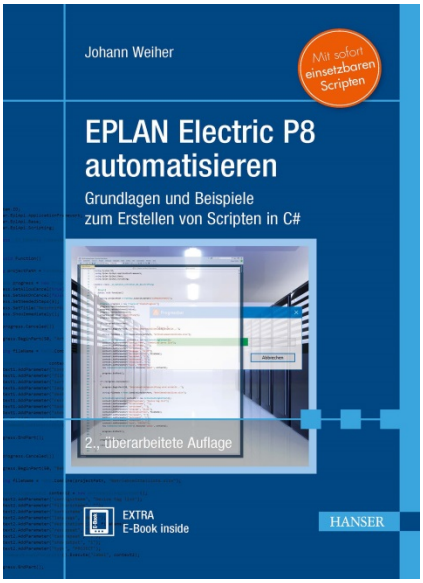
HANSER




# Leseprobe

zu

# EPLAN Electric P8 automatisieren

Grundlagen und Beispiele zum Erstellen von Scripten in C#.

Mit sofort einsetzbaren Scripten

von Johann Weiher

ISBN (Buch): 978-3-446-45492-7
ISBN (E-Book): 978-3-446-45712-6
ISBN (E-Pub): 978-3-446-45833-8

Weitere Informationen und Bestellungen unter
http://www.hanser-fachbuch.de/978-3-446-45492-7
sowie im Buchhandel

# Inhalt

# Vorwort

Liebe Leserinnen und liebe Leser,

mit diesem Buch möchte ich Ihnen einen einfachen und unkomplizierten Einstieg in die Erstellung von Scripten für EPLAN Electric P8 ermöglichen. Das Buch richtet sich an alle EPLAN-Anwender, ganz gleich, ob es sich dabei um regelmäßige oder sporadische Konstrukteure handelt, die mithilfe von Scripting ihre Aufgaben automatisieren wollen. Programmierkenntnisse werden nicht vorausgesetzt. Sie werden erstaunt sein, wie schnell dabei ein Resultat zustande kommt, das Sie begeistert. Schon mit einem kleinen Script, das aus nur ein paar Zeilen besteht, können Sie viel Zeit bei der Projektierung sparen. Auf Grundlage der im Buch vermittelten Informationen werden Sie rasch imstande sein, EPLAN-Aktionen zu verwenden und gegebenenfalls zu erweitern. Darüber hinaus lernen Sie auch, eigene Erweiterungen zu programmieren. Scripte können ab der Version EPLAN Electric P8 Compact genutzt werden. Das API-Modul ist dafür nicht erforderlich.

Im Script, wie in der Programmierung selbst, ist vieles, wenn nicht sogar alles, möglich. Deshalb stellt sich die Frage, in welchem Umfang dieses Buch das Themenfeld abdecken kann. Die Sprache C#, die ich zum Erstellen der Scripte verwende, ist sehr komplex und mit ihrer Beschreibung allein könnte man mehrere Tausend Seiten füllen, ohne irgendeine EPLAN-Funktion zu erklären. Aus diesem Grund beschränke ich mich darauf, die Grundlagen von C# zu vermitteln, die notwendig sind, um neue Scripte zu erstellen oder bestehende zu erweitern bzw. zu verändern. Auch auf die wichtigsten Erweiterungen durch eigenen Programmcode gehe ich ein.

Alle EPLAN-Aktionen werden anhand von praxisnahen Beispielen beschrieben und erklärt. Viele der Beispiele werden Ihren Workflow beschleunigen. Hinzu kommt, dass mehr Zeit für die wesentlichen Aufgaben, nämlich die der Konstruktion, bleibt. Jeder kennt die wiederkehrenden, monotonen Aufgaben, die z. B. beim Projektabschluss anstehen. Viele Auswertungen und Beschriftungen müssen erzeugt werden, zusätzlich muss der Plan als PDF erstellt werden. All dies können Sie per Knopfdruck erledigen. Wie? Das wird Schritt für Schritt im Buch erklärt.

Auf der Internetseite *https://eep8a.de* finden Sie das komplette Projekt mit allen Beispielen und fertigen Scripten, welche Sie direkt in EPLAN verwenden können.

An dieser Stelle möchte ich mich recht herzlich bei allen bedanken, die mir geholfen haben, dieses Buch zu schreiben.

Allen voran danke ich meiner Frau Daniela für die Motivation, das Buch zu schreiben, und die Unterstützung, um genügend Zeit zu finden. Vielen Dank auch an meine wundervollen Töchter Leni & Fina für die erfreulichen Unterbrechungen und Ablenkungen beim Schreiben.

Großer Dank geht an meinen Chef, Kollegen und Freund Michael Kastl für die Freiheit, dieses Buch zu schreiben. Es macht einfach Spaß, mit dir zu arbeiten.

Ein besonderer Dank gilt Florian Reiter – hier ist aus einem Berater ein Freund geworden. Ein großes Lob möchte ich auch Herrn Andreas Krämer für die immer sehr guten Hilfestellungen aussprechen.

Zu guter Letzt möchte ich mich bei Julia Stepp vom Carl Hanser Verlag für die Hilfe und Unterstützung bedanken.

Johann Weiher

Dürnhart, im Mai 2018

# 1 Einführung

### Was ist Scripting?

Scripting bezeichnet die Möglichkeit, einzelne Befehle bzw. Programmcode in EPLAN auszuführen. Dies geschieht über die sogenannte API (Application Programming Interface, dt. Programmierschnittstelle). Hinter der EPLAN-API verbergen sich alle Funktionen, die in der Plattform (Electric P8, Fluid, Pro-Panel usw.) vorhanden sind. Diese Programme bauen alle auf dem gleichen Programmcode auf und sind dadurch untereinander kompatibel. In den verschiedenen Applikationen sind ähnliche bzw. gleiche Funktionen enthalten, z. B. kann man Beschriftungen sowohl in Fluid als auch in Electric P8 erzeugen. Einziger Unterschied ist der Inhalt.

Diese Abläufe werden in EPLAN *Actions* (Aktionen) genannt. Ihnen ist Abschnitt 2.2, „Actions ausführen“, gewidmet, da es mehrere Wege gibt, solche Actions auszuführen. Das Wort Scripting bezieht sich meistens nur auf Scripte, die Sie ab der Ausbaustufe EPLAN Electric P8 Compact nutzen können. Um weitere Befehle oder Funktionen ausführen zu können, benötigen Sie das API-Modul von EPLAN. In diesem Buch gehe ich aber ausschließlich auf den Standardumfang der Compact-Version ein.

### Was sind Scripte?

Scripte sind kleine Programmcodes. In EPLAN können diese in zwei Programmiersprachen erstellt werden:

- Microsoft C# (C-Sharp)
- Microsoft VB.NET (Visual Basic.NET)

Ich werde in den folgenden Kapiteln nur Beispiele in C# bereitstellen, da EPLAN mit dieser Sprache fertigen Code generiert und dadurch eine optimale Vorlage liefert. Ein Script ist nicht alleine ausführbar. Es muss in Verbindung mit EPLAN gestartet werden.

### Was können Scripte?

Scripte können vieles, aber nicht alles. EPLAN stellt eine Reihe von Befehlen bereit, schränkt diese aber auf einen überschaubaren Bereich ein. Dadurch wird dem Anwender der Einstieg enorm erleichtert. Auf diese Weise wird auch sichergestellt, dass keine ungewollten Aktionen, z. B. auf das Projekt, ausgeführt werden.

Sicherlich kennen Sie die wiederkehrende Aufgabe, Beschriftungen auszugeben. Je Projekt sind mehrere Exporte nötig. Jedes Mal muss das Beschriftungsschema neu ausgewählt, zusätzlich der Ordner benannt und ein Dateiname vergeben werden. Mit einem Script können Sie all diese Arbeitsschritte zusammenfassen und z. B. auf einen Menüpunkt legen. Sie können über diese Funktion auch mehrere Beschriftungen nacheinander erzeugen. Auch der PDF-Export kann automatisiert werden. Möchten Sie z. B. beim Schließen des Projekts automatisch ein PDF zur Änderungsverfolgung erzeugen? Mit einem Script lässt sich dies problemlos realisieren. Sie wollen Schnittstellen schaffen und Informationen außerhalb von EPLAN, z. B. in Ihrem ERP-System, nutzen? Gar kein Problem! Über die Möglichkeiten im Scripting geht das auf Knopfdruck. Oft muss zwischen verschiedenen Einstellungen hin und her gewechselt werden. Das Suchen in den unzähligen Einstellungen in EPLAN ist mühselig. Schreiben Sie stattdessen ein Script für Ihre Konfigurationen und erledigen Sie dies unter der Projektierung.

Dies ist eine kleine Auflistung der Möglichkeiten, die mit Scripten realisiert werden können:

- Beschriftungen automatisieren
- PDF-Export
- Backup
- Eigene Menüs/Toolbars erstellen
- Grafische Formulare, z. B. mit Buttons, Checkboxen und Auswahldialogen, erstellen
- Eigenschaften verändern
  - Projekteigenschaften
  - Seiteneigenschaften
- Einstellungen
  - Lesen
  - Schreiben

Und das sind noch längst nicht alle Funktionen! Durch das Erweitern des Programmcodes können noch mehr Funktionen hinzugefügt werden.

**Was kann das API-Modul im Vergleich zum Scripting?**

Um den Unterschied etwas deutlicher zu machen, finden Sie im Folgenden eine kleine Auflistung der wichtigsten Merkmale des API-Moduls in EPLAN:

- Zugriff auf das komplette EPLAN-Datenmodell
- Einfacheres Lesen von Objekten
- Zugriff auf mehr Objekte
- Direkter Zugriff auf Projekteigenschaften/Projekteinstellungen
- Lesen/Schreiben von Daten in der Artikeldatenbank
- Mehr verfügbare Actions

# 1.1 Toolbars – der erste Schritt zum Script

## 1.1.1 Toolbars anpassen

In EPLAN gibt es die Möglichkeit, eigene Toolbars zu erstellen. Doch was hat das mit Scripting zu tun? Ein Script ist eigentlich eine Erweiterung der Funktionalität einer Symbolleiste. In einer benutzerdefinierten Toolbar können vordefinierte Befehle ausgeführt werden. Dies sind alle von EPLAN offiziell unterstützten Actions, welche zudem in der Hilfe dokumentiert sind. Einen Verweis zur Hilfe finden Sie in Kapitel 5. Diese Befehle werden auch in einem Script verwendet. Ein Vorteil des Scripts gegenüber der klassischen Toolbar ist, dass mehrere Actions ausgeführt werden können. In der Symbolleiste müsste man mehrere Schaltflächen erstellen, um zum gleichen Ergebnis zu kommen. Bei der Menge an Möglichkeiten wird der Arbeitsbereich schnell unübersichtlich.

Im Folgenden wollen wir eine neue Toolbar erstellen. Dazu führen wir einen Rechtsklick auf die grafische Oberfläche aus und wählen den Punkt Anpassen... im Kontextmenü an (Bild 1.1).

# 2 Scriptfunktionen

## 2.1 Attribute

Ein Script kann, wie wir schon in Kapitel 1 erfahren haben, verschiedenste Funktionen ausführen. Darum müssen Sie im Programmcode hinterlegen, welche Funktion das Script bereitstellt. Es können auch mehrere Actions in einem Script vorhanden sein. Sie können z. B. eine Beschriftung automatisch erzeugen lassen. Außerdem kann für diese Action ein eigener Menüpunkt in der Menüleiste erzeugt werden. Dies ist alles innerhalb einer Datei möglich. In diesem Fall macht das auch Sinn, da der Menüpunkt nicht ohne die dazugehörige Action ausgeführt werden kann. Es macht aber keinen Sinn, alle Actions in eine Datei zu packen, da es so schnell unübersichtlich wird.

Für unser erstes Beispiel verwenden wir das Attribut `[Start]`. Das ist der schnellste Weg, mehrere Scripte zu testen. Oftmals stellt sich die Frage, wann ein Script geladen und was ausgeführt werden muss. Über die Attribute können Sie dies schnell und einfach herausfinden. Dies ist eine Auflistung der Attribute fürs Ausführen und Laden:

- Ausführen:
  - `[Start]`
- Laden:
  - `[DeclareAction]`
  - `[DeclareEventHandler]`
  - `[DeclareMenu]`
  - `[DeclareRegister]`
  - `[DeclareUnregister]`

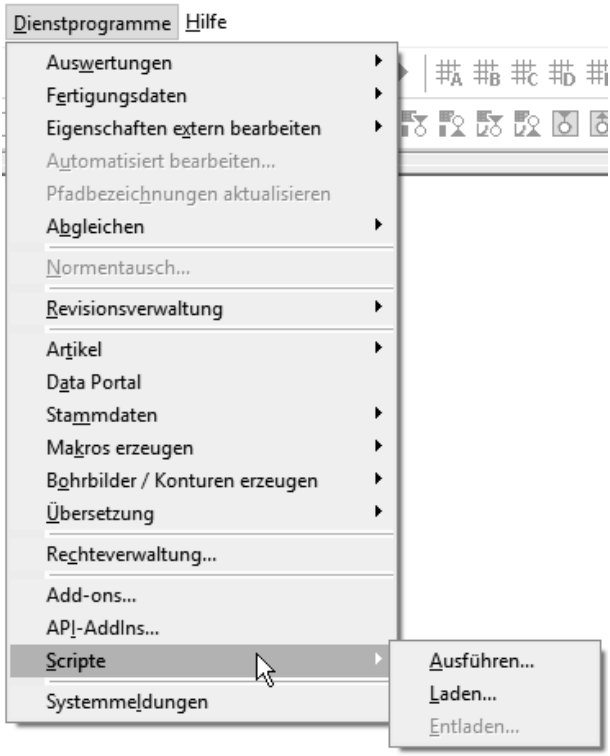



**Bild 2.1** Scripte ausführen/laden/entladen

### 2.1.1 Start

Das Attribut `[Start]` wird meistens verwendet, wenn man ein Script nur einmal oder eher selten benötigt. Sie müssen bei jedem Starten der Action über die Menüleiste gehen, um das Script auszuführen.

In der Entwicklungsumgebung wechseln Sie ins Codefenster und betrachten die ersten Zeilen. Führen Sie hierfür einen Doppelklick auf die Datei *01_Start.cs* aus.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

Mit dem Befehl `using` beschreiben Sie, welche Verweise verwendet werden sollen. In einer sogenannten Using-Direktive werden bestimmte Programmbefehle ge-

speichert und können, sofern sie geladen sind, abgerufen werden. Für das erste Script sind folgende Verweise erforderlich:

- *System.Windows.Forms*
- *Eplan.EplApi.Scripting*

Es werden nicht immer alle Verweise benötigt. Dennoch ist es ratsam, eine Vorlage zu erstellen, die alle für Sie wichtigen Using-Direktiven enthält. Später können Sie nicht benötigte Elemente entfernen. Folgende Namespaces werden im Scripting häufig verwendet:

- *Eplan.EplApi.ApplicationFramework*
- *Eplan.EplApi.Base*
- *Eplan.EplApi.Scripting*

Beim Tippen des Codes erscheint ein kleiner Dialog mit verschiedenen Begriffen (Bild 2.2). Diesen Dialog nennt man *IntelliSense*. Es ist eine Möglichkeit zur Wortvervollständigung einzelner Begriffe in der Programmierung. Das von Microsoft entwickelte Hilfsmittel ist in fast allen Entwicklungsumgebungen für sämtliche Programmiersprachen vorhanden. Wenn Sie einen Begriff eingeben, werden übereinstimmende Ergebnisse angezeigt. Sie können nun mit den Pfeiltasten auf der Tastatur den gewünschten Begriff auswählen. Um das Getippte zu vervollständigen, drücken Sie auf die **Tab**-Taste. Ist der *IntelliSense* nicht sichtbar, kann der Dialog mit dem Tastatur-Shortcut **Strg** + **Leertaste** wieder eingeblendet werden.

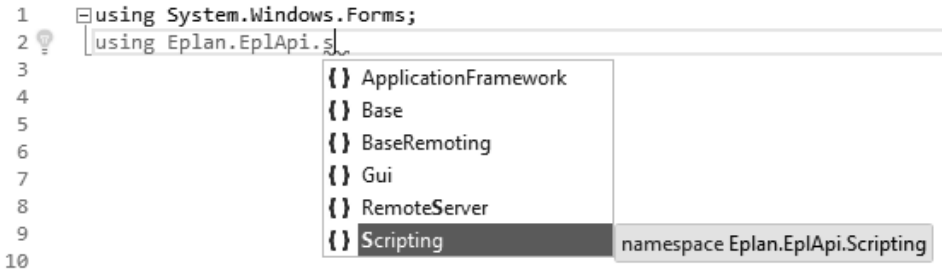



**Bild 2.2** IntelliSense

Eine Code-Folge wird in C# immer mit einem Semikolon (**;**) beendet. Wer schon einmal in Visual Basic programmiert hat, weiß, dass es dort der Zeilenumbruch ist. In C# können Sie beliebig viele Zeilenumbrüche im Code einfügen.

Den Aufbau eines Programmcodes in C# können Sie sich vorstellen wie eine Zwiebel (Bild 2.3). Es gibt verschiedene Schichten, die mit geschweiften Klammern eingegrenzt werden. Der Aufbau ist allerdings nicht so kompliziert, dass die Zwiebel Sie zum Weinen bringen wird.

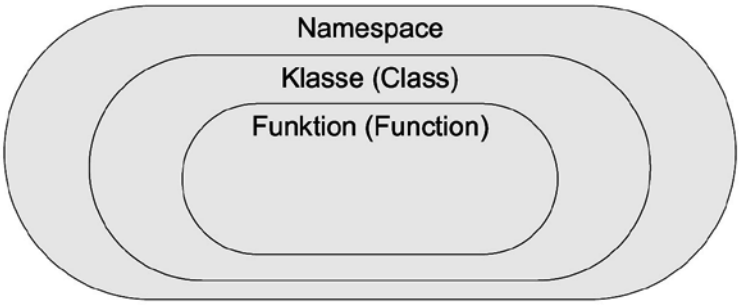



**Bild 2.3** Aufbau des Programmcodes in C#

Die „äußere Schale“ ist der Namespace:

```
namespace Namespace
{
}
```

Die „mittlere Schale“ ist die Klasse (Class):

```
class Class
{
}
```

In einer Klasse muss mindestens eine Funktion vorhanden sein. Man spricht hier von Methoden. Methoden werden mit `void` gekennzeichnet. Ergänzen Sie den Programmcode um folgende Zeilen:

```
void Function()
{
}
```

In der Funktion steht nun der eigentliche Programmcode (*Was soll EPLAN tun?*).

Wir lassen eine MessageBox anzeigen, um festzustellen, ob unser Script ausgeführt wurde. Dies ist der Aufbau des Befehls in kurzen Worten:

- Was soll ich machen? → `MessageBox`
- Was soll ich damit anstellen? → `Show` (anzeigen)
- Was soll ich anzeigen? → *Ich kann scripten!*

Die einzelnen Klassen und Methoden werden durch einen Punkt getrennt. Stellen Sie sich hierzu eine Treppe vor. Auf jeder Ebene befinden sich andere Räume mit unterschiedlichen Funktionen. Zu Beginn findet man oft nicht sofort die gewünschte Funktion, aber mit der Zeit versteht man die Struktur des .NET Frameworks.

Beispielaufbau eines Methodenaufrufs:

```
Ebene0.Ebene1.Ebene2.Methode();
```

Nehmen wir als Beispiel eine Aktivität an einem Ort, müsste dies wie folgt angegeben werden:

```
München.Theresienwiese.Oktoberfest.Achterbahnfahren();
```

Text steht im Programmcode immer in Hochkommas (`"Anführungszeichen"`):

```
München.Bahnhof.Schaffner.Ruft("Alles einsteigen!");
```

Kommentare können mit zwei Schrägstrichen eingefügt werden:

```
// Ich bin ein Kommentar
```

Dadurch weiß das Programm, dass es sich um einen Kommentar handelt, der nicht abgearbeitet werden muss.

In Visual Studio gibt es fertige Buttons  für das Einfügen von Kommentaren. Ist diese Toolbar nicht sichtbar, können Sie diese, wie in Bild 2.4 dargestellt, einblenden. Es wird dann die komplette Zeile, welche gerade aktiv ist, ein- bzw. auskommentiert.

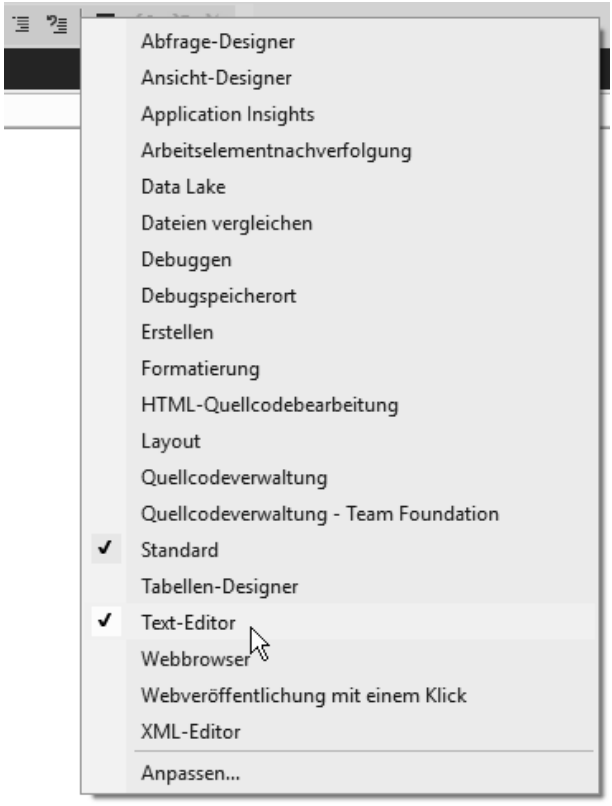



**Bild 2.4** Toolbar *Text-Editor* einblenden

Um eine MessageBox anzuzeigen, ist folgender Code notwendig:

```
MessageBox.Show("Ich kann scripten!"); // Kommentar
```

Eine Funktion wird mit `return` (zurückkehren) abgeschlossen. Dies ist nicht zwingend notwendig, wenn jeder Teil des Programmcodes Werte zurückgibt.

```
return;
```

Nicht verwendete Verweise sollten Sie entfernen, um das Starten des Scripts zu beschleunigen. Durch einen rechten Mausklick und das Ausführen der Funktion USING-DIREKTIVEN ORGANISIEREN > ENTFERNEN UND SORTIEREN in Visual Studio geschieht dies automatisch (Bild 2.5).

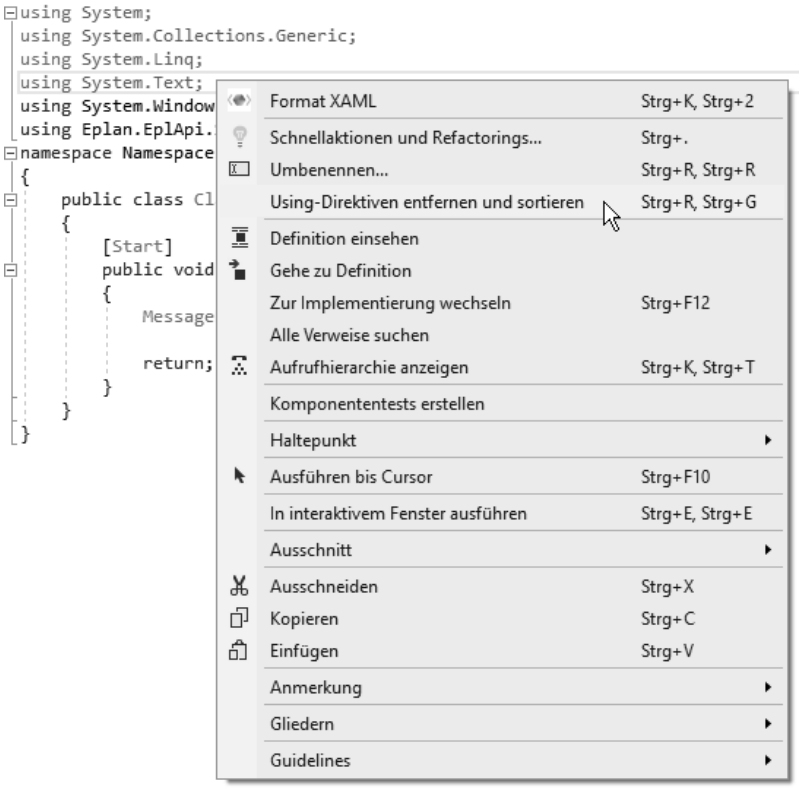



**Bild 2.5** Using-Direktiven organisieren

Nun setzen wir unsere „Schalen“ zusammen. Das komplette Beispiel ist immer am Ende einer Lektion zu finden:

```
using Eplan.EplApi.Scripting;
using System.Windows.Forms;

namespace Namespace
{
  public class Class
  {
    [Start]
    public void Function()
    {
      MessageBox.Show("Ich kann scripten!"); // Kommentar
      return;
    }
  }
}
```

Wechseln Sie nun nach EPLAN und testen Sie das Script über Dienstprogramme > Scripte > Ausführen .... Im darauffolgenden Dialog wählen Sie das Script aus. Die Scriptdatei befindet sich im Ordner Ihres Visual-Studio-Projekts (Bild 2.6).

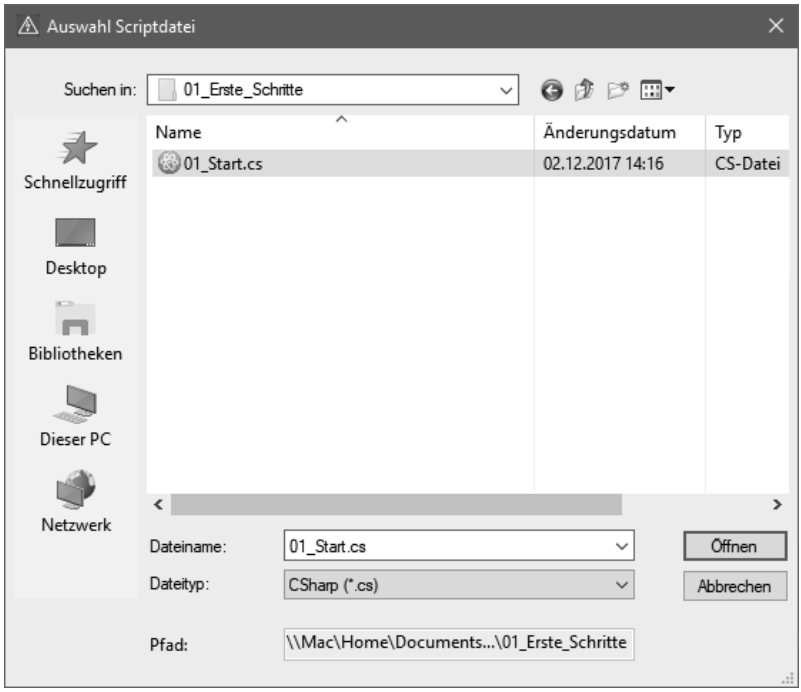



**Bild 2.6** Script ausführen

Wurde der Programmcode fehlerfrei ausgeführt, sehen Sie den in Bild 2.7 dargestellten Dialog.

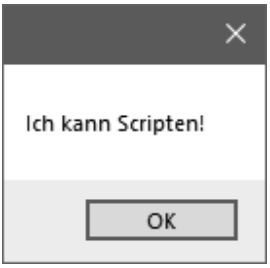



**Bild 2.7** MessageBox

Beim Schreiben des Programmcodes können mehrere Fehler passieren. Kommentieren Sie testweise die Initialisierung der Using-Direktiven aus:

```
//using System.Windows.Forms;
//using Eplan.EplApi.Scripting;
```

Vielleicht ist Ihnen beim Auskommentieren der Using-Anweisungen aufgefallen, dass im Fehlerfenster von Visual Studio einige Fehler hinzugekommen sind. Dies resultiert aus der Tatsache, dass die Entwicklungsumgebung die Ausdrücke aufgrund fehlender Verweise nicht finden kann. Diese Fehlerliste ist ähnlich wie die Meldungsverwaltung von EPLAN aufgebaut (Bild 2.8). Hier wird ebenfalls unterschieden zwischen:

- Fehlern
- Warnungen
- Mitteilungen

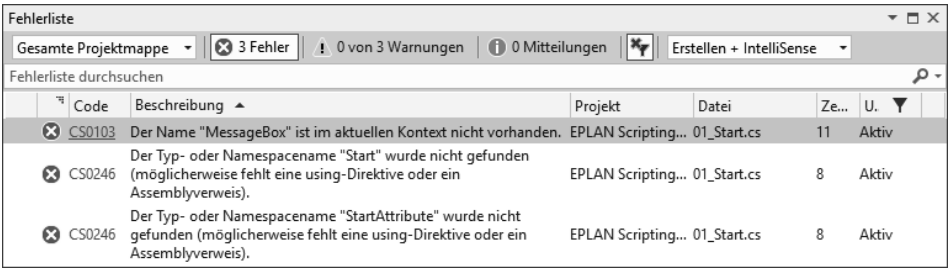



**Bild 2.8** Fehlerliste

Ist das Fenster aus Bild 2.8 nicht sichtbar, können Sie es über Ansicht > Fehlerliste einblenden (Bild 2.9).

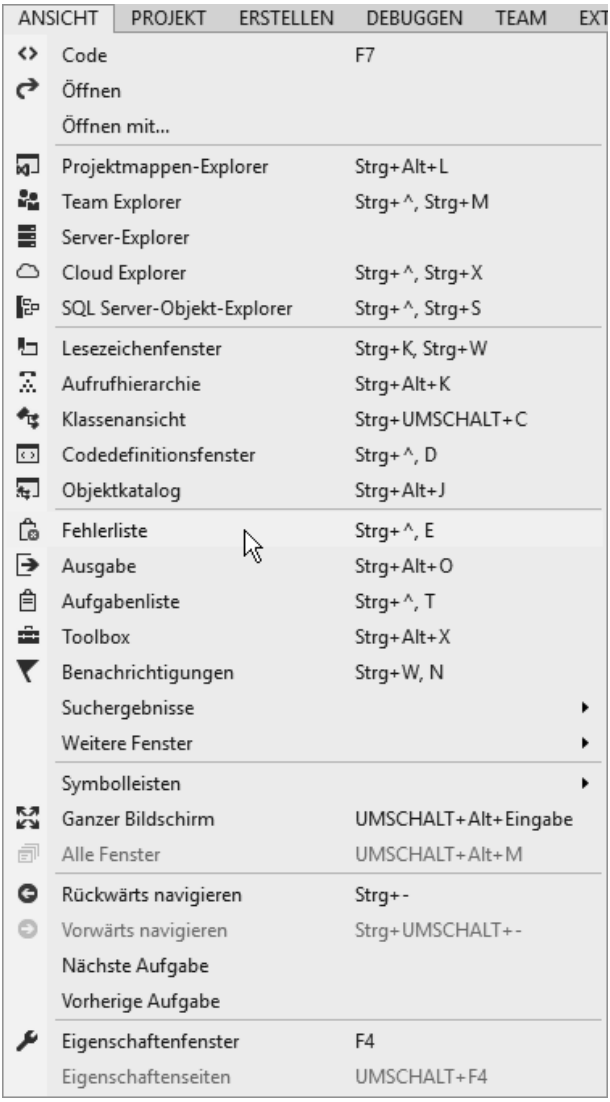



**Bild 2.9** Fehlerliste einblenden

### Fehler: Using-Direktive

Es kann sein, dass Fehler in Visual Studio nicht auftreten, jedoch beim Ausführen/Starten des Scripts in EPLAN gefunden werden. Ist dies der Fall, werden sie in den Systemmeldungen angezeigt (Bild 2.10). EPLAN beachtet dabei nicht, welche Using-Direktiven intern schon bekannt waren. Ist ein anderer Fehler im Script vorhanden, werden diese (fehlerhafterweise) auch angezeigt, können aber guten Gewissens ignoriert werden.

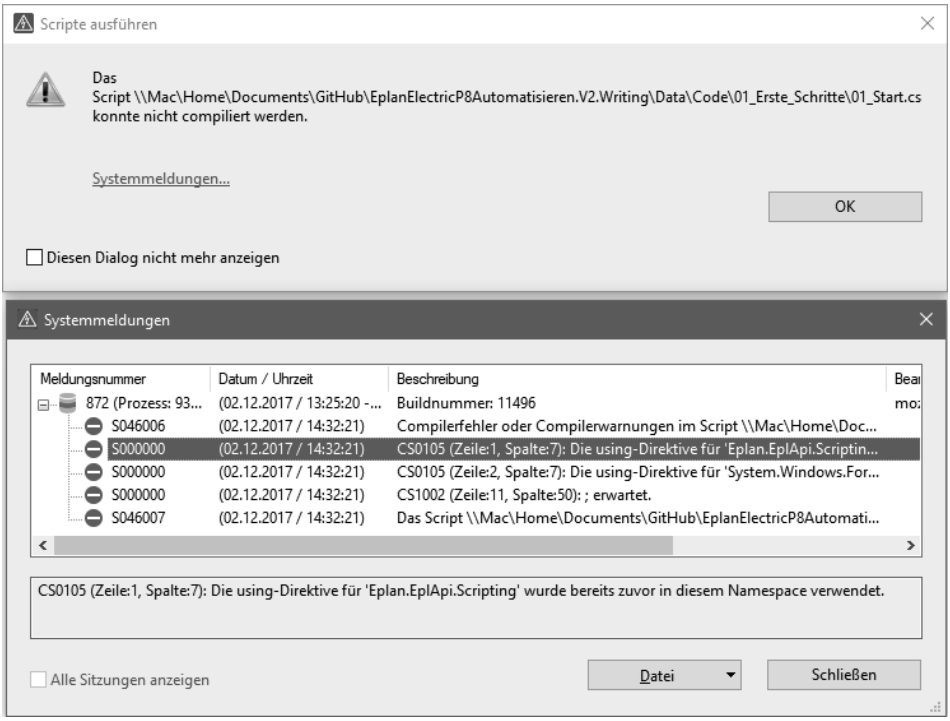



**Bild 2.10** Fehler: Using-Direktive wurde bereits zuvor verwendet.

In den Systemmeldungen in Bild 2.10 werden folgende Fehler angezeigt:

1. Compilerfehler: Information, dass das Script einen Fehler verursacht hat
2. Die Using-Direktive wurde bereits zuvor in diesem Namespace verwendet.
3. Die Using-Direktive wurde bereits zuvor in diesem Namespace verwendet.
4. Semikolon erwartet: Dies ist der eigentliche Fehler, den Sie als einzigen im Script korrigieren müssen. Danach wird das Script korrekt ausgeführt.
5. Das Script konnte nicht compiliert werden.

Um eventuell auftretende Fehler schnell zu finden, können Sie über die *Optionen* in Visual Studio auch die Zeilennummern einblenden, falls diese noch nicht sichtbar sind (Bild 2.11 und Bild 2.12).

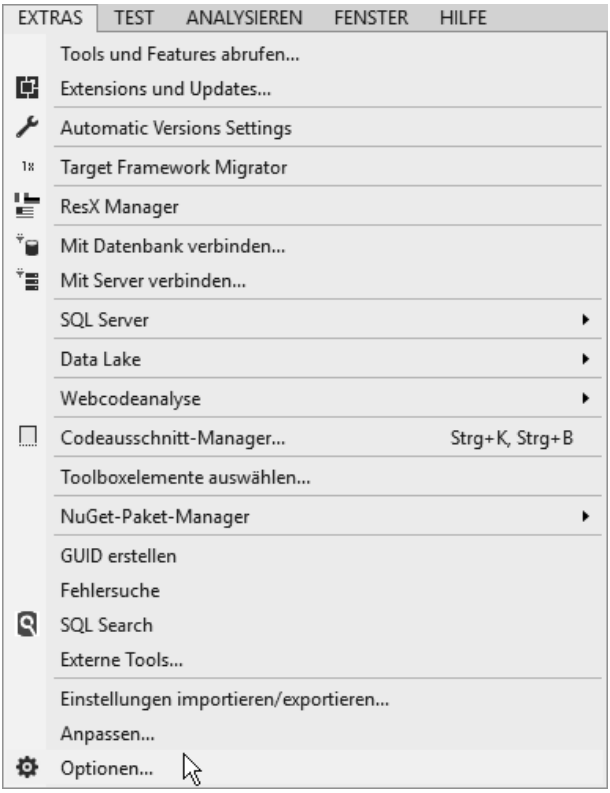



**Bild 2.11** Optionen in Visual Studio

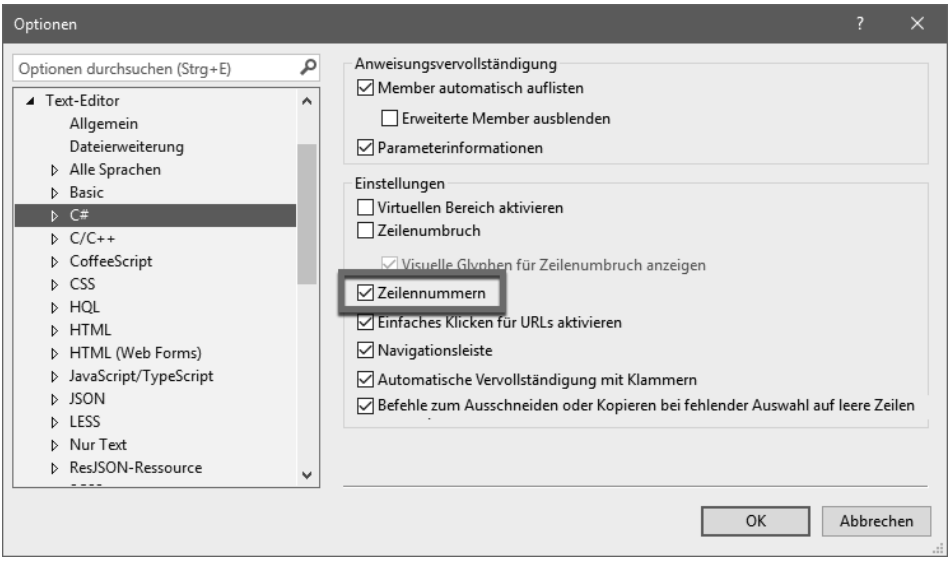



**Bild 2.12** Anzeigen der Zeilennummern aktivieren

### Semikolon erwartet

Gerade am Anfang vergisst man öfter mal, den Code mit einem Semikolon abzuschließen. Wenn Sie die Fehlerliste eingeblendet haben, können Sie dies schon vor dem Ausführen in EPLAN sehen. Im Code-Fenster wird die Stelle rot unterstrichen (Bild 2.13).

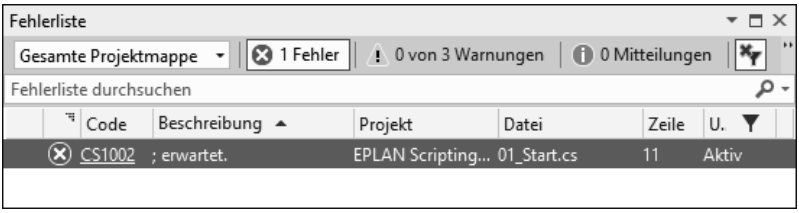



**Bild 2.13** Visual Studio: Semikolon erwartet

Auch EPLAN überprüft den Code, bevor dieser geladen oder ausgeführt wird (Bild 2.14).

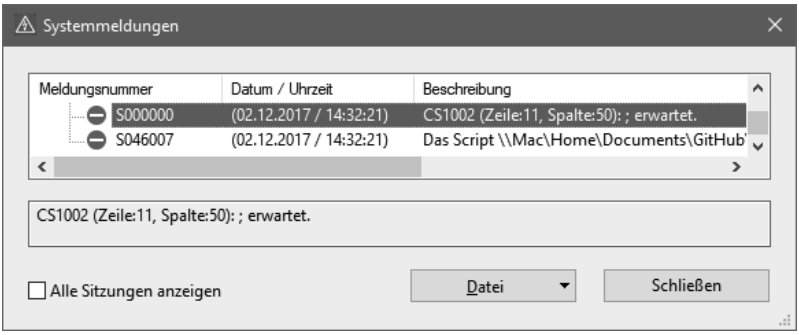



**Bild 2.14** EPLAN: Semikolon erwartet

### Scriptdatei nicht gefunden

Wenn Sie auf Dateiebene den Dateinamen eines Scripts ändern oder es löschen, weiß EPLAN davon nichts. Darum bekommen Sie in diesem Fall auch einen Fehler in den Systemmeldungen angezeigt (Bild 2.15). Bei jedem Start von EPLAN werden die Scripte überprüft. Das Script muss entladen und neu geladen werden. Dies ist nur notwendig, wenn ein Script geladen wurde (wie in Abschnitt 2.1.2, „DeclareAction“).

Bei bestimmten Projekteinstellungen kann es dazu kommen, dass diese zwar gesetzt werden, EPLAN diese jedoch nicht als Änderung feststellt. Tritt dieses Fehlverhalten z. B. beim Ändern des Normblatts auf, müssen Sie folgenden Programmcode hinzufügen:

```
new EventManager().Send("PageManagement.ProjectSettings.Changed", new
 EventParameterString());
```

## 2.6 Menüs

Wie bereits zu Beginn des Buches beschrieben, können in EPLAN eigene Menüs erzeugt werden. Es gibt mehrere Arten von Menütypen, welche unterschiedliche Funktionen haben:

- Menüpunkt in Dienstprogrammen
- Menüpunkt in einem bestehenden Menü
- Hauptmenü mit einem Menüpunkt
- Popup-Menü

Jedem neu hinzugefügten Menüpunkt muss eine Action zugewiesen werden. Dies kann eine EPLAN-Action oder eine eigene Action sein. In den folgenden Beispielen werden wir immer die Action mit dem Namen *MenuAction* verwenden. Es darf kein anderes Script mit dieser Action vorhanden sein, wenn das Script in EPLAN geladen wird.

Wird ein Menü-Script entladen, wird das Menü nicht immer entfernt. Auch bei Änderungen werden diese oft nicht übernommen, oder das Menü enthält doppelte Einträge. Tritt eines dieser Probleme auf, müssen Sie EPLAN neu starten.

### 2.6.1 Menüpunkt in Dienstprogramme

Erstellen Sie ein neues Script mit dem Namen *01_Menüpunkt_in_Dienstprogramme.cs* und zusätzlich einen neuen Ordner für das Abschnitt Menüs (*06_Menüs*). Als Attribute benötigen wir `[DeclareAction("MenuAction")]` für die Action und `[DeclareMenu]` für den dazugehörigen Menüpunkt.

Unsere Action soll nur eine MessageBox mit dem Text *Action wurde ausgeführt!* anzeigen, um die Funktionalität zu testen. Erstellen Sie die folgenden zwei Funktionen:

- `ActionFunction()`: Hier wird die Action für das Menü bereitgestellt.
- `MenuFunction()`: Hier wird das Menü erzeugt und die Action aufgerufen.

```
[DeclareAction("MenuAction")]
public void ActionFunction()
{
  MessageBox.Show("Action wurde ausgeführt!");

  return;
}
```

In der neuen Funktion `MenuFunction()` erstellen wir nun ein Objekt der Klasse `Menu`, das im EPLAN-Namespace zu finden ist:

```
Eplan.EplApi.Gui.Menu menu = new Eplan.EplApi.Gui.Menu();
```

Es ist notwendig, den kompletten Namespace zur Klasse zu hinterlegen, da es z. B. in *System.Windows.Forms* auch eine Klasse `Menu` gibt.

Um einen einfachen Menüpunkt unter dem Dienstprogramm zu erstellen, nutzen wir die Methode `AddMenuItem()`. Dadurch wird das Menü *Dienstprogramme* durch den angegebenen Menüpunkt ergänzt. Die Methode besitzt folgende Eigenschaften:

- *Name Menüpunkt*
- *Name Action*

Als Actionnamen tragen Sie die zuvor initialisierte `MenuAction` ein:

```
menu.AddMenuItem(
    "Menüpunkt am Ende von Dienstprogramme", // Name: Menüpunkt
    "MenuAction" // Name: Action
    );
```

Sie können diesen Programmcode auch in einer Zeile schreiben. Wegen der besseren Lesbarkeit habe ich den Code hier aufgeteilt.

**HINWEIS:** Der Action können auch gleich die Parameter mitgegeben werden (ähnlich wie bei einer Toolbar-Befehlszeile).

Da bei späteren Methoden nicht immer klar ist, für was eine Eigenschaft zuständig ist, füge ich die Beschreibung immer als Kommentar hinzu.

Wie schon beschrieben, ist es bei Menü-Scripten wichtig, diese zu laden. Haben Sie alles richtig gemacht, erscheint ein neuer Menüpunkt unter *Dienstprogramme* (Bild 2.99).

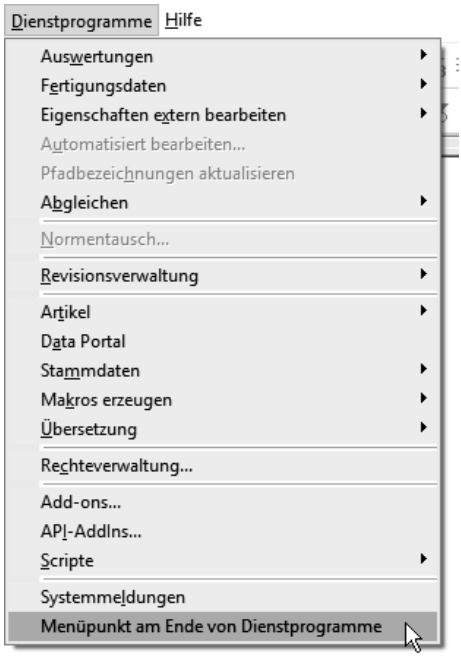



**Bild 2.99** Neuer Menüpunkt unter *Dienstprogramme*

Das komplette Script sieht wie folgt aus:

```
using System.Windows.Forms;
using Eplan.EplApi.Scripting;

public class _06_Menues_01_Menuepunkt_in_Dienstprogramme
{
  [DeclareAction("MenuAction")]
  public void ActionFunction()
  {
    MessageBox.Show("Action wurde ausgeführt!");

    return;
  }

  [DeclareMenu]
  public void MenuFunction()
  {
    Eplan.EplApi.Gui.Menu menu = new Eplan.EplApi.Gui.Menu();

    menu.AddMenuItem(
        "Menüpunkt am Ende von Dienstprogramme", // Name: Menüpunkt
        "MenuAction" // Name: Action
        );

    return;
  }
}
```

### 2.6.2 Bestehendes Menü erweitern

Für die zweite Variante der Menü-Klasse kopieren Sie das erste Script und vergeben den Namen *02_Bestehendes_Menü_erweitern.cs*. Lassen Sie die erste Funktion unberührt, und verändern Sie nur die Methode des Menü-Objekts.

Damit ein bestehendes Menü erweitert werden kann, steht die Methode `AddMenuItem()` bereit. Folgende Eigenschaften stehen zur Verfügung:

- *Name Menüpunkt*
- *Name Action*
- *Statustext*
- *Menü-ID*
- *Hinter/vor Menüpunkt*

Der Statustext wird in der Fußzeile von EPLAN angezeigt und kann eine längere Beschreibung enthalten (Bild 2.100).

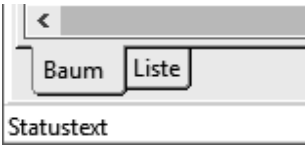



**Bild 2.100** Statustext

Die Menü-ID (*Integer*) dient als Referenz-Menüpunkt, um die Position festzulegen. Über den Diagnose-Dialog (**Strg** + **^**) können wir nach dem Aufrufen des Menüpunkts die Menü-ID herausfinden. Führen Sie dazu zuerst EINFÜGEN > FENSTERMAKRO aus und öffnen dann den Diagnose-Dialog (Bild 2.101).

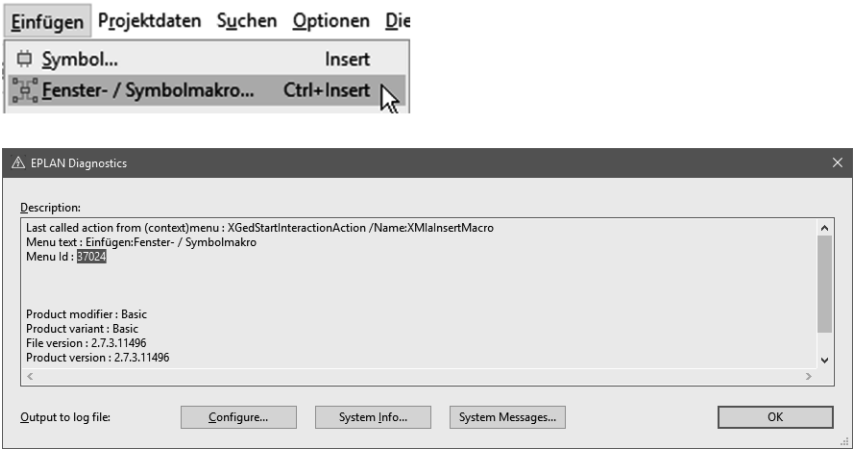



**Bild 2.101** Menü-ID über den Diagnose-Dialog herausfinden

Mit der letzten Eigenschaft können Sie festlegen, ob der neue Menüpunkt vor oder hinter dem Referenz-Menüpunkt eingefügt werden soll:

- 1: hinter Menüpunkt
- 0: vor Menüpunkt

Die beiden letzten Überladungen „Separator davor/dahinter“ zeigen eine Trennlinie im Menü an. Dieser boolesche Wert hat die bekannten Zustände *true* oder *false*.

Ändern Sie das Script wie folgt:

```
using System.Windows.Forms;
using Eplan.EplApi.Scripting;

public class _06_Menues_02_Bestehendes_Menue_erweitern
{
  [DeclareAction("MenuAction")]
  public void ActionFunction()
  {
    MessageBox.Show("Action wurde ausgeführt!");

    return;
  }

  [DeclareMenu]
  public void MenuFunction()
  {
    Eplan.EplApi.Gui.Menu menu = new Eplan.EplApi.Gui.Menu();

    menu.AddMenuItem(
        "Bestehendes Menü erweitern", // Name: Menüpunkt
        "MenuAction", // Name: Action
        "Statustext", // Statustext
        37024, // Menü-ID: Einfügen/Fenstermakro…
        1, // 1 = hinter Menüpunkt, 0 = vor Menüpunkt
        false, // Separator davor anzeigen
        false // Separator dahinter anzeigen
        );

    return;
  }
}
```

Stellen Sie sicher, dass kein Script geladen ist, und laden Sie die neue Datei. Nach dem Eintrag EINFÜGEN > FENSTERMAKRO erscheint nun ein neuer Untermenüpunkt (Bild 2.102).

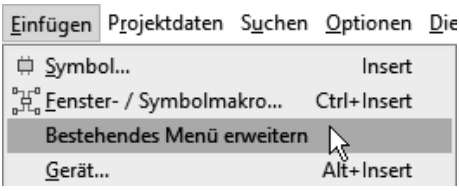



**Bild 2.102** Bestehendes Menü erweitern

### 2.6.3 Hauptmenü mit einem Untermenüpunkt

Kopieren Sie das Script erneut und vergeben den Namen *03_Hauptmenü_mit_Untermenüpunkt.cs*. Die erste Methode lassen Sie unverändert, da die Action erneut ausgeführt wird. In diesem Beispiel erzeugen Sie ein neues Hauptmenü in der EPLAN-Oberfläche. Zusätzlich müssen Sie diesem Menü einen Menüpunkt hinzufügen.

Dafür verwenden Sie die Methode `AddMainMenu()` mit folgenden Eigenschaften:

- *Name Menü*
- *Neben Menüpunkt*
- *Name Menüpunkt*
- *Name Action*
- *Statustext*
- *Hinter/vor Menüpunkt*

Mit *neben Menüpunkt* geben Sie an, an welcher Stelle das Menü eingefügt wird. Es handelt sich hier um eine sogenannte Enumeration, welche immer eine feste Auflistung ist. Alternativ können Sie einen String angeben, der den Menünamen trägt.

Zusätzlich können Sie festlegen, ob das Menü davor oder dahinter platziert werden soll:

- hinter Hauptmenüpunkt: 1
- vor Hauptmenüpunkt: 0

Möchten Sie z. B. das Menü an erster Stelle erzeugen, müssten Sie „Projekt“ und „0“ angeben.

Es ist allerdings ratsam, die Reihenfolge der Grundeinstellung von EPLAN nicht zu verändern. Sonst finden sich andere User meist nicht so schnell zurecht. Aus diesem Grund füge ich neue Menüs immer hinter dem Menü *Hilfe* ein.

```
menu.AddMainMenu(
    "Menü 1", // Name: Menü
    Eplan.EplApi.Gui.Menu.MainMenuName.eMainMenuHelp, // neben Menüpunkt
    "Hauptmenü mit einem Menüpunkt", // Name: Menüpunkt
    "MenuAction", // Name: Action
    "Statustext", // Statustext
    1 // 1 = hinter Menüpunkt, 0 = vor Menüpunkt
    );
```

Stellen Sie sicher, dass kein Script geladen ist, und laden Sie die neue Datei. Nun erscheint ein neues Hauptmenü mit dem Namen *Menü 1* auf der Oberfläche (Bild 2.103).

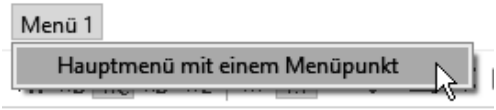



**Bild 2.103** Eigenes Hauptmenü

Das komplette Script sieht so aus:

```
using System.Windows.Forms;
using Eplan.EplApi.Scripting;

public class _06_Menues_02_Bestehendes_Menue_erweitern
{
  [DeclareAction("MenuAction")]
  public void ActionFunction()
  {
    MessageBox.Show("Action wurde ausgeführt!");

    return;
  }

  [DeclareMenu]
  public void MenuFunction()
  {
    Eplan.EplApi.Gui.Menu menu = new Eplan.EplApi.Gui.Menu();

    menu.AddMenuItem(
        "Bestehendes Menü erweitern", // Name: Menüpunkt
        "MenuAction", // Name: Action
        "Statustext", // Statustext
        37024, // Menü-ID: Einfügen/Fenstermakro…
        1, // 1 = hinter Menüpunkt, 0 = vor Menüpunkt
        false, // Separator davor anzeigen
        false // Separator dahinter anzeigen
        );

    return;
  }
}
```

### 2.6.4 Bestehendes Menü mit Popup-Menü erweitern

Kopieren Sie das erste Beispiel aus Abschnitt 2.6, „Menüs“. Vergeben Sie den Namen *04_Bestehendes_Menü_mit_Popup-Menü_erweitern.cs*. EPLAN verfügt über die Möglichkeit, ein Popup-Menü zu erstellen. Dies ist ein geschachteltes Steuerelement. Es können mehrere Menüpunkte hinter einem Popup-Menü erstellt werden.

Das Popup-Menü funktioniert ähnlich wie die Erweiterung eines bestehenden Menüs. Es kommt jedoch der Name des Menüs als Überladung hinzu. Im folgenden Beispiel wird ein Popup-Menü mithilfe der Methode `AddPopupMenuItem()` unter *Einfügen* erstellt (Bild 2.104):

```
using System.Windows.Forms;
using Eplan.EplApi.Scripting;

public class _06_Menues_04_Bestehendes_Menue_mit_Popup_Menue_erweitern
{
  [DeclareAction("MenuAction")]
```

# Index