HOCHSCHULE
HAMM-LIPPSTADT

Programmierrichtlinie

für die Erstellung von Software in C, Matlab/Simulink, Python

von:	Prof. Dr. Mirek Göbel
	Prof. Dr. Ulrich Schneider
Stand:	6. Juli 2018

Inhaltsverzeichnis

Programmierrichtlinie für das Erstellen von C- und MATLAB-Code

1 Einleitung

Dieses Dokument ist ein Leitfaden für die Namenskonventionen beim Programmieren. Diese Konventionen sind abgeleitet von der ungarischen Notation und entsprechen der Hella Richtlinie HP-GE-232-12. Diese wurde für das zu bearbeitende Hochschulprojekt angepasst. In erster Linie beziehen sich die hier beschriebenen Konventionen auf C- und Matlab-Code sowie Simulink-Modelle.

2 Modul- und Funktionsheader

Für die Verständlichkeit der implementierten Software sind Kommentare und Beschreibungen von Modulen und Funktionen unerlässlich. Um einen schnellen Überblick über die vorliegende Implementierung zu erhalten, werden Module durch einen Modul-Header eingeleitet. Dieser befindet sich stets zu Beginn des Moduls. Selbiges gilt für die Beschreibung von Funktionen. Vorlagen für Modul- und Funktions-Header werden im Folgenden für verschiedene Programmiersprachen vorgestellt. Die Beispiele können aus diesem Dokument kopiert werden. Wichtig ist jedoch die gewissenhafte Änderung der Header-Einträge.

2.1 C und C++ Code

Ein Modul-Header für eine Implementierung in C oder C++ wird in Quelltext 1 gezeigt. Dieser wird eingeleitet vom Modulnamen, gefolgt vom Erstellungsdatum. Dieses Datum darf nach der erstmaligen Implementierung des Moduls nicht mehr verändert werden. In der Modulbeschreibung wird der Zweck des Moduls im Stil eines Titels genannt. Anschließend wird die verwendete Entwicklungsumgebung und der Name des Autors aufgeführt. Außerdem können Bemerkungen zum Modul hinzugefügt werden. Abschließend wird das Datum der letzten Änderung genannt.

Listing 1: Strukturierung eines Modul-Headers in C oder C++.

```
/************************************************************\
*
* Modul          : ModulName.c
*
* Datum          : 04. Oktober 2013
*
* Beschreibung   : Zweck dieses Moduls
*
* Implementierung : Visual Studio 2012 Professional
*
* Autor          : Mustermann, Max
*
* Bemerkung      : Demo für den ersten Meilenstein
*
* Letzte Änderung : 04. Mai 2018
*
\************************************************************/
```

Funktionen werden innerhalb eines Moduls implementiert. Der Funtions-Header ist ähnlich wie ein Modul-Header aufgebaut. Zunächst wird der Funktionsname genannt. Bei der Wahl des Funktionsnamens müssen die Bedingungen der Namenskonventionen beachtet werden. Die folgenden Einträge werden wie vorangegangen beschrieben ausgefüllt.

Der Header einer Funktion unterscheidet sich im Wesentlichen von dem eines Moduls durch die Beschreibung der übergebenen und der zurückgegebenen Funktionsparameter. Die Beschreibung der Übergabeparameter erfolgt in tabellarischer Form wie im Quellcode 2 gezeigt. Es wird der Datentyp, Name und eine Beschreibung des Parameters genannt. Ebenso werden der Datentyp und die Beschreibung des Rückgabeparameters der Funktion genannt.

Listing 2: Strukturierung eines Funktions-Headers in C und C++.

```
/************************************************************\
*
* Funktion        : MD_FunktionsName
*
* Datum           : 04. Oktober 2013
*
* Beschreibung    : Zweck dieser Funktion
*
* Implementierung : Visual Studio 2012 Professional
*
* Autor           : Mustermann, Max
*
* Bemerkung       : Code-Review noch ausstehend
*
* Letzte Änderung : 04. Mai 2018
*
* Übergebeparameter :
* Typ      Name       Beschreibung
* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* int      n          Anzahl Elemente des Arrays
* double[] a          Array mit double-Werten
*
* Rückgabeparameter :
* Typ      Beschreibung
* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* int      Rückgabe eines Fehlercodes
*
\************************************************************/
```

2.2 MATLAB Code

Auch in MATLAB-Skripten wird ein Modul mit einem entsprechen Header wie in Quellcode 4 dargestellt eingeleitet. Dieser muss sich vor weiteren Kommentaren oder Implementierungen befinden, damit er als Hilfetext in MATLAB interpretiert wird. Durch die Eingabe des Befehls ”help ModulName” in das *Command Window* der MATLAB-Benutzeroberfläche, soll die Beschreibung des Moduls aufgeführt werden. Dazu wird zunächst der Modulname in Großbuchstaben geschrieben um diesen hervorzuheben. Anschließend erfolgt eine kurze Beschreibung des Zwecks dieses Moduls. Nach einer Leerzeile folgen weitere Modul-Daten, die beim Aufrufen der Hilfefunktion nicht aufgeführt werden sollen. Diese beginnen mit dem

Erstellungsdatum des Moduls. Anschließend wird die verwendete MATLAB-Version aufgeführt, um Inkompatibilitäten zwischen Programmen zu vermeiden. Aus diesem Grund werden auch die benötigten MATLAB-Toolboxen aufgelistet. Außerdem werden der Name des Autors, optionale Bemerkungen und das Datum der letzten Änderung dokumentiert.

Listing 3: Strukturierung eines Modul-Headers in MATLAB.

```
% MODULNAME gefolgt von einer kurzen Beschreibung des Zwecks
%   dieses Moduls

% ************************************************************\
%
% Modul          : ModulName.m
%
% Datum          : 04. Oktober 2013
%
% Implementierung : MATLAB R2013a
%
% Toolbox        : Example Toolbox
%
% Autor          : Mustermann, Max
%
% Bemerkung      : Code-Review noch ausstehend
%
% Letzte Änderung : 04. Mai 2018
%
%************************************************************/
```

Bei in MATLAB implementierten Funktion soll der Header ebenfalls Daten als Hilfetext enthalten. Dazu wird der Header innerhalb der zugehörigen Funktion unmittelbar unter dem Funktionskopf platziert, wie im Quelltext **??** an der Beispielfunktion ”Math_Add_A2B” dargestellt.

Listing 4: Beispiel zur Strukturierung eines Funktions-Headers, der durch ”help Math_Add_A2B” im *Command Window* aufgerufen werden kann

```
function result = Math_Add_A2B(a, b)
% MATH_ADD_A2B addiert zwei Zahlen
%
% Syntax:
%   ergebnis = MATH_ADD_A2B(a, b)
%
% Beschreibung:
%   Es werden die zwei Zahlen 'a' und 'b'
%   beliebigen Datentyps addiert.
%   result = a + b
%
% Eingangswerte:
%   a: erster Summand
%   b: zweiter Summand
%
% Rückgabewerte:
%   result: Ergebnis der Addition von 'a' und 'b'
%
% Beispiel:
%   ergebnis = MATH_ADD_A2B(7, 12.583)

%*************************************************************\
%
% Modul            : Math.m
%
% Datum            : 14. Mai 2018
%
% Implementierung  : MATLAB R2017a
%
% Toolbox          : -
%
% Autor            : Marks, Stephan
%
% Bemerkung        : Beispiel eines Funktions-Headers
%
% Letzte Änderung  : 04. Mai 2018
%
% *************************************************************/

result = a + b;
end
```

Innerhalb des Headers wird der Funktionsname immer in Großbuchstaben geschrieben, damit dieser beim Aufrufen der Hilfe hervorgehoben wird. Der Funktions-Header beginnt mit dem Funktionsnamen und einer kurzen Beschreibung des Zwecks der Funktion. Anschließend wird die Syntax genannt, um die Funktionsdefinition mit allen Parametern zu zeigen. In der Beschreibung wird die Funktion genauer als zuvor beschreiben und Auskunft über die Arbeitsweise beziehungsweise Art der Implementierung gegeben. Des Weiteren werden die Bedeutungen der Übergabeparameter der Funktion erläutert. Ebenso die Rückgabewerte. Abschließend wird ein Beispiel zum Funktionsaufruf gegeben. Der Ausgegebene Hilfetext dieses Beispiel-Headers ist in Abbildung 1 dargestellt. Nach einer Leerzeile folgen wie bereits bei der Beschreibung des Modul-Headers erläutert zusätzliche Daten, die nicht beim Aufruf der Hilfefunktion ausgegeben werden sollen. Nach dem Funktions-Header erfolgt die Implementierung der Funktion.

```
>> help Math_Add_A2B
    Math_Add_A2B addiert zwei Zahlen

    Syntax:
        ergebnis = Math_Add_A2B(a, b)

    Beschreibung:
        Es werden die zwei Zahlen 'a' und 'b'
        beliebigen Datentyps addiert.
        result = a + b

    Eingangswerte:
        a: erster Summand
        b: zweiter Summand

    Rückgabewerte:
        result: Ergebnis der Addition von 'a' und 'b'

    Beispiel:
        ergebnis = Math_Add_A2B(7, 12.583)
```

Bild 1: Ausgegebener Hilfetext der im Quelltext **??** Implementierten Funktion.

2.3 Simulink Model

In Programmen, die in Simulink umgesetzt werden, sollen ebenfalls Header vorhanden sein. Innerhalb eines Simulink Modells können per Doppelklick in das Modell Textfelder zur Kom-

mentierung eingefügt werden wie in Abbildung 2 veranschaulicht. Der Header aus Quelltext 5 wird in ein Textfeld im oberen linken Bereich des Modells eingefügt.

Listing 5: Strukturierung eines Modell-Headers in Simulink.

```
%*********************************************************************\
%
% Modell         : ModellName.slx
%
% Datum          : 04. Oktober 2013
%
% Beschreibung   : Zweck dieses Modells
%
% Implementierung : MATLAB R2013a
%
% Toolbox        : Example Toolbox
%
% Autor          : Mustermann, Max
%
% Bemerkung      : Code-Review noch ausstehend
%
% Letzte Änderung : 04. Mai 2018
%
%*********************************************************************/
```

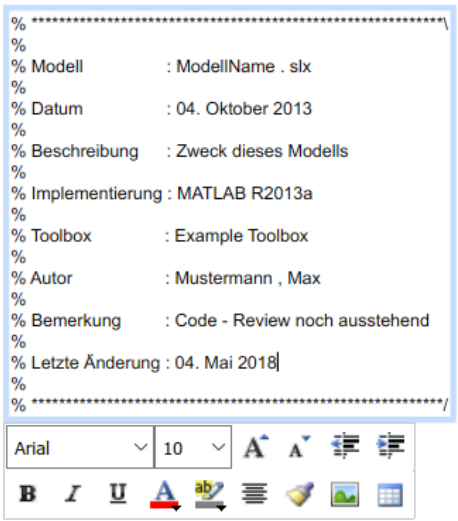

Bild 2: Kommentarfeld in einem Simulink-Modell mit eingefügtem Header aus Quelltext 5.

3 Namenkonventionen von Variablen

Aufbau von Variablen:

Beschreibung	**Modulname**	**Beschreibung**	**_**	**Typ**
Inhalt	Kürzel des Modulnamens. 2 – 5 Buchstaben.	Aussagekräftige Beschreibung des Inhaltes	Unterstrich	u8, s8, u16, s16, u32, s32, st (struct), bit (bit)
Beispiel	**MD_**ErrCount_u8	MD_**ErrCount**_u8	MD_ErrCount**_**u8	MD_ErrCount_**u8**
Bemerkung	projektspezifisch einheitlich, im Datenlexikon definiert. Gilt nicht für lokale Funktionen und lokale Variablen.	Verwendung einheitlicher Bezeichner	Entfällt bei einer Variable die von einem typedef abgeleitet ist.	Entfällt bei einer Variable die von einem typedef abgeleitet ist. Wenn bei der Definition einer struct, enum oder union ein tag deklariert wird, bekommt der Tagname das Suffix tag angehäng t.

Da in der Hochsprache Matlab keine Datentypen definiert werden, kann der Datentyp im Variablennamen entfallen, d. h. „_Typ“ fällt weg.

3.1 Namen von Funktionen

Funktionsnamen werden durch eine Aneinanderreihung von Wörtern aufgebaut, wobei in jedem dieser Wörter der erste Buchstabe groß geschrieben wird und der Rest klein. Werden Funktionen außerhalb eines Moduls verwendet, wird das Modulkürzel vorangestellt, handelt es sich um locale Funktionen wird der erste Buchstabe klein geschrieben. Funktionen bekommen keinen Typen-Suffix.

Beispiele:

```
void MD_MisalignmentDetection(void);
static BIT getRawData (void);
```

3.2 Namen von Konstanten

Unter Konstanten werden an dieser Stelle die Definitionen für den Präprozessor (per `#define`) verstanden. Über die Forderung nach einer aussagekräftigen Wahl des Namens hinaus gelten folgende Regeln:

1. Symbolische Konstanten werden durchgehend mit Großbuchstaben beschrieben, wobei längere Bezeichnungen zur besseren Lesbarkeit durch Unterstriche gegliedert werden.

2. Analog zu der Namenskonvention für Variablen wird auch an die Konstante der Typ angehängt. Der angegebene Typ wird dabei durch den Wertebereich vorgegeben, den diese Konstante annehmen kann.

3. Ein Modulpräfix sollte bei globalen Konstanten Aufschluss über den Definitionsort geben.

Beispiel:

```
#define MD_MAX_RAWCHANNEL_TRANSITIONS_u8 ((u8)20) /* globale Konstante */
static const u16 ENOUGH_DATA_LIMIT_u16 = 10;  /* modullokale Konstante */
```

3.3 Namen von Datentypen

Alle vom Benutzer definierten Typen, also diejenigen die mit `typedef` definiert sind, müssen mit der Endung `_t` versehen werden. Der Name von Datentypen wird wie bei Funktionsnamen aus Wörtern gebildet, deren erster Buchstabe groß geschrieben wird.

Beispiel:

```
typedef struct
{
UncertaintyStruct_t Uncertainty_st; // uncertainty of result
s16 Value_s16; // resulting value
}
ResultValueStruct_t;
```

3.4 Namen von Variablen

Diese Notation legt die Benennung der Variablen bezüglich ihres Typs fest. Der Name wird wie bei Funktionsnamen aus Wörtern gebildet, deren erster Buchstabe groß geschrieben wird. Lokale Variablen beginnen mit einem Kleinbuchstaben und globale mit dem Modulpräfix. Als Suffix ist einer der folgenden Typenbezeichner getrennt mit einem Unterstrich zu verwenden:

u8	unsigned Char (8 Bit Integer ohne Vorzeichen)	[0,255]
s8	signed Char (8 Bit Integer mit Vorzeichen)	[−128,+127]
u16	Word (16 Bit Integer ohne VZ)	[0, 65 535]
s16	Word (16 Bit Integer mit VZ)	[−32 768, +32 767]
u32	Double-Word (16 Bit Integer ohne VZ)	[0, 4 294 967 295]
s32	Double-Word (16 Bit Integer mit VZ)	[−2 147 483 648, +2 147 483 647]

Empfehlung (nicht bindend):

bit	Bit (FALSE, TRUE)	[0,1]
p	Zeiger (pointer) auf nachfolgenden Typ	
a	Feld (array) vom nachfolgenden Typ	
st	Struktur	-

Beispiele:

```
u16 MD_StdAlignmentAngle_u16;          /* globale Variable             */
static TrackStruct_t TrackList_ast[25];  /* modulglobales Feld ein Struktur */
static BIT StartupWithDTC1678Active_bit;  /* modulglobale Variable    */
TrackStruct_t* TrackList_pst           /* Zeiger auf eine Struktur     */
BIT channelTransValid_bit;             /* lokale Variable vom Typ BIT  */
```

4 Sonderregeln

4.1 Schleifeninkrement

Array Indices dürfen in einfachen For-Schleifen zur Übersichtlichkeit mit einfachen Zählervariablen belegt werden. In komplexen oder verschachtelten Schleifen sollten jedoch sprechenden Namen verwendet werden. Beispiel:

```
for (i=1;i<3;i++)
{
  Memory[i]=0;
}
```

besser

```
for (index_u8=1; index_u8<3; index_u8++)
{
  Memory[index_u8]=0;
}
```

Weitere Beispiele fur Indexbezeichner sind:

`cnt_u8, trackCnt_u8, channelNr_u8`

5 Wortwahl

Die Wortwahl kann durch einige Basisregeln erleichtert werden

- Variablennamen müssen aus Wörtern der deutschen Sprache gebildet werden.
- Die Variablen müssen einfach lesbar und verständlich sein. Die Syntax der englischen Worte HorizontalAlignment ist beispielsweise einfacher lesbar als AlignmentHorizontal.
- Lesbarkeit geht vor Kürze . CalculateHorizontalAlignment ist beispielsweise verstandlicher als CalPsi (Ψ ist hier die schwer verständliche Abkürzung für den horizontalen Ausrichtungswinkel).
- Unterstriche dienen nur zur Trennung bei Konstantennamen oder zur Abgrenzung des Datentypen Suffix.
- Bindestriche und andere nicht alphanumerische Zeichen (Buchstaben und Ziffern) sind zu vermeiden.
- Die zuvor beschriebene Variante der ungarischen Notation ist zu verwenden.
- Bezeichnungen, die in Konflikt mit gängigen Bezeichnern von Programmiersprachen stehen, sind zu vermeiden, z. B. `sprintf`, `if`, `for`, `while` etc.

Programmierrichtlinie für das Erstellen von Signalflussplänen (z. B. Matlab/Simulink)

6 Namenskonventionen für Variablen und Parameter

Für die einzelnen Module im Signalflussplan werden Modulkürzel mit meist je 3 Buchstaben verwendet, z. B. für das Modul Lenkwinkelsensor das Kürzel LWS und für das Hauptmodul Sensoren die Kürzel SEN.

Signalnamen enthalten diese Bestandteile, die jeweils durch einen Unterstrich (_) miteinander verbunden werden:

1. Modulkürzel in Groß- und Kleinschreibweise, ggf. mehrere, z. B. SenLws

2. Variablenname, z. B. Lenkwinkel oder auch abgekürzt Lw

3. Koordinatensystem, z. B. _K (körperfest), _V (visionssystem, z. B. Kamera), _I (Inertialsystem), _B (Bahnsystem, tangential an Schwerpunktsbahn)

4. Datentyp wie in der Tabelle auf Seite 9 beschrieben. In Signalflussplänen meist float mit 64 bit, also _f64 oder für logische Variablen mit den Zuständen 0 oder 1 ein _bit

Beispiel ohne Koordinatensystem (gibt es nicht bei allen Größen): **SenLws_Lw_f64**

Beispiel mit Koordinatensystem: **SenVx_vx_K_f64**

Parameter werden genauso wie Variablen bezeichnet, enthalten aber den Präfix PAR_, so z. B. **PAR_SenLws_Initialwert_u8**

7 Kommentare

Alle Variablen und Konstantendefinitionen müssen einen kurzen Kommentar zur Beschreibung der Verwendung behinhalten. Alle Funktionen und Algorithmen müssen einen Kommentar zu ihrer Funktion, sowie eine genauere Spezifikation von Ein- und Ausgabeparametern enthalten. Auch komplexere Codeblöcke innerhalb einer Funktion müssen mit einer kurzen Beschreibung versehen werden. Die Funktionsweise des Codes muss im Großen und Ganzen allein anhand der Kommentare nachvollzogen werden können.

8 Erstellung von Signalflussplänen (Matlab/Simulink)

Um eine einfache Lesbarkeit von Signalflussplänen zu gewährleisten, werden folgende Richtlinien aufgestellt, sie Abbildung 3:

- Eingänge: Hintergrund gelb
- Ausgänge: Hintergrund grün
- Blöcke, die verstellbare Parameter enthalten: Hintergrund hellblau

Bild 3: Einheitliche Darstellung von Ein- und Ausgängen sowie Blöcken mit Parametern.

Alle Parameter (außer physikalische Konstanten, Zahlen wie 1 und 2 etc.) müssen per Parametername angelegt werden. Diese Parameter werden dann außerhalb der Signalflusspläne (Simulink) eingestellt bzw. bedatet.