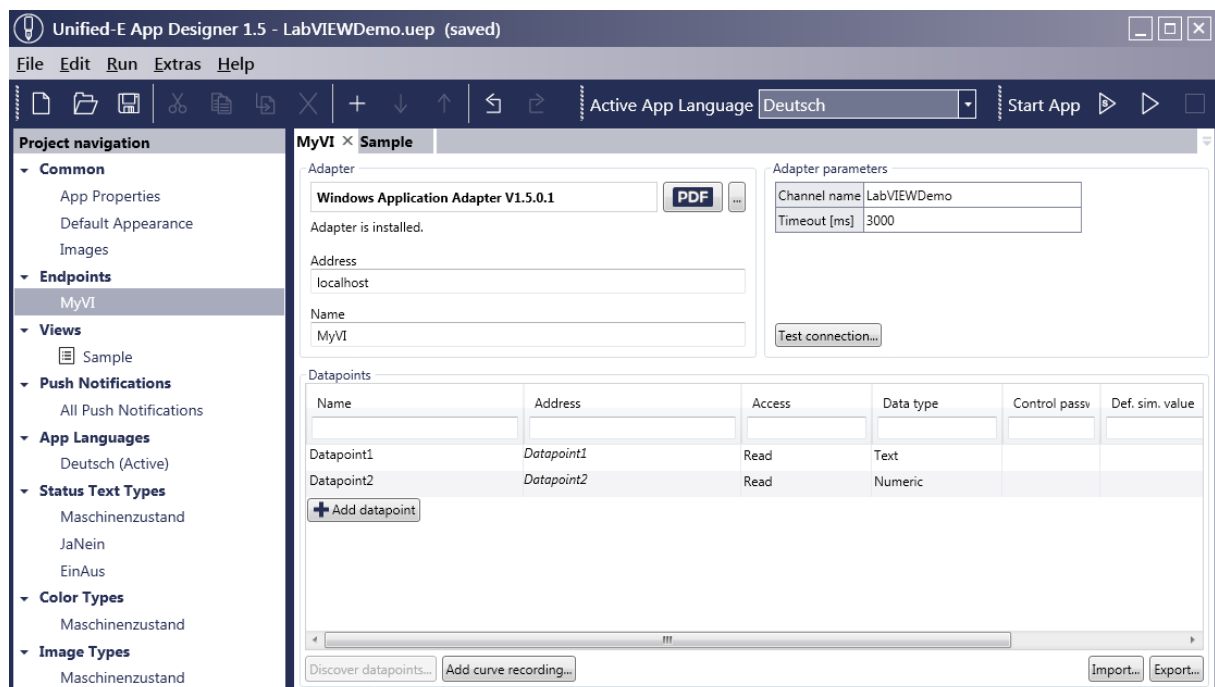
3. Set the channel name parameter: The name must match the channel name in the block diagram (ApplicationDataPoints constructor).

For the example set "LabVIEWDemo" as channel name.

### Configure the datapoints:

Make sure, all datapoints are also registered in the LabVIEW block diagram with the right access type, datatype and name.

For the LabVIEWDemo example two datapoints must be configured which associate with the registered datapoints in the block diagram above.

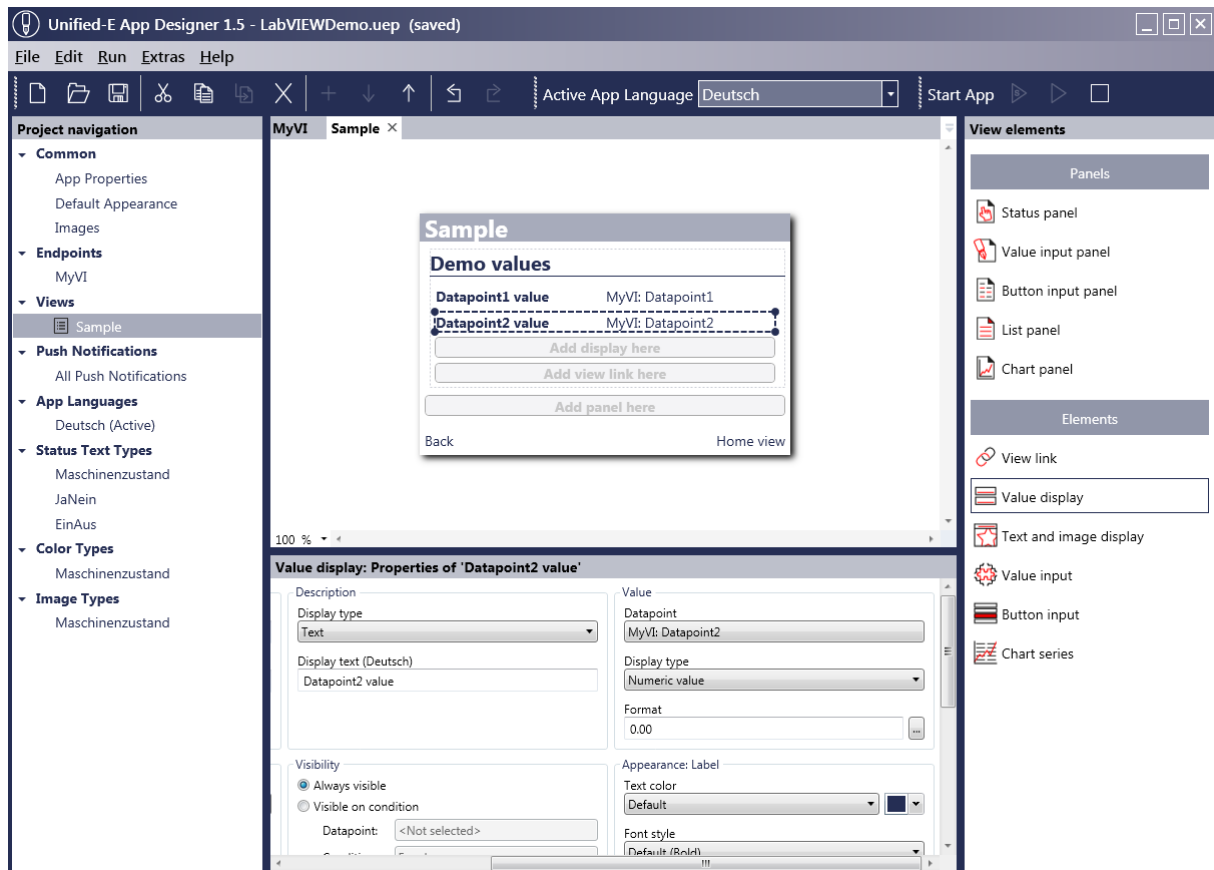


## 2.3.2 Datapoint Usage within View

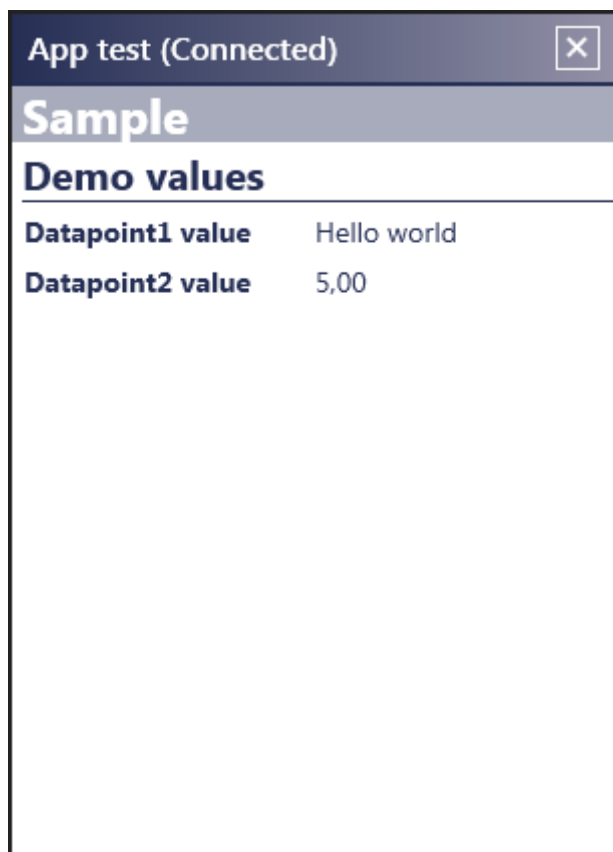
After defining the datapoints they can be used in a view configuration or push notification configuration.

### Create view for LabVIEW example:

1. Add a status panel to the view (via drag&drop)
2. Add two value display elements into the status panel
3. Set the properties so the value is attached to the appropriate datapoint (see picture below)



After configuring the view, the app simulator can be started with F5 key and looks like this:



## 2.4 App Deployment via App Manager

After you have configured your app you need to deploy it via the Unified-E App Manager to get it displayed on a Smartphone.

1. Create an app package in the Unified-E App Designer (Menu Run->Create App Package).
2. Install this app package in the Unified-App Manager.
3. Finally, the Smartphone users can register the operator app with the Unified-E app from store.

## 3 .NET Example with C#

### 3.1 Referencing the .NET Library

During installation of the Windows Application Adapter the .NET library will be copied into the following directory:

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

DLL Name:

*UnifiedHmi.EndpointAccessAdapter.WindowsApplication.Library.dll.*



The dll file is a .NET assembly (requires .NET runtime 4.0 or higher, others on request).  
You must reference this library and deploy it with your windows program.

## 3.2 Providing Datapoints in the Windows Program

C# is used for all method signatures and example codes here.

Basically there are two options how to interface with a windows program:

- Use of the `ApplicationDataPoints` object
- Use of the `DataPointRequestListener` object

Important: You can't use both objects simultaneously with the same channel name!

There are C# examples available for both options (C# code and App Designer project):

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

### 3.2.1 Option 1: Interfacing with the `ApplicationDataPoints` Object

Datapoints are the interface between the windows program and Unified-E. The datapoints which need to be published have to be registered first. Then they must be read or updated periodically, for that the class `ApplicationDataPoints` is provided in the .NET library.

#### 3.2.1.1 Creating the `ApplicationDataPoints` Object

Syntax:

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

The parameter "channelName" identifies the communication channel between adapter and library. It must equal with the channel name given in the adapter parameter.

Common:

Unified-E knows four data types:

- YesNo
- Numeric
- Text
- DataTable

There are Get-, Set and Register methods for these data types. For the following examples, all methods are only presented for the data type "Numeric".

All Get- and Set-methods do have the optional parameter "languageOrUser". If the endpoint is configured with datapoint context "User" or "Language" then you can read or write the user- or language-specific value with this parameter.

If the languageOrUser parameter has the value null then the datapoint value is valid for all users/languages (global context).

### 3.2.1.2 Registering Datapoints

Before you can read and write datapoint values a datapoint has to be registered.

#### Syntax:

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

**dataPointName:** The datapoint name, this name must be used in the App Designer for addressing in the datapoint table.

**accessMode:** "Read" means the operator app is only allowed to read the value. So the datapoint value has to be updated periodically within the windows program.

"ReadWrite" means, that also an operator app can set the datapoint value for controlling purposes, e. g. to start or stop process. So, the windows program has to read the datapoint value periodically and depending on the datapoint value-specific actions have to be executed.

**initialValue:** The initial value of the datapoint.

**forceGlobal:** If true then the datapoint is handled like a global datapoint even if the datapoint context is set to "User" in the adapter parameters.

### 3.2.1.3 Updating Datapoint Values

#### Syntax:

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

**dataPointName:** Name of the datapoint to be updated.

**newValue:** The new datapoint value.

#### 3.2.1.4 Reading Datapoint Values

This is required if the operator app can write datapoint values.  
(DataPointAccessMode.ReadAndWrite).

##### Syntax:

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

dataPointName: The name of the datapoint to be read.

Return value: the current datapoint value. This could be written by the windows program itself or by the operator app (via App Manger).

#### 3.2.1.5 Notification on datapoint value changes

The event DataPointsExternallyModified notifies about changes of datapoint values which are done by an app user. So, it is not necessary to poll the datapoint values periodically to detect changes.

#### 3.2.1.6 Notification on datapoint value requests

With the help of the UpdateDataPointsRequested event, the endpoint does not have to periodically update the data point values, but only on request.

A data point value request raises the UpdateDataPointsRequested event. This allows the endpoint to update the requesting data points so that the latest values are returned to the clients.

#### 3.2.1.7 Releasing the ApplicationDataPoints Object

Before exiting the windows program, the Dispose method must be called to release system resources.

#### 3.2.1.8 Example: Sequential Control in a Windows Program

```
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 Option 2: Interfacing with the DataPointRequestListener Object

Here for each Unified-E request an event is raised which has to be handled in your windows program.

Pro:

Datapoint values are calculated or requested on demand (e. g. database query).

Contra:

The App Manager (and Smartphone users) have to wait a bit longer until the view is fully displayed because the values are calculated/requested on demand in the windows program

### 3.2.2.1 Creating the DataPointRequestListener Object

The class can be found in the following namespace:

*UnifiedHmi.EndpointAccessAdapter.WindowsApplication.Library.NativeAccess*

### 3.2.2.2 Handling Requests

The following events are available on the DataPointRequestListener class to handle requests:

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

You can find usages of the events in the example code:

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

### 3.2.2.3 Optimizing Data Table Access

A modification ID has been introduced which helps not transferring the complete data table on each request.

The modification id must be updated each time the table has been modified. Typically, the modification id is a counter or a time stamp and should be set on each ReadDataTablesRequested call.

The ReadDataTablesRequested event provides the modification id of the last call (LastModificationId). If this value is identical with the current modification id there is no need to transfer the data table since no changes have been made since the last call.

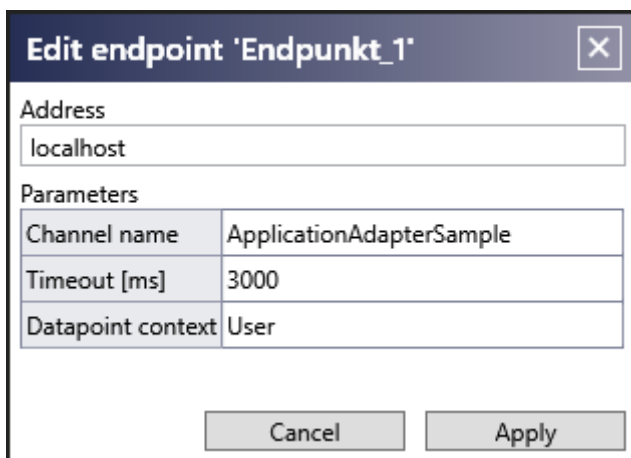
#### Example:

```
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 Parameters



**Edit endpoint 'Endpunkt\_1'**

Address  
localhost

Parameters

Channel name	ApplicationAdapterSample
Timeout [ms]	3000
Datapoint context	User

Cancel Apply

### Address:

Currently only "localhost" is allowed here.

### Channel name:

The unique channel name. This must match the name in the constructor of the ApplicationDataPoints class.

### Timeout [ms]:

The timeout value for an operation like connecting or reading values.

### Datapoint context:

The following data point contexts are supported.

### Global:

All users receive the same value when requesting a particular datapoint. This is typical for datapoints representing machine states.

## User:

Different users can receive different values when requesting a particular data point.

### Examples:

- Diagram or list display: Users only want to see their configured range which is also done via user-specific data points
- Current production orders of the app user
- Performance indicators of the app user

## Language:

All users of a language receive the same datapoint value for a particular datapoint. This context makes sense if datapoints of type "text" are to be multilingual.

# 5 Addressing Datapoints in the App Designer

Datapoints are defined in the datapoint table in the App Designer.

The address is the datapoint name which has been used in the windows program while registering the datapoint. If no address is set the datapoint name is used for addressing.

