TREIETSTRASSE 42
A-6832 SULZ, AUSTRIA
TEL. +43-5522/41600-0
FAX +43-5522/41600-6
www.team-electronics.com

Programmierrichtlinien für C, C++, C#

Standards, Regeln, Vereinbarungen

Autor Günter Marte

Stand 30. März 2009

Verwandte Dokumente -

1. Allgemeines

Dieses Dokument beschreibt die grundsätzlichen Standards, welche bei der Software-Erstellung und -Wartung berücksichtigt werden sollten. Diese Standards beziehen sich besonders auf die Programmierung in den Sprachen C, C++ und C# - die meisten Regeln gelten aber auch für alle anderen Programmiersprachen.

2. Dateien

2.1. Software-Bezeichnung

Jede Software, welche nicht nur zu kleinen Test- und Übungszwecken erstellt wird, erhält eine eindeutige Bezeichnung dieser Form:

S*iiii-vv*

wobei *iiii* die vierstellige Software-Identnummer und *vv* die zweistellige Versionsbezeichnung bzw. den Änderungsindex kennzeichnet.

Beispiel: S1234-01

Die resultierende Datei eines Softwareprojekts (exe, dll, bin...) sollte dieselbe Bezeichnung tragen, also z. B. S1234-01.exe. Dies ist aber nicht sinnvoll, wenn die Datei unabhängig von der Version ihren Namen nicht ändern darf was z. B. bei folgenden Szenario der Fall ist:

Auf einem PC einer Steuerung läuft die Visualisierungssoftware "hmi.exe". Auf dem Desktop existiert eine Verknüpfung zu diesem Programm und zudem wird dieses beim Start des Rechners per "Autostart" ebenfalls gestartet.
Würde nun dieses Programm statt "hmi.exe" "S1234-01.exe" heissen und wäre ein Update auf "S1234-02.exe" notwendig, so wäre die Desktop-Verknüpfung nicht mehr gültig und auch das automatische Starten nicht mehr möglich.
Eine Änderung dieser Einstellungen mag zwar für den Software-Entwickler am eigenen Rechner kein Problem sein, stellt aber einen Kunden an der weit entfernten Maschine vor ein unlösbares Problem.

Trotzdem muss aber auch in diesem Fall die ausführbare Datei "hmi.exe" eine Möglichkeit bieten, die exakte Software-Information zu liefern (Hilfe-Funktion...)

2.2. Speicherort

Jedes Softwarefile findet sich innerhalb folgender Verzeichnisstruktur:

Drive:\Software\S*hh00*_99\S*iiii-vv*\

hh bezeichnet dabei eine Hundertergruppe von Software-Identnummern.
Untergeordnete Verzeichnisse können beliebig (aber sinnvoll) strukturiert werden.

Beispiel: Das Sourcefile main.c aus dem Projekt S1234-01 kann sich an folgendem Speicherort befinden:

C:\Software\S1200_99\S1234-01\Source\main.c

Jede Software findet sich stets in allen Versionen auf dem Firmen-Server, welcher natürlich dieselbe Verzeichnisstruktur besitzt.

2.3. Version und Kompatibilität

Die Urversion jeder Software ist immer 00. Erfährt eine Software, welche schon irgendwo im Einsatz ist, eine Änderung, so muss die Software-Version um 1 erhöht werden. Beispiel: Aus S1234-00 wird dann S1234-01.

Für den Anwender einer Software (Kunde, Service-Personal...) kann eine neue Software-Version nur bedeuten:

- es wurden Fehler beseitigt
- die Software wurde erweitert oder verbessert

Es muss also in jedem Fall gewährleistet sein, dass eine neue Software-Version ohne Änderung der Systemumgebung (Hardware, Betriebssystem, Maschine...) lauffähig ist.

Beispiel:

Kunde 1 besitzt die Software S1234-00 für seine Maschine A.
Kunde 2 besitzt die Software S1234-01 für seine Maschinen B, C, und D.
Kunde 3 besitzt die Software S1234-02 für seine Maschinen E und F.

Wird nun eine neue Version S1234-03 erzeugt, so muss gewährleistet sein, dass alle Kunden diese Software verwenden können ohne an den Maschinen A..F auch nur die geringste Änderung vorzunehmen.
Kann dies nicht sichergestellt werden, so muss eine neue Software-Identnummer vergeben werden.

2.4. Info-File

Jede Software besitzt ein sogenanntes Info-File mit dem Namen S*iiii-vv*.inf, welche direkt im jeweiligen Software-Verzeichnis zu finden ist - also beispielsweise:

```
C:\Software\S1200_99\S1234-01\S1234-01.inf
```

Diese Datei gibt Auskunft über wichtige Informationen zur Software (Zweck, Autor, zugehörige Dokumente, Hardware-Einstellungen...) und die Änderungen, welche diese erfahren hat.

Beispiel (S0875-07.inf)

```
Folgeantriebe (Kuehlwalzen, Tiefdruck, Bahnzugregelung...) MOVEisto

Autor                        : Guenter Marte
Kunden                       : PAGO Aichtal, PAGO Grabs
Hardware                     : MOVEisto A961
Beschriftungsstreifen        : D0535
Blockschaltbild, I/O-Belegung: D0536
Bedienungsanleitung          : D0537
Maschinendaten, Parameter    : D0538
Inbetriebnahmeanleitung      : D0543

S0875-01, Aenderungen gegenueber S0875-00:    G. Marte, 31-10-2002
----------------------------------------

1. Die Dehnungseinstellung (siehe antrieb_kw in antrieb.c) arbeitet
   nun mit der Grunduebersetzung 20000:20000 (vorher 10000:10000).
   Dadurch kann die Dehnung auf 0.005% (vorher 0.01%) genau eingestellt
   werden.

.
.
.
weitere Änderungen...
```

3. Grundsätzliche Standards

3.1. Wartbarkeit, Lesbarkeit, Ästhetik

Bei der Erstellung von Software sollten immer folgende Faktoren in Erinnerung gerufen werden:

- Kann ich diesen Source-Code auch in einem Jahr noch verstehen?
- Ist meine Software ordentlich strukturiert/modularisiert oder gibt's haufenweise dubiose Querverbindungen zwischen den Modulen?
- Was passiert, wenn jemand anderer meinen Code betrachtet ("Code–Review") oder Änderungen vornehmen muss – wird er sich zurechtfinden?
- Erscheint meine Software aus "einem Guss" (konsequentes Anwenden von Kommentierungen, Einrückungen, Variablennamen...)?

3.2. Software-Aktualität

Wird eine Software erstellt oder geändert, so muss diese wie auch alle Dokumente und andere Unterlagen auf dem Server gespeichert werden. Nach einer eventuellen Software-Änderung ausser Hause und muss diese Software sofort bei der Rückkehr (und nicht erst Stunden später oder nie) auf den gemeinsamen Server.

3.3. Firmen-Name, Firmen-Logo

Wird in einer Software der Firmen-Name irgendwo verwendet (innerhalb eines Hilfe-Fensters o. ä.) so lautet dieser

TEAM ELECTRONICS

und nicht etwa "TEAM Electronics" oder "Team Electronics" oder gar "TEAM Elektronic".
Das Firmenlogo findet sich im Dokument D0393 – bitte keine Eigenkreationen ersinnen.

3.4. Sprache

Grundsätzlich orientieren sich viele Variablennamen an der englischen Sprache und die Kommentare sind in Deutsch verfasst, sofern nicht Kundenanforderungen dies verhindern.
Wie in der Programmierung üblich, ergibt sich oft ein Mix aus deutsch/englisch, was aber im Rahmen der natürlichen Empfindung zulässig ist. Es gelten allgemein die Regeln der neuen deutschen Rechtschreibung.

3.5. Dokumentation

Lästig, aber notwendig: Inbetriebnahmeanleitungen, Beschreibung von Funktionsbibilotheken, Bedienungsanleitungen, Berechnungs-Dokumentationen (Übersetzungsverhältnisse...) usw.
Verweise auf diese Dokumente finden sich im jeweiligen Info-File.

4. Programmierstandards

4.1. Bezeichnernamen

4.1.1. Notationen

Grundsätzlich existieren diese Notationen (Schreibstile) von Bezeichnern:

Notation	Bemerkung	Beispiel
Pascal	Der erste Buchstabe jeweils groß	MyIndetifier
Kamel	Am Anfang klein, danach jeweils der erste Buchstabe groß.	myIdentifier
Ungarisch	Wie Pascal, aber mit einem vorangestellten kleinengeschriebenen Typ-Präfix	nMyIdentifier
Groß	Alles groß, Worttrennung durch Unterstrich	MY_IDENTIFIER
Klein	Alles klein, Worttrennung durch Unterstrich	my_identifier

Je nach Bezeichnertyp (Methode, Konstante, Variable...) werden unterschiedliche Notationen angewendet. Die Notation "Klein" sollte nur noch bei Standard C-Programmen verwendet werden und keinesfalls in C++ oder C# - Code Einzug finden. Es gelten diese Notationen:

Bezeichner	Notation	Beispiel
Klasse	Pascal	`NumInput`
Variable	Ungarisch	`nTestVar`
Konstante, Read-Only-Variable	Groß	`ERR_EEPROM_CHKSUM`
Enumeration (Typ)	Pascal	`EColor`
Enumeration (Wert)	Groß oder Pascal	`RED_RAL3020, Red`
Enumeration (Instanz)	Ungarisch	`eColor`
Namespace	Pascal	`System.Drawing`
Methode, Funktion	Pascal	`ToString`
Property	Pascal	`BackColor`
Event-Handler	Ungarisch_Event	`btnTest_Click`
Interface	Pascal mit Präfix I	`ISerializable`
Exception (Klasse)	Pascal mit Suffix Exception	`DeviceException`

4.1.2. Typ-Präfixe

Je nach Typ einer Variablen oder Instanz einer Klasse werden diese mit einem bestimmten Präfix versehen. Ausnahme: Laufvariblen wie `i` in `for (int i = 0; ...)` können auf das Präfix verzichten.
Die folgende Liste gibt einen kurzen Überblick ohne Anspruch auf Vollständigkeit:

Typ	Präfix	Beispiel
int (Ganzahlen – auch long-, short- und unsigned-Derivate)	n	`int nTestVar` `unsigned long nMyCounter`
float	f	`float fTemperature`
double	d	`double dAcceleration`
string	str	`string strFileName`
bool	b	`bool bEnable`
Enumeration	e	`EColor eColor`
Array	a	`int[] anValue`
2-dimensionales Array	aa	`double[][] aadPosition`
Pointer	p	`double* pdValue`
Point	pt	`Drawing.Point ptStart`
Size	size	`Drawing.Size sizeRect`
Button	btn	`Forms.Button btnFileOpen`
Label	lbl	`Forms.Label lblValue`
TextBox	txt	`Forms.TextBox txtValue`
GroupBox	gb	`Forms.GroupBox gbValue`
CheckBox	cb	`Forms.CheckBox cbEnable`
RadioButton	rb	`Forms.RadioButton rbGreen`

4.1.3. Dominanz gemeinsamer Merkmale

Bei mehreren Bezeichnern ähnlicher Art sollten die gemeinsamen Merkmale vorangestellt werden.

Beispiel:

Für die Fehler "Kommunikationsfehler", "Antriebsfehler" und "Schutz offen" sollen 3 Konstanten definiert werden:

gut	schlecht

`const unsigned int ERR_COMM = 1;`
`const unsigned int ERR_DRIVE = 2;`
`const unsigned int ERR_SCHUTZ_OFFEN = 3;`
	`const unsigned int COMM_ERR = 1;`
	`const unsigned int DRIVE_ERR = 2;`
	`const unsigned int SCHUTZ_OFFEN = 3; // (*)`

Da alle 3 Konstanten etwas mit Fehlern zu tun haben, sollte das Kürzel ERR vorangestellt werden und nicht das Ende des Bezeichners bilden oder gar verschwinden wie in (*).

4.1.4. Das Links-Rechts-Vorne-Hinten-Problem

Speziell bei Maschinensteuerungen sind die Bezeichnungen "links", "rechts", "vorne" und "hinten" tunlichst zu vermeiden, da je nach Standpunkt ("vor" oder "hinter" der Maschine) aus links rechts werden kann oder umgekehrt. Lediglich "oben" und "unten" sind zulässige Ortsangaben, sofern sich die Maschine im Bereich der Erdanziehungskraft befindet.

Aus diesen Gründen existieren folgende gebräuchlichen Positions-/Richtungsangaben für materialverarbeitende Maschinen:

Position	Abkürzung (Groß)	Abkürzung (Pascal)	Bemerkung
Einlaufseite	ELS	Els	im herkömmlichen Sinn meist "links"
Auslaufseite	ALS	Als	im herkömmlichen Sinn meist "rechts"
Antriebsseite	ATS	Ats	im herkömmlichen Sinn meist "hinten"
Bedienseite	BDS	Bds	im herkömmlichen Sinn meist "vorne"
Oben	O	O	
Unten	U	U	

4.1.5. Bezeichnung logischer Zustände

Besonders bei der Namensvergabe für digitale I/Os stellt sich oft das Problem des logischen Zustands bei aktivem oder inaktivem Signal. Hier gilt die Regel: Ein-/Ausgänge werden nach ihrer Wirkung bei High-Signalpegel bezeichnet.

Klassisches Beispiel: Eine Maschine verfügt über eine Taste "Start" und eine Taste "Stop" wobei die Start-Taste ein Schliesser, die Stop-Taste aber ein Öffner ist. Der Maschinenstart erfolgt also bei aktivem Signal "Start", während die Maschine bei inaktivem Signal "Stop" anhalten soll.

Zustände, welche "aktiv low" sind, werden daher mit einem vorangestellten "n" gekennzeichnet.
Beispiele:

```
const unsigned int DI_START   = 1<<0; // Taste "Start"
const unsigned int DI_nSTOP   = 1<<1; // Taste "Stop"
const unsigned int DI_nNOTAUS = 1<<2; // Not-Aus-Kreis

const unsigned int DI_CWnCCW  = 1<<3; // High: Drehrichtung im Uhrzeigersinn
                                      //       (clockwise)
                                      // Low : Drehrichtung gegen Uhrzeigersinn
                                      //       (counterclockwise)
```

4.1.6. Felder, lokale Methodenvariablen

Variablen, welche innerhalb einer Klasse definiert sind (Felder) und daher "Membervariablen" dieser Klasse sind, sollten stets als `private` deklariert werden und erhalten das zusätzliche Präfix `m_`.
Auf diese Variablen kann von extern nur über Properties oder Get-/Set-Methoden zugegriffen werden.
Lokale Variablen innerhalb einer Methode und Konstanten erhalten kein solches Präfix.

Beispiel:

```
namespace Test
{
  class MyClass
  {
    // ###########################################################################
    // Felder (Membervariablen)
    // ###########################################################################

    private       int m_nVariable;
    private const int MAX_VARIABLE_VALUE = 1000;

    // ###########################################################################
    // Properties
    // ###########################################################################

    // ///////////////////////////////////////////////////////////////////////////
    // Variable

    public int Variable
    {
      get
      {
        return m_nVariable;
      }

      set
      {
        if (value < MAX_VARIABLE_VALUE)
          m_nVariable = value;
      }
    }

    // ###########################################################################
    // Methoden
    // ###########################################################################

    // ///////////////////////////////////////////////////////////////////////////
    // MyClass (Konstruktor)

    public MyClass (int nVariable)
    {
      m_nVariable = nVariable;
    }

    // ///////////////////////////////////////////////////////////////////////////
    // AdjustVariable (Variable ändern)

    public int AdjustVariable ()
    {
      int nX = 2; // lokale Variable ohne Präfix _m

      m_nVariable /= nX;
      return m_nVariable;
    }
  }
}
```

4.2. Code-Formatierung

4.2.1. Zeilenlänge, Sonderzeichen und Umlaute

Die Zeilenlänge (Breite des Codes) sollte maximal 100 Zeichen betragen. Üblich und historisch bedingt (DOS-Fenster, Nadeldrucker...) sind hingegen 80 Zeichen, was auch im Zeitalter hochauflösender Bildschirme eine sinnvolle Begrenzung darstellt. Sonderzeichen und Umlaute wie "ß", "@", "ä" oder "$" sind natürlich nur innerhalb von Strings oder Kommentaren zulässig und auch erlaubt.
Ausnahme: Info-Files und "alte C-Files" sollten keines dieser Zeichen verwenden, das diese oft mit alten Editoren bearbeitet werden, welche ein anderes Sonderzeichen-Format besitzen.

4.2.2. Einrückungen (Indents)

Einrückungen erfolgen in Schritten von 2 Leerzeichen, wobei diese nicht als Tabulatorzeichen definiert sein sollten. Dies kann in den meisten Entwicklungsumgebungen mit Einstellungen wie "Insert spaces for tabs" vorgenommen werden.

Beispiel:

```
namspace Test
{
  public class MyClass
  {
    private int Sum (int nA, int nB)
    {
      return (nA + nB);
    }
  }
}
```

und niemals so:

```
namspace Test
{
public class MyClass
{
    private int Sum (int nA, int nB)
      {
      return (nA + nB);
                      }
}
}
```

4.2.3. Verwendung der geschweiften Klammern

Die Zeichen { und } stehen immer als einzige Zeichen in einer Zeile. Zugehörige Klammernpaare stehen in derselben Spalte.

gut	schlecht

`if (nX > 0)`
`{`
`  for (int i = 0; i < 10; i++)`
`  {`
`  adValueX[i]++;`
`  adValueY[i]--:`
`  }`
`}`
	`if (nX > 0) {`
	`  for (int i = 0; i < 10; i++) {`
	`  adValueX[i]++;`
	`  adValueY[i]--: }`
	`}`

Code-Blöcke, welche nur aus einer einzigen Anweisung bestehen, dürfen auf die geschweiften Klammern verzichten:

Richtig	Ebenso richtig

`if (nX > 0)`
`  dValue = 1;`
	`if (nX > 0)`
	`{`
	`  dValue = 1;`
	`}`

4.2.4. Verwendung von Leerzeichen und Klammern

Leezeichen, Klammern und Aufteilungen in einzelne Zeilen fördern die Lesbarkeit von Anweisungen erheblich:

gut

```
for (
      int x = 0, int y = 100;         // x und y initialisieren
      (x < 10) && (adValueY[y] > 0); // Bedingung noch ok?
      x++, y--                        // x und y ändern
    )
{
  adValueX[x]+= y + 2;
  adValueY[y]-= y + 4;
}

if (
     (
       ( nDigitalInputs & (1<<0)) || // Eingang 0 aktiv
       (~nDigitalInputs & (1<<2))    // oder Eingang 2 nicht aktiv?
     )
     &&
     (nValue1 < nValue2)             // zudem nValue1 kleiner nValue2?
   )
{
  nDigitalOutputs = 0xFF;            // ja? -> alle Ausgänge setzen
}
```

schlecht

```
for(int x=0,int y=100;x<10&&adValueY[y]>0;x++,y--){
  adValueX[x]+=y+2; adValueY[y]-=y+4;}

if (nDigitalInputs & 1<<0 || ~nDigitalInputs & 1<<2 && nValue1 < nValue2){
  nDigitalOutputs = 0xFF;}
```

4.2.5. Deklarationen und Zuweisungszeichen in derselben Spalte

Typdeklarationen, Klasseninstanzen, Zuweisungszeichen usw. sollten für zusammenhängende Code-Abschnitte in derselben Spalte stehen.

gut	schlecht

`double dTestVar1;`
`float  fTestVar2;`
`long   nValue;`
`int    nValueLessOne;`
`bool   bEnable;`
 	
`dTestVar1     =   1.89;`
`fTestVar2     = -17.89;`
 	
`nValue        = (long)dTestVar1;`
`nValueLessOne = (int) nValue - 1;`
 	
`bEnable       = true;`
	`double dTestVar1;`
	`float fTestVar2;`
	`long nValue;`
	`int nValueLessOne;ß`
	`bool bEnable;`
	
	`dTestVar1 = 1.89;`
	`fTestVar2 = -17.89;`
	`nValue = (long)dTestVar1;`
	`nValueLessOne = nValue - 1;`
	`bEnable = true;`

4.3. Kommentare und Code-Strukturierung

4.3.1. Allgemeines

Kommentare und Kommentarzeichen dienen der Verständlichkeit und Strukturierung von Code. Kommentare sind insbesondere dort anzuwenden, wo eine spezielle Vorgangsweise dokumentiert werden soll. Sinnlose Kommentierungen wie

```
nTestVar++; // Testvariable inkrementieren
```

sind zu vermeiden, da sich diese Anweisung selbst schon ausreichend beschreibt.

4.3.2. Gruppierungs-Kommentare

Zur Strukturierung von Code wird dieser in zusammenhängende Konstrukte zusammegefasst – also alle Felddefinitionen, Enumerationen, Konstanten, Properties, Methoden, Event-Handler usw. gehören sinnvollerweise gruppiert und nicht wahllos verstreut, obwohl der Compiler dies meist zulassen würde.

Zur Erhöhung der Übersichtlichkeit dienen die sogenannten Gruppierungs-Kommentare, welche je nach Priorität mehr oder weniger "wuchtig" in Erscheinung treten.

Gruppierung	Beispiel

Zusammengehörende Konstrukte wie Enumerationen, Properties, Methoden, Event-Handler...	`// ###################################`
	`// Properties`
	`// ###################################`

Mitglied einer Gruppe (einzelne Methode, Property...)	`// /////////////////////////////////` `// BackColor`
Separierbare Gruppe innerhalb eines Gruppenmitglieds	`// ---- digitale Eingänge -----------`

4.3.3. Methoden-, Property-, Event-Header usw...

Zur besseren Übersichtlichkeit werden einzelne Methoden, Properties usw. mit einem Gruppierungskommentar voneinander getrennt. Dieser Kommentar beinhaltet zumindest den Namen der Methode, Property o. ä. selbst, kann aber beliebig erweitert werden, falls die Eigenschaft "selbsterklärend" nicht zutreffend sein sollte.

Gut	Schlecht

`// /////////////////////////////////`
`// SortList (alphabetische Sortierung)`
`//`
`// Parameter:`
`// bAtoZ : true  = von A nach Z`
`//         false = von Z nach A`
 	
`public bool SortList (bool bAtoZ)`
`{`
`  ...`
`}`
 	
`// /////////////////////////////////`
`// GetList`
 	
`public object GetList (int nIndex)`
`{`
`  ...`
`}`
	`public bool Sort (bool bAtoZ)`
	`{`
	`  ...`
	`}`
	
	`public object GetList (int nIndex)`
	`{`
	`  ...`
	`}`

4.4. Konstrukte

4.4.1. Enumerationen

Enumerationen erhalten das Präfix E. Die einzelnen Enumerationswerte werden je nach Typ entweder in den Notationen "Groß" oder "Pascal" definiert. "Groß" wird verwendet, wenn die Enumertationswerte den Charakter von Konstanten haben, ansonsten kann die Notation Pascal benutzt werden.

Instanzen von Enumerationen erhalten das Präfix e.

Beispiel:

Enumeration mit Konstanten	Wertunabhängige Enumeration

`public enum EDigitalInputs : ushort`
`{`
`  START   = 1<<0,`
`  nSTOP   = 1<<1,`
`  nNOTAUS = 1<<2,`
`}`
 	
`EDigitalInputs eDiBoard1;`
 	
`if ((eDiBoard1 & EDigitalInputs.START) != 0)`
`{`
`  // ...`
`}`
	`public enum EColor`
	`{`
	`  Red,`
	`  Blue,`
	`  Green,`
	`}`
	
	`EColor eColor;`
	
	`eColor = EColor.Red;`

5. Beispiel-Programm

```
using System;

namespace Test
{
  class Car
  {
    // ############################################################################
    // Enumerationen und Konstanten
    // ############################################################################

    // Wagenfarbe
    public enum EColor
    {
      Undefined,
      Red,
      Blue,
      Green,
      Yellow,
    }

    // Maximale Leistung [kW]
    private const double MAX_POWER = 250;


    // ############################################################################
    // Felder
    // ############################################################################

    private EColor m_eColor       = EColor.Undefined; // Wagenfarbe
    private double m_dPower       = 100;              // Leistung
    private string m_strNameOwner = "undefined";      // Besitzer


    // ############################################################################
    // Properties
    // ############################################################################

    // ////////////////////////////////////////////////////////////////////////////
    // Color

    public EColor Color
    {
      get
      {
        return m_eColor;
      }
      set
      {
        m_eColor = value;
      }
    }


    // ////////////////////////////////////////////////////////////////////////////
    // Power

    public double Power
    {
      get
      {
        return m_dPower;
      }
      set
      {
        if (value <= MAX_POWER)
          m_dPower = value;
      }
    }
```

```
    // ////////////////////////////////////////////////////////////////////////
    // Owner

    public string Owner
    {
      get
      {
        return m_strNameOwner;
      }
      set
      {
        m_strNameOwner = value;
      }
    }


    // ########################################################################
    // Methoden
    // ########################################################################

    // ////////////////////////////////////////////////////////////////////////
    // Car (Konstruktor)

    public Car()
    {
    }

    // ////////////////////////////////////////////////////////////////////////
    // RandomPower
    //
    // Leistung per Zufallsgenerator bestimmen

    public void RandomPower(double dMin)
    {
      Random r = new Random();
      double dPower;

      do
      {
        dPower = r.NextDouble() * MAX_POWER; // Leitung 0..MAX_POWER
      }
      while (dPower < dMin);                 // Minimale Leistung erreichen

      m_dPower = dPower;
    }


    // ////////////////////////////////////////////////////////////////////////
    // Main
    //
    // Hauptprogramm

    static void Main(string[] args)
    {
      // Das gelbe Auto von Tom...
      Car carTom   = new Car();

      carTom.Owner = "Tom";
      carTom.Color = Car.EColor.Yellow;
      carTom.RandomPower(50);
    }
  }
}
```