IfA - Software-Richtlinien

DOXYGEN-DOKUMENTATIONS-RICHTLINIEN FÜR C

Erstellt von: Dr.-Ing. S. Dyblenko, Dipl.-Ing. S. Reimann

Datum: 24.01.2006

Inhaltsverzeichnis

ALLGEMEINE RICHTLINIEN IM ZUSAMMENHANG MIT DER DOXYGEN-DOKUMENTATION 2
FORMATIERUNG DER QUELLCODEKOMMENTARE FÜR DOXYGEN 2
WO/ WIE SIND KOMMENTARE IM QUELLCODE EINZUFÜGEN? 3
INHALT DER QUELLCODEKOMMENTARE 3
EINFÜGEN VON ZUSÄTZLICHEN DOKUMENTATIONSSEITEN ODER -LISTEN 3
DOKUMENTATION VON MODULEN 4
DOKUMENTATION VON FUNKTIONEN UND MAKROS 5
ERSTELLEN DER AUSGABEFORMATE 5

Wichtige Hinweise zur Installation von Doxygen!

➔ *Die Installationsdateien und ein Beispielprojekt befinden sich im Zip-Archiv* `dxg_bspl.zip`*. Die Datei ist unter dem Link Dokumentationsbeispiel zu finden. Bitte lesen sie zuerst die Datei* `hinweise_liesmich.doc`*. Führen Sie danach die Installation von Doxygen wie beschrieben aus. Konsultieren Sie das Doxygen-Handbuch (liegt im html-Format im Doxygen-Installationsverzeichnis vor) für weitere Informationen zur Handhabung von Doxygen. Erstellen Sie danach die Doxygen-Dokumentation des Beispielprojekts* `Randgen`*.*

Allgemeine Richtlinien im Zusammenhang mit der Doxygen-Dokumentation

- Alle Moduldateien (*.c) sowie Headerdateien (*.h) des dokumentierten Projekts müssen sich in einem **Projektverzeichnis** befinden, in das auch die entsprechende **Doxygen-Konfigurationsdatei** abzulegen ist. Der Dateiname der Doxygen-Konfigurationsdatei ist „`Doxyfile.`“ (ohne Extention).
- Im Projektverzeichnis können auch **Unterverzeichnisse** angelegt werden, die jedoch in die Doxygen-Dokumentation einzubinden sind:
  ➔ `INPUT = ./\`
  `./Unterverzeichnisname`
- Die gesamte Doxygen-Dokumentation ist in einem extra Verzeichnis mit Namen „doxygen“ abzulegen bzw. zu erzeugen. Das doxygen-Verzeichnis sollte sich im selben Pfad bzw. Verzeichnis befinden, wie das Projektverzeichnis:
  ➔ `OUTPUT = ./doxygen`
- In die Doxygen-Dokumentation werden ausschließlich Dateien mit den Endungen *.c, *.h, *.txt einbezogen. Eine entsprechende Voreinstellung ist in der Konfigurationsdatei unter „`FILE_PATTERNS`“ spezifiziert. Projektdateien sollten daher ausschließlich diese Dateiendungen verwenden.
- **Alle** Module und **globalen** Syntaxelemente (Funktionen, Variablen, Konstanten, Makros, Klassen, Methoden, Attribute, Datentypen usw.) des dokumentierten Projekts müssen in die Doxygen-Dokumentation einbezogen werden, d. h. mit entsprechenden Kommentaren im Quellcode dokumentiert werden. Doxygen erzeugt automatisch eine vollständige Liste dieser Syntaxelemente.
- Die Transparenz der Gesamtdokumentation wird durch einen **hohen Modularisierungsgrad** verbessert werden:
  - ➔ Implementieren Sie Unterfunktionen bzw. Makros (z. B. mit `#define`) für häufig benutzte Algorithmen bzw. Berechnungen.
  - ➔ Definieren Sie Makros (mit `#define`) für häufig benutzte oder wichtige Parameter in Gleichungen oder Algorithmen.
- Die Doxygen-Dokumentation ist in deutscher Sprache zu erzeugen:
  ➔ `OUTPUT_LANGUAGE = German`
- Alle in den Doxygen-Richtlinien nicht explizit beschriebenen Einstellungen der Konfigurationsdatei sind in der vorgegebenen Konfigurationsdatei bereits korrekt eingestellt. Diese sollten nur in Ausnahmefällen verändert werden. Zum Beispiel die Optionen `PROJECT_NAME` und `PROJECT_NUMBER` müssen für das bearbeitete Softwareprojekt aktualisiert werden.

Formatierung der Quellcodekommentare für Doxygen

- In der Quellcode-Datei sind Doxygen-Commandos mit „`@`“ einzuleiten.
- **Kurzbeschreibungen** sind mit `@brief` einzuleiten.
- Die **Langbeschreibung** wird nicht durch einen speziellen Doxygen-Befehl eingeleitet, sondern durch eine vorangehende Leerzeile im Kommentarblock. Sie muss mit einer Leerzeile beendet werden oder am Ende des Kommentarblocks stehen.
- Auf umfangreiche **Formatierungen** von Quellcodekommentaren für die Doxygen-Dokumentation im `html`-Stil kann verzichtet werden.
- Hilfreich für die Übersichtlichkeit bzw. Strukturierung der Doxygen-Dokumentation sind jedoch Gruppierungen (`@name`, Kapitel 5 im Doxygen-Handbuch), Zeilenumbrüche (`\n`) oder Aufzählungen (`@li`, `@arg`, Kommentarzeile beginnt mit „-“).

Beispiel für einen Kommentarblock:

```
/** @brief Kurzbeschreibung
 *
 * Langbeschreibung, durch eine Leerzeile
 * abgetrennt, über mehrere Zeilen.\n
 * Aufzählungen in der Langbeschreibung werden von
 * Doxygen automatisch erkannt:
 * - Listenelement1
 * - Listenelement2...
 */
```

Wo/ Wie sind Kommentare im Quellcode einzufügen?

- Alle Module und globalen Syntaxelemente sind mit jeweils einer **Kurz**- und einer **Langbeschreibung** zu versehen. Beide sind unmittelbar vor bzw. über der Definition bzw. Deklaration des zu beschreibenden Syntaxelements einzufügen.
- **Funktionen**, **Modulschnittstellen** (Variablen, Makros) und andere Syntaxelemente (z. B. Datentypen), die bereits in der **Headerdatei** (*.h) deklariert wurden, sind unmittelbar vor bzw. über der Deklaration zu dokumentieren. Sie sollten kein zweites Mal in der Moduldatei (*.c) dokumentiert werden, um Zweideutigkeiten zu vermeiden.
- **Alle modulinternen Syntaxelemente**, die nicht in der Headerdatei deklariert wurden, sind unmittelbar vor bzw. über ihrer Definition in der Moduldatei (*.c) zu dokumentieren.
- Bei der Definition von **benutzerdefinierten** Datentypen (Strukturen, Klassen) sind alle Einzelelemente mit den entsprechenden Beschreibungen zu versehen.

```
/** @brief Datenstruktur zur Demonstration der Doxygen-Richtlinien
 *
 * Langbeschreibung zur Datenstruktur durch eine Leerzeile\n
 * abgetrennt, über mehrere Zeilen.
 */
typedef struct
{
    /** @brief Kurzbeschreibung des ersten Strukturelements
     *
     * Langbeschreibung zum Strukturelement durch eine Leerzeile\n
     * abgetrennt, über mehrere Zeilen.
     */
    int iStrukturelement1;
} MOD_typeStructDatentypname;
```

Inhalt der Quellcodekommentare

- Die Doxygen-Dokumentation, insbesondere die **Schnittstellendokumentation**, sollte während der gesamten Entwicklungsphase möglichst **aktuell** gehalten werden, da sie ein wichtiges Hilfsmittel darstellt. ➔ Nutzen sie dafür die **Kurzbeschreibungen**!
- **Langbeschreibungen** sollten nach Projektende ergänzt bzw. überarbeitet werden.
- Für das Verständnis von implementierten Gleichungen und Algorithmen sind **Wertebereiche** von Variablen, Konstanten usw. hilfreich. **Einheiten** sind in jedem Fall anzugeben.
- **Komplexe Algorithmen**, Zustandsmaschinen oder Gleichungen sind in einem **gesonderten Dokument** mathematisch bzw. theoretisch (verbal) zu dokumentieren. Der Name des Dokuments oder ein Verweis auf eine Literaturquelle muss in der Doxygen-Dokumentation erscheinen.
- Die Beschreibungen sollten beinhalten:
  - **Was** das Element beinhaltet (Variable) bzw. was es berechnet (Funktion).
  - **Wie** das Element zu interpretieren ist (Variable) bzw. wie etwas berechnet wird (Funktion).
- Die Doxygen-Dokumentation kann auch zur **Versionsverwaltung**, **Änderungs-** und **Fehlerverfolgung** eingesetzt werden, siehe Befehle: `@version`, `@test`, `@todo`, `@bug`. Die entsprechenden Listen können in der Doxygen-Dokumentation sehr übersichtlich eingesehen werden.

Einfügen von zusätzlichen Dokumentationsseiten oder -listen

Doxygen bietet außerdem die Möglichkeit **externe Textdateien** (*.txt) im Projektverzeichnis mit in die Dokumentation einzubinden. Diese erscheinen in der Doxygen-Dokumentation unter dem Link „`Zusätzliche Informationen`“. Die **Textdateien** können nur in die Doxygen-Dokumentation eingebunden werden, wenn sie der **Doxygen-Dokumentationssyntax** genügen und mit dem Befehl `@page <Listenname>` eingeleitet werden. Der **Listenname** darf nicht „`Test`“, „`Bug`“ oder „`Todo`“ sein.

➔ Beispiel für eine zusätzliche Textdatei (z. B. `Listenname.txt`) im Projektverzeichnis:

```
/*!
@page Listenname

@date 24.01.2006 Datum der Erstellung dieser Seite

...Text...

*/
```

Dokumentation von Modulen

Die Dokumentierung von Modulen im Doxygen-Stil erfolgt an erster Stelle in einem Kommentarblock in der Moduldatei (*.c) sowie in der entsprechenden Headerdatei (*.h). Dieser einleitende Modul-Dokumentationsblock beinhaltet:

- den vollständigen **Dateinamen** mit Endung unter der Rubrik: `@file`
- eine **Kurzbeschreibung** des Inhalts bzw. der Funktionsweise des Moduls bzw. der Datei unter der Rubrik: `@brief`
- alle an der Datei mitwirkenden **Autoren** unter der Rubrik: `@author`
  Die Angabe der email-Adresse kann hilfreich sein.
- eine ausführlichere Beschreibung der Datei in der **Langbeschreibung**
- Angaben über das **Projekt** (Diplomarbeit, Studienarbeit, Projektname, Betreuer, mitwirkende Firmen/ Institute) sowie über das **Modul** (Modulkürzel) in der **Langbeschreibung** der Datei
- eine Beschreibung aller vorgenommenen **Änderungen** mit **Datum** unter der Rubrik: `@date`
  ➔ In der Änderungsbeschreibung ist auch der **jeweilige Autor** (evtl. Abkürzung verwenden) anzugeben sofern mehrere Autoren an der Datei mitwirken.
- Es können weiterhin **verschiedene Listen** zur Versionsverwaltung (`@version`), Fehlerverfolgung (`@bug`), zu anstehenden Verbesserungen (`@todo`) oder zu ausstehenden Tests (`@test`) automatisch mit Doxygen erstellt werden.

➔ Beispiel für einen einleitenden Kommentarblock einer Headerdatei:

```
/************************************************************/
/**
 *  @file bspdatei.h
 *  @brief Kurzbeschreibung des Inhalts von bspdatei.h
 *  @author Mustermann mustermann@ifa.et.tu-dresden.de
 *
 *  Projekt     : Demonstration der Dokumentationsrichtlinien\n
 *  Betreuer(in): Musterfrau, TU Dresden, IfA \n
 *
 *  Modulkürzel : BSP
 *
 *  @date 19.10.2005 - erste Implementierung
 *  @date 19.11.2005 - Änderung Funktion X, Variable Y
 *
 *  @todo Funktion Y implementieren
 *
 *  @test Funktion Y zusammen mit Funktion X testen
 *
 *  @bug Fehler in Funktion X aufgetreten am 1.1.2000
 *
 *  @version 1.5
 ************************************************************/
```

Dokumentation von Funktionen und Makros

Funktionen sind unmittelbar vor ihrer Deklaration in der Headerdatei (*.h) und Makros unmittelbar vor ihrer Definition in der Moduldatei (*.c) in einem Kommentarblock zu dokumentieren. Auf eine wiederholte Dokumentierung im Doxygen-Stil an anderer Stelle sollte verzichtet werden.

1. Zuerst ist eine **Kurzdokumentation** (`@brief`) der Funktion bzw. des Makros vorzunehmen.
2. Anschließend sind **Übergabeparameter** aufzuführen (`@param`) und ausführlich zu dokumentieren. Die Datentypen sind anzugeben.
   Die Auflistung der Übergabeparameter entfällt für parameterlose Funktionen oder Makros.
3. Anschließend ist der **Rückgabedatentyp** bzw. -parameter aufzuführen (`@return`) und ausführlich zu dokumentieren. Auch der Datentyp `void` ist explizit anzugeben.
4. Danach ist eine **Langbeschreibung** der Funktion bzw. des Makros vorzunehmen.
5. Weiterhin sind alle **Änderungen** mit Datum aufzulisten (`@date`) und kurz zu Beschreiben. Autoren sind anzugeben sofern es für ein Modul mehrere Autoren gibt.
6. Hieran können sich noch **weitere Listen** anschließen (`@todo`, `@test`, `@bug`, `@version`).

Beispiel:

```
/**
 * @brief Kurzbeschreibung der Funktion
 * @param iVar1 ausführliche Beschreibung des Parameters\n
 *              auch über mehrere Zeilen
 * @return int ausführliche Beschreibung des Parameters\n
 *             auch über mehrere Zeilen
 *
 *  Ausführliche Beschreibung (Langbeschreibung) der Funktion\n
 *  über mehrere Zeilen.
 *
 *  @date 19.10.2005 - erste Implementierung
 *  @date 19.11.2005 - Änderung Funktion X, Variable Y
 *
 *  @todo Funktion Y implementieren
 *
 *  @test Funktion Y zusammen mit Funktion X testen
 *
 *  @bug Fehler bei Berechnungen in Funktion X bei speziellen\n
 *       Eingabewerte aufgetreten
 *
 *  @version 1.2
*/
int MOD_iFunktion(int iVar1)
```

Erstellen der Ausgabeformate

- Die Doxygen-Dokumentation ist ohne extra Unterverzeichnisse anzulegen.
  ➔ entsprechende Zeilen der Konfigurationsdatei:
  `CREATE_SUBDIRS = NO`
- Der aktuelle Quellcode ist in der Doxygen-Dokumentation zu verlinken:
  ➔ `SOURCE_BROWSER = YES`
  ➔ `INLINE_SOURCES = NO`
- Das `html`-Format ist in jedem Fall zu generieren:
  ➔ `GENERATE_HTML = YES`
- Andere Ausgabeformate können wahlweise generiert werden.
- Zur Übersichtlichkeit sind call graphs zu generieren:
  ➔ `CALL_GRAPH = YES`
- Bitmaps (z. B. call graphs) sind im `gif`-Format zu speichern:
  ➔ `DOT_IMAGE_FORMAT = gif`