Programmdokumentation
-
DOXYGEN

Die Achillesferse der Softwareentwicklung

Finke, Lutz
IN01
LV: Software-Entwicklung

Gliederung

1. Motivation
2. Geschichte
3. Eigenschaften
4. Benutzung
5. Bedienung
   - 5.1. Einführungsbeispiel
6. Dokumentation
   - 6.1. Variablen
   - 6.2. Klassen
   - 6.3. Funktionen
7. weitere Dokumentationsmöglichkeiten (Gruppierungen, …)
8. grafische Darstellungsmöglichkeiten
9. Kurzvorstellung und Vergleich von Javadoc
10. Fragen

Gliederung

1. **Motivation**
2. Geschichte
3. Eigenschaften
4. Benutzung
5. Bedienung
   - 5.1. Einführungsbeispiel
6. Dokumentation
   - 6.1. Variablen
   - 6.2. Klassen
   - 6.3. Funktionen
7. weitere Dokumentationsmöglichkeiten (Gruppierungen, …)
8. grafische Darstellungsmöglichkeiten
9. Kurzvorstellung und Vergleich von Javadoc
10. Fragen

1. Motivation

Für die Entwickler von Programmen ist es immer wieder ein mühsames Unterfangen zum (zumeist sehr komplexen) Programm nebenbei oder wohl zumeist eher hinterher auch eine Dokumentation für andere Entwickler zu erstellen.

Dies betrifft natürlich nicht nur Informatiker, wenngleich diese doch recht viel Zeit vor dem Rechner verbringen. (Manche von Ihnen zuviel Zeit...)

Zur Bewältigung genau dieser Aufgabe soll heute das Dokumentationswerkzeug

DOXYGEN

(http://www.doxygen.org)

vorgestellt werden. Beginnen wir dabei mit einer kurzen Übersicht, wobei auch andere Programmme mit ähnlicher Funktion nicht unerwähnt bleiben sollen.

1. Motivation

Warum dokumentiert man den Quelltext?

- Programmbeschreibung und Motivation des Programms geben
- Programmstruktur sichtbar machen und alternative Lösungen aufzeigen
- Arbeitsaufwand durch gemeinsam genutzte Bibliotheken und Klassen reduzieren
- Wiederverwendbarkeit des Codes durch Dokumentation von Schnittstellen ermöglichen
- Verständnis des Quelltexts für Programmierer (und Autor) erleichtern

1. Motivation

Es gibt dabei grundsätzlich drei verschiedene Leser des Quelltexts und damit verbundene Arten einer Dokumentation:

- **User:** Benutzer des Endproduktes erwartet ein Handbuch (Bedienungsanleitung) zum Programm.
- **Programmierer:** erwarten ein Referenzhandbuch in denen die Schnittstellen genau dokumentiert sind und evtl. auch Beispiele zu deren Verwendung gegeben werden.
- **Autor:** möchte sein eigenes Programm auch nach 3 Monaten noch verstehen können und verwendet deshalb Kommentare im Quelltext.

In der Regel wird man eine Dokumentation in jeder dieser drei genannten Ebenen bereitstellen.

Gliederung

1. Motivation
2. **Geschichte**
3. Eigenschaften
4. Benutzung
5. Bedienung
   - 5.1. Einführungsbeispiel
6. Dokumentation
   - 6.1. Variablen
   - 6.2. Klassen
   - 6.3. Funktionen
7. weitere Dokumentationsmöglichkeiten (Gruppierungen, …)
8. grafische Darstellungsmöglichkeiten
9. Kurzvorstellung und Vergleich von Javadoc
10. Fragen

2. Geschichte

Doxygen wird seit 1997 als Dokumentationswerkzeug für C und C++ Quelltexte entwickelt. Gedacht war es für die Dokumentation großer Softwareprojekte. Es entstanden für einige Linux-Oberflächen bzw. Programmiersprachen allerdings auch eigene Dokumentationswerkzeuge. Mit am bekanntesten dürften dabei:

- JavaDoc für die Sprache Java
- DOC++ für C++ Programme
- cocoon für C++ Programme
- KDoc für C++ Programme für die Oberfläche KDE

Gliederung

1. Motivation
2. Geschichte
3. **Eigenschaften**
4. Benutzung
5. Bedienung
   - 5.1. Einführungsbeispiel
6. Dokumentation
   - 6.1. Variablen
   - 6.2. Klassen
   - 6.3. Funktionen
7. weitere Dokumentationsmöglichkeiten (Gruppierungen, …)
8. grafische Darstellungsmöglichkeiten
9. Kurzvorstellung und Vergleich von Javadoc
10. Fragen

3. Eigenschaften

Viele dieser Dokumentationstools haben allerdings eigene Anforderungen zur Formatierung an den Quelltext. Die Hauptvorteile von Doxygen sind die unterstützten Sprachen:

Voll	Teil
C, C++, Obejctive C	C#
Java	D
IDL	PHP

Ein weiterer Vorteil ist die Kompatibilität zu

- JavaDoc
- KDoc
- Qt-Dokumentation

3. Eigenschaften

Die unterstützten Datenformate

- HTML (z.B. für online-Dokumentationen)
- Compressed HTML
- Latex (z.B. für Handbücher und interne Dokumente)
- PostScript
- Acrobat Reader PDF (verlinkt)
- RTF (z.B. für die Weiterverarbeitung mit Word)
- Unix man-Seiten
- XML

Von den Ausgabeformaten werden nur HTML, Latex und PDF weiterbetrachtet, da die anderen Formate teile von Doxygen nicht unterstützt. Bsp.:
keine Formeldarstellung in RTF-Format.

Gliederung

1. Motivation
2. Geschichte
3. Eigenschaften
4. **Benutzung**
5. Bedienung
   - 5.1. Einführungsbeispiel
6. Dokumentation
   - 6.1. Variablen
   - 6.2. Klassen
   - 6.3. Funktionen
7. weitere Dokumentationsmöglichkeiten (Gruppierungen, …)
8. grafische Darstellungsmöglichkeiten
9. Kurzvorstellung und Vergleich von Javadoc
10. Fragen

4. Benutzung

- prinzipielle Benutzung: (ist immer gleich)
  - Konfiguration für das konkrete Projekt erstellen
  - Dokumentieren des (vorhandenen) Quelltextes (während der Entwicklung!)
- Reihenfolge der Schritte spielt keine Rolle
- eine neue Dokumentation eines Projektes (auch mit geänderter Konfiguration) ist jederzeit möglich
- Es lohnt sich auch völlig unkommentierten Quelltext mit in ein (vorhandenes) Projekt aufzunehmen.
- Doxygen generiert dann zwar keine ausführlichen Informationen, aber:
  - Dateien,
  - Klassen
  - Funktionen
  - werden zumindest mit in die Übersicht aufgenommen, und man kann sich einen groben, schnellen Überblick verschaffen!

4. Benutzung

Konfigurationsdatei:

Die initiale Konfigurationsdatei wird entweder mit

doxygen -g

oder über den Wizard (der gleich noch in ein schickes Fenster eingebettet ist)

doxywizard

erzeugt. Standardmäßig heißt diese Datei "Doxyfile" und sieht etwa so aus.

4. Benutzung

Konfigurationsdatei: Doxyfile

```
# Doxyfile 1.4.1

#---------------------------------------------------------------------------
# Project related configuration options
#---------------------------------------------------------------------------
PROJECT_NAME           = test_doy
PROJECT_NUMBER         =
OUTPUT_DIRECTORY       = "D:/Dokumente und Einstellungen/billy/Eigene Dateien/doxy"
CREATE_SUBDIRS         = NO
OUTPUT_LANGUAGE        = English
USE_WINDOWS_ENCODING   = YES
BRIEF_MEMBER_DESC      = YES
REPEAT_BRIEF           = YES
ABBREVIATE_BRIEF       = "The $name class" \
                         "The $name widget" \
                         "The $name file" \
                         is \
                         provides \
                         specifies \
```

Gliederung

1. Motivation
2. Geschichte
3. Eigenschaften
4. Benutzung
5. **Bedienung**
   - 5.1. Einführungsbeispiel
6. Dokumentation
   - 6.1. Variablen
   - 6.2. Klassen
   - 6.3. Funktionen
7. weitere Dokumentationsmöglichkeiten (Gruppierungen, …)
8. grafische Darstellungsmöglichkeiten
9. Kurzvorstellung und Vergleich von Javadoc
10. Fragen

5. Bedienung

einzeilig:

```
//! Eine Zeile Dokumentationstext im Qt-Stil.
/// Eine Zeile Dokumentationstext im JavaDoc-Stil.
```

mehrzeilig:

```
/*!
 mehrzeiliger Dokumentationstext
 im Qt-Stil
 führende * in der Zeile werden entfernt;
 werden sie dort benötigt, müssen sie zweimal eingegeben werden
 */

/**
 * mehrzeiliger Dokumentationstext
 * im JavaDoc-Stil
 * führende * in der Zeile werden ebenfalls entfernt;
 * werden sie dort benötigt, müssen sie zweimal eingegeben werden
 */
```

5. Bedienung

Wichtige Befehle des doxygen Dokumentationssystems! ...

Jeder Befehl in doxygen beginnt mit dem Backslash "\" und muß innerhalb eines Kommentares stehen. Wichtige Befehle sind

- \mainpage kennzeichnet den Text der Hauptseite der Dokumentation
- \page erstellt eine neue Seite
- \file kennzeichnet den Text, der die aktuelle Datei beschreibt
- \anchor setzt einen Ankerpunkt, auf den man mit \ref verweisen kann
- \ref verweist auf einen Ankerpunkt oder einen Abschnitt
- \param kennzeichnet die Beschreibung eines Parameters einer Funktion
- \return kennzeichnet die Beschreibung der Rückgabe einer Funktion
- \b schreibt das nachfolgende Wort fett

5. Bedienung

... weitere wichtige Befehle!

Sprache der zu erstellenden Dokumentation:

- mit Hilfe der Konfigurationsdatei kann eine Vielzahl von Sprachen gewählt werden
- einige dieser Sprachen erfordern Erweiterungen der betrachteten Systeme (Japanisch)
- Auszug:

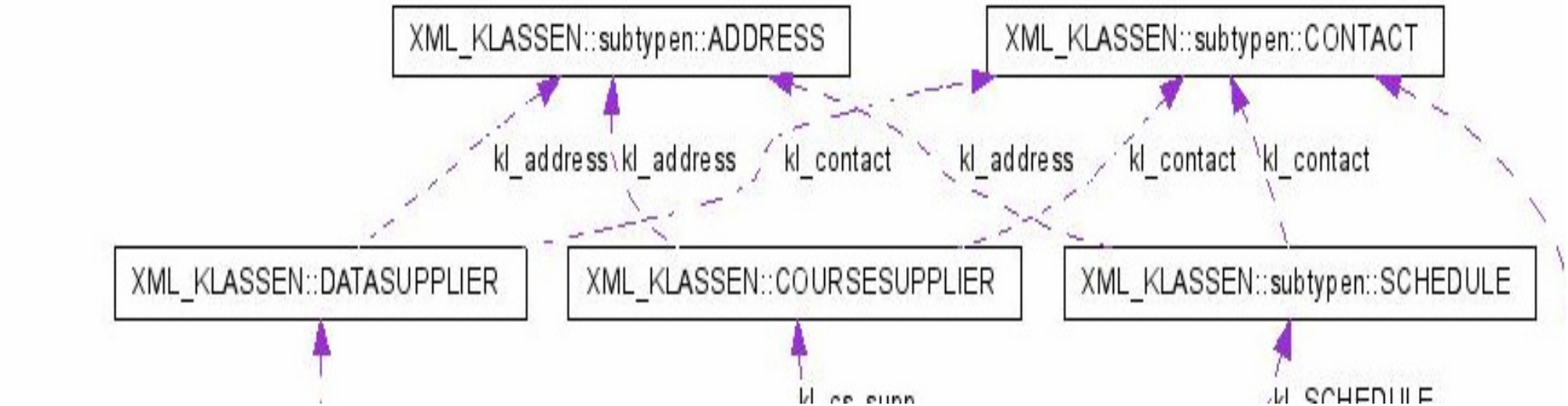
  Brazilian, Catalan, Chinese, Croatian, Czech, Dansih, English, Finnish, French, German, Greek, Hungarian, Italian, Japanese, Korean, Norwegian, Polish, Portuguese, Romanian, Russion, Serbian, Slovak, Slovene, Spanish, Swedish, Ukrainian,

Zur Verwendung dieser Befehle sehe man sich ein erstes Beispiel an.

5. Bedienung

クラス DEFTISCAT

DEFTISCATのコラボレーション図



Gliederung

1. Motivation
2. Geschichte
3. Eigenschaften
4. Benutzung
5. Bedienung
   5.1. Einführungsbeispiel
6. Dokumentation
   6.1. Variablen
   6.2. Klassen
   6.3. Funktionen
7. weitere Dokumentationsmöglichkeiten (Gruppierungen, …)
8. grafische Darstellungsmöglichkeiten
9. Kurzvorstellung und Vergleich von Javadoc
10. Fragen

5.1. Einführungsbeispiel

Beispiel:

Im Bsp. wollen wir das Integral der Funktion $u(x,y)=2x \cdot y$ berechnen.
In einem ersten Ansatz verwenden wir hierzu die Trapezregel mit 20, 40, 60 und 100 Stützstellen auf dem Einheitskreisrand und geben die berechneten Werte aus.

Vorgehensweise :

- bevor man die erste Zeile Quelltext schreibt sollte man mit der Dokumentation beginnen
- jede neue Funktion, Klasse oder (globale) Variable sollte dokumentiert werden bevor sie implementiert wird

  → verhindert undokumentierten Quelltext und erleichtert die Implementierung

→ Resultierender Quelltext und die daraus generierte Dokumentation

5.1. Einführungsbeispiel

Quelltext: main.cpp

```
#include <cmath>
/** \mainpage
* Das Programm berechnet das Integral der Funktion \f$ u(x,y)=2x\cdot y\f$
* entlang des positiv orientierten Rand des Einheitskreises.
* In einem ersten Ansatz verwenden wir hierzu die Trapezregel mit 20, 40, 60
* und 100 Stützstellen auf dem Einheitskreisrand und geben die
* berechneten Werte aus. Die Funktion main() steuert den Programmablauf.
*/
…
```

- Dokumentation enthält Text für die Hauptseite
- Befehl \mainpage darf in einer beliebigen Datei des Projekts stehen, doxygen als Input-File bekannt gemacht wurde

5.1. Einführungsbeispiel

Quelltext: main.cpp

```
…
/** \file
* main.cpp ist die einzige Datei dieses Projekts. Die Berechnung des Integrals
* spielt sich in der main Funktion ab.
*/
…
```

- **\file** bezieht sich auf diejenige Datei, in der dieser Befehl auftritt
- d.h. der Kommentar wird der Datei main.cpp zugeordnet

5.1. Einführungsbeispiel

Quelltext: main.cpp

- Dokumentation der Funktion main

```
…
/**
* berechnet das Integral der Funktion \f$ u(x,y)=2x\cdot y\f$
* entlang des positiv orientierten Rand des Einheitskreises.
* Es wird die Trapezregel mit 20, 40, 60 und 100 Stützstellen auf
* dem Einheitskreisrand verwendet und die berechneten Werte ausgegeben.
*/
int main(){…}
```

5.1. Beispiel

MainPage:

Integral Dokumentation

V3

Das Programm berechnet das Integral der Funktion $u(x,y) = 2x \cdot y$ entlang des positiv orientierten Rand des Einheitskreises. In einem ersten Ansatz verwenden wir hierzu die Trapezregel mit 20, 40, 60 und 100 Stützstellen auf dem Einheitskreisrand und geben die berechneten Werte aus. Die Funktion **main()** steuert den Programmablauf.

Erzeugt am Thu May 12 19:34:53 2005 für Integral von Finke, Lutz 01IN *1.4.1*

5.1. Beispiel

Verzeichnisse:

Integral Verzeichnisse

Diese Verzeichnishierarchie ist -mit Einschränkungen- alphabetisch sortiert:

- vortrag
  - schriftform
    - bsp
      - integral

5.1. Beispiel

Dateiauflistung:

Integral Auflistung der Dateien

Hier folgt die Aufzählung aller dokumentierten Dateien mit einer Kurzbeschreibung:

D:/vortrag/schriftform/bsp/integral/main.cpp [code]

5.1. Beispiel

Dateibeschreibung:

main.cpp-Dateireferenz

```
#include <cmath>
```

Include-Abhängigkeitsdiagramm für main.cpp:

gehe zum Quellcode dieser Datei

Funktionen

int **main** ()

Ausführliche Beschreibung

main.cpp ist die einzige Datei dieses Projekts. Die Berechnung des Integrals spielt sich in der main Funktion ab.

Definiert in Datei **main.cpp**.

Gliederung

1. Motivation
2. Geschichte
3. Eigenschaften
4. Benutzung
5. Bedienung
   5.1. Einführungsbeispiel
6. **Dokumentation**
   - **6.1. Variablen**
   - **6.2. Klassen**
   - **6.3. Funktionen**
7. weitere Dokumentationsmöglichkeiten (Gruppierungen, …)
8. grafische Darstellungsmöglichkeiten
9. Kurzvorstellung und Vergleich von Javadoc
10. Fragen

6.1. Variablen

Wie dokumentiert man Variablen innerhalb von main()? …

- nachdem wir Funktion main() dokumentiert haben beginnen wir mit der Implementation
- lokale Variable innerhalb main()

```
int main(){/** \file */
/** ist ein array aus 4 Integern, … erklärt.
*/
int stuetzstellen[]={20,40,60,100};
}//Ende von main()…
```

- /** \file*/ , macht es möglich Variablen innerhalb von Funktionen zu deklarieren, DOXYGEN bearbeitet sie als wären es globale Variablen
- "HIDE_UNDOC_MEMBERS,, muss aktiviert werden

6.1. Variablen

… Wie dokumentiert man Variablen innerhalb von main()?

weitere Dokumentationsarten:

- folgende Methoden zeigen Dokumentationsarten einer Variable

```
int var; /*!< Detailed description after the member */
int var; /**< Detailed description after the member */
int var; //!< Detailed description after the member
int var; ///< Detailed description after the member
int var; //!< Brief description after the member
int var; ///< Brief description after the member
```

6.2. Klasse

Wie dokumentiert man eine Klasse?

- dokumentiert wird jede öffentliche Methode und jedes öffentliche Datenelement der Klasse
- private Methoden und Datenelemente werden von DOXYEN ignoriert solange „EXTRACT PRIVATE“ deaktiviert ist
- \code … \endcode Quelltext-Beispiele, welche die Verwendung der Klasse beschreiben

```
/** Kommentar beginnt
* ...
* \code
* IntegralOperator (...);
* \endcode
* ... weitere Kommentare ...
*/
```

6.2. Klassen

Wie dokumentiert man eine Klasse?

Folgende Parameter können innerhalb der Klassendokumentation angegeben werden.

\author,	**Bezeichnung des Authors**
\date,	**Datum**
\version,	**Version der Klasse**
\todo,	**Todo - Anweisungen**
\warning,	**Warnhinweise**
\bug,	**Bug - Meldungen**
\deprecated,	**Einteilung in Paragraphen**
\sa (see also)	**spezifiziert Referenzen zu andere Klassen, Methoden, Variablen ...**

6.3. Funktionen

Wie dokumentiert man eine Funktion?

- es stehen die Befehle \param und \return zur Verfügung, mit denen man die Parameter bzw. den Rückgabewert der Funktion beschreiben kann
- Option **"SOURCE_BROWSER,,** bindet Quelltext in Dokumentation ein

- Dokumentation die nicht von doxygen verarbeitet werden soll, kann mit folgender Syntax erzeugt werden

```
// Das ist ein von doxygen ignorierter Kommentar.

// Das ist ein von doxygen ignorierter Kommentar, der
// über
// mehrere Zeilen geht
```

6.3. Funktionen

Kernfunktion funktion1:

```
/** ist eine Kernfunktion für die Klasse IntegralOperator. Die Kernfunktion des
* Integraloperators \f[ (K\varphi)(x):=\int_{\partial D} k(x,y) \varphi(y) ds(y)\f]
* ist die Funktion \f$ k(x,y) \f$.
* \param x1 ist die \a x -Koordinate des Auswertungspunktes.
…
* \return gibt einen double-Wert zurück der dem Funktionswert
* des Kerns an der Stelle (x,y) mit x=(x1,x2) und y=(y1,y2) entspricht.
*/
double funktion1(double x1,double x2,double y1,double y2);
```

- \f$ markiert den Begin und das Ende einer in-text Formel
- \f[markiert den Begin einer Formel, welche zentriert und auf einer Separaten Zeile dargestellt wird (Ende \f])

Gliederung

1. Motivation
2. Geschichte
3. Eigenschaften
4. Benutzung
5. Bedienung
   - 5.1. Einführungsbeispiel
6. Dokumentation
   - 6.1. Variablen
   - 6.2. Klassen
   - 6.3. Funktionen
7. **weitere Dokumentationsmöglichkeiten**
8. grafische Darstellungsmöglichkeiten
9. Kurzvorstellung und Vergleich von Javadoc
10. Fragen

finke:

<label> unique

7. weitere Möglichkeiten

Gruppierungen:

2 Arten: Module, Member Group

Module: generiert neue Seite für jede Gruppe

- files, namespaces, classes, functions, variables, enums, typedefs, und defines, aber auch andere groups
- **\defgroup <label>** in eigenen Dokumentationsblock definiert eine Gruppe
- **\ingroup** <groupname> fügt Elemente einer Gruppe hinzu

Member Group:

- Elemente die innerhalb eines @{ **...** @} stehen werden der Übergeordneten Gruppe zugeordnet

7. weitere Möglichkeiten

Gruppierungen Bsp.:

```
/** @defgroup group1 Erste Gruppe
*Das ist die erste Gruppe
*@{ */

   /** @brief Ueberschrift der Klasse1 in Gruppe1
   */ class C1 {};
   /** Funktion in Gruppe1 */ void func() {}

   /** @} */ // Ende der Gruppe1

/**
* @ingroup group1
* @brief Ueberschrift der Klasse C3 in Gruppe1
*/ class C3 {};
...
```

7. weitere Möglichkeiten

Befehle:

- **\htmlonly, \xmlonly ...** Dokumentation innerhalb dieser Befehle wird nur für die jeweils bezeichnete Ausgabe verwendet **... \endxmlonly \endhtmlonly**
- **\image <format> <file> ["caption"] [<sizeindication>=<size>]**
  - fügt ein Image der Dokumentation bei
  - Achtung: Image-Format bei HTML abhängig vom Browser
- **\copydoc <link-object>**
  - kopiert Dokumentationsblöcke der angegebenen Object-Spezifikation und fügt Sie an der Position des Kommandos ein
  - link-object = Member (einer Klasse, Datei oder Gruppe), eine Klasse, ein Namensraum, eine Gruppe, eine Seite, oder ein File

7. weitere Möglichkeiten

Verzweigungen:

- **\if <section-label> , \endif, \ifnot, \else, \elseif**
- ermöglichen Verzweigung innerhalb der Dokumenation
- Verzweigung wird verfolgt wenn <section-label> in ENABLED_SECTIONS aktiviert wurde
- Vorteil:
  - kann ausserhalb des Quellcodes, d.h. nur mit Hilfe der Konfigurationsdatei verändert werden
  - Quellcodedokumentation für mehrere Einstellung kann innerhalb einer Datei erstellt werden (Bsp.: Sprachen)

```
\if if1
      wenn if1 gesetzt ist.
      \if if2
            wenn if2 && if1 gesetzt ist
\endif
```

```
/*! \if english
* This is English.
* \endif
* \if german
* Das ist Deutsch.
*\endif */
```

Gliederung

1. Motivation
2. Geschichte
3. Eigenschaften
4. Benutzung
5. Bedienung
   5.1. Einführungsbeispiel
6. Dokumentation
   6.1. Variablen
   6.2. Klassen
   6.3. Funktionen
7. weitere Dokumentationsmöglichkeiten (Gruppierungen, …)
8. **grafische Darstellungsmöglichkeiten**
9. Kurzvorstellung und Vergleich von Javadoc
10. Fragen

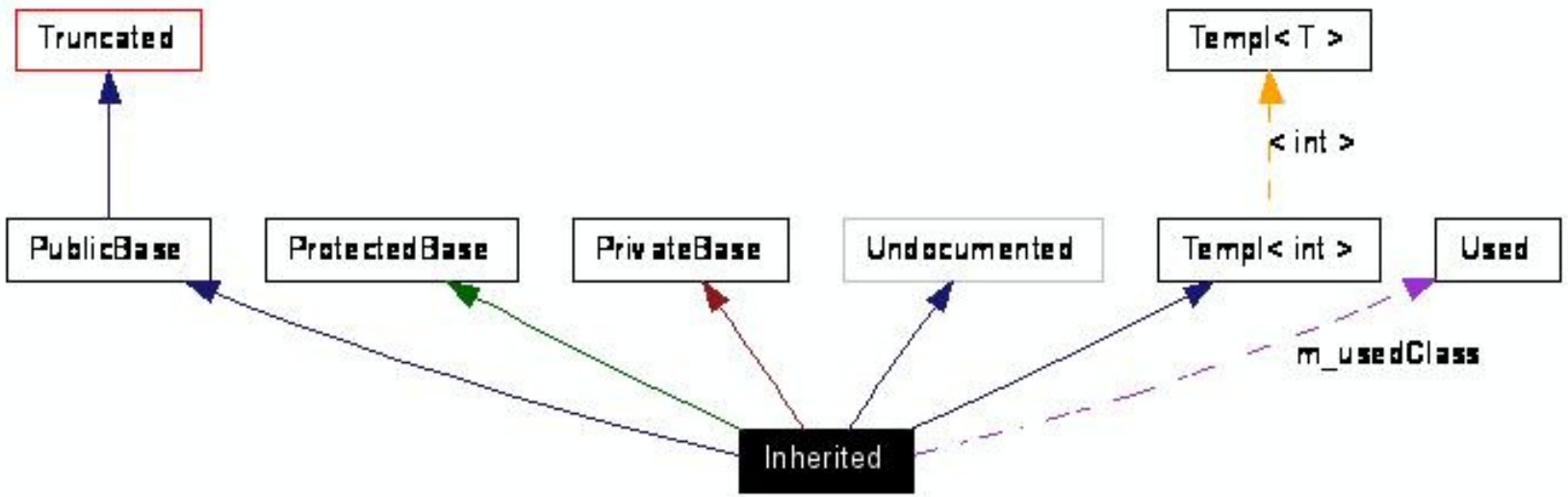
8. grafische Darstellungsmöglichkeiten

Voraussetzung:

- Tool **dot**, ist Teil von **Graphviz**, von AT&T und Lucent Bell Labs
- `HAVE_DOT` Flag aktiv

Befehle ... :

- **\callgraph** innerhalb Kommentarblock aktiv, wird die damit beschriebene Entity grafisch dargestellt
- `CALL_GRAPH` aktiv, generiert Grafik für jede globale Funktion und Klasse



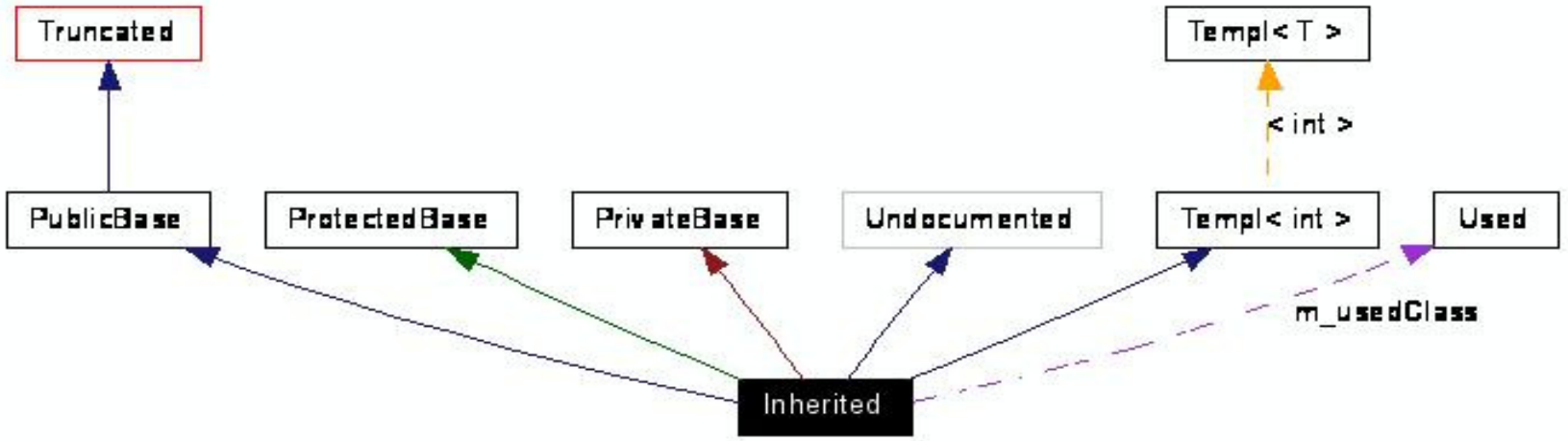
8. grafische Darstellungsmöglichkeiten

Beschreibung der Knoten:

gelb:	initalisiert eine Klasse
weiße:	Klasse deren Dokumentation gerade angezeigt wird
grau:	undokumentierte Klasse
rot:	Klasse ist abgeschnitten in Darstellung

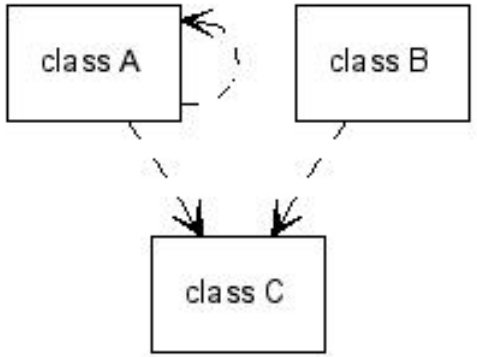
Beschreibung der Kanten:

blau:	public
gestrichelt grün:	protected
gepunktet grün :	private
gelb:	template <Klasse>



8. grafische Darstellungsmöglichkeiten

... Befehle ... :



- **\dot** Startet Textfragment welches ein Beschreibung eines dot-graph enthält
- **\enddot** Ende des Textfragments
- Knoten können mittels URL Attribut versehen werden
- **\ref** innerhalb der URL Value generiert einen Link zu einem Item innerhalb doxygen

```
/*! \brief class A HTWK-Link */ class A {};
/*! class B */ class B {}; ...
/*! \mainpage Class ... :
\dot digraph example { node [shape=record,
fontname=Helvetica, fontsize=10];
a [ label="class A"URL="http://www.imn.htwk-leipzig.de"];
b [ label="class B" URL="\ref B"];
c [ label="class C" URL="\ref C"];
b -> c [ arrowhead="open", style="dashed" ];
a -> a [ arrowhead="open", style="dashed" ];
a -> c [ arrowhead="open", style="dashed" ];
} \enddot */
```

8. grafische Darstellungsmöglichkeiten

... Befehle:

- **`\dotfile <file> ["caption"]`**
- fügt ein mit dot generiertes Image in die Dokumentation ein
- **„caption“** optional, wird als Caption des Images verwendet
- Image wird im angegebenen Output-Pfad hinterlegt
- Leerzeichen innerhalb des File-Namens erfordern Quotes

Beispiel: dot.dot

```
digraph example
{
  edge [fontname="Helvetica",fontsize=10labelfontsize=10];
  node [shape=record, fontname=Helvetica, fontsize=10];
  b [shape="box", label="class B" URL="\ref B"];
  c [shape="box", label="class C" URL="\ref C"];
  b -> c [dir=back, fontsize=10,style="solid",fontname=„..."];
}
```

8. grafische Darstellungsmöglichkeiten

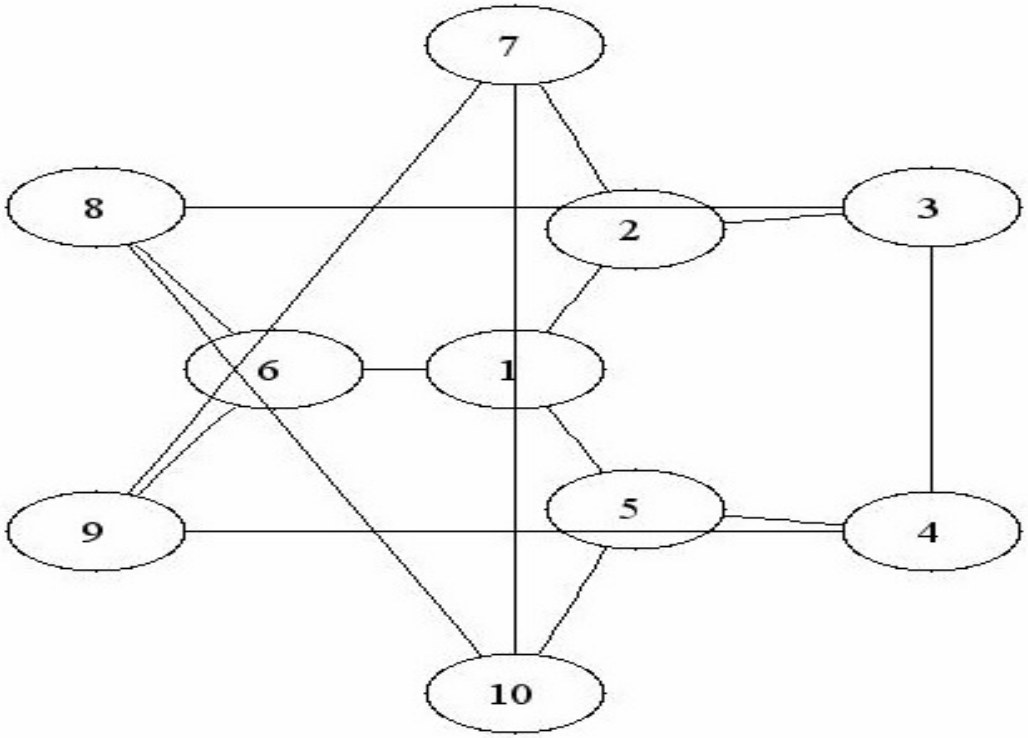
Einbinden eines Graphen

V1

Eingefügtes Bild

Der folgende Graph dürfte trotz veränderten Bild jeden bekannt sein.

```
\dot
graph Petersen
   { 1 -- 2  ;  2 -- 3; 3 -- 4;
     4 -- 5  ;  5 -- 1; 6 -- 8;
     8 -- 10; 10 -- 7; 7 -- 9;
     9 --  6; 1 -- 6; 2 --  7;
     3 -- 8; 4 --  9;  5 -- 10
   }
\enddot
```



Gliederung

1. Motivation
2. Geschichte
3. Eigenschaften
4. Benutzung
5. Bedienung
   5.1. Einführungsbeispiel
6. Dokumentation
   6.1. Variablen
   6.2. Klassen
   6.3. Funktionen
7. weitere Dokumentationsmöglichkeiten (Gruppierungen, …)
8. grafische Darstellungsmöglichkeiten
9. **Kurzvorstellung und Vergleich von Javadoc**
10. Fragen

9. Kurzvorstellung und Vergleich von Javadoc

- JavaDoc ist ein Dokumentationswerkzeug,
  das aus Java-Quelltexten automatisch HTML-Dokumentationsdateien erstellt
- Tool benutzt Teile des javac um die Deklaration zu compilieren
- Javadoc Doclets
  - mittels JD kann Inhalt und Format der Ausgabe Verändert werden
  - mittels eigener Doclets kann jede gewünschte Form der Ausgabe generiert werden (HTML, XML, MIF, RTF)
- Aufruf:

```
javadoc –d C:\home\html Button.java Canvas.java
Graphics*.java
```

- generiert Dokumentation in HTML - Form für die Klassen Button, Canvas und alle Klassen die mit Graphics beginnen

9. Kurzvorstellung und Vergleich von Javadoc

Vorgehensweise:

- erstellt interne Repräsentation der Klassen
- fügt Klassen-Hierarchie mit ein, benutzt die Beziehungen
- generiert daraus den HTML-Code

Nachteil:

- nur eine Programmiersprache wird unterstützt
  - standardmäßig nur HTML-Ausgabeformat
  - für weitere Ausgabeformate benötigt man neue Doclets

Quellen

www.doxygen.org
www.clug.in-chemnitz.de
www.num.math.uni-goettingen.de
www.graphviz.org

Bsp.:
www.kdevelop.org

? Fragen?