Technische Universität Braunschweig

Institut für Flugführung

Dateieingabe und -ausgabe

Abspeichern und Einlesen von Daten

Prof. Dr.-Ing. Peter Hecker, Dipl.-Ing. Paul Frost, 23. Mai 2017

Agenda

04. April	Kick-Off
11. April	Projektmanagement
18. April	Prozessmodelle
25. April	Versionsverwaltung
02. Mai	Einführung Arduino/Funduino
09. Mai	Entwicklungsumgebungen und Debugging
16. Mai	Dokumentation und Testing
23. Mai	**Dateieingabe und -ausgabe**
30. Mai	GUI-Erstellung mit Qt
06. Juni	Exkursionswoche
13. Juni	Bibliotheken
20. Juni	Netzwerke
27. Juni	Projektarbeit
04. Juli	Projektarbeit
11. Juli	Vorbereitung der Abgabe

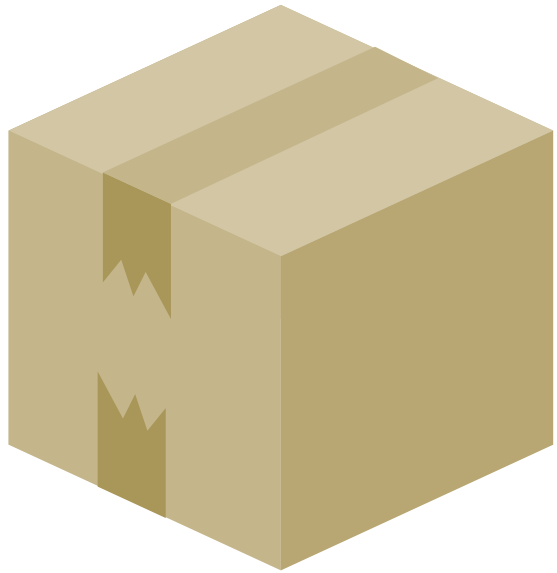
Teil I

Wiederholung

Teil II

Dateieingabe und -ausgabe

Schreiben von Daten



Datei

Schreiben von Daten

1. Datei zum Schreiben öffnen

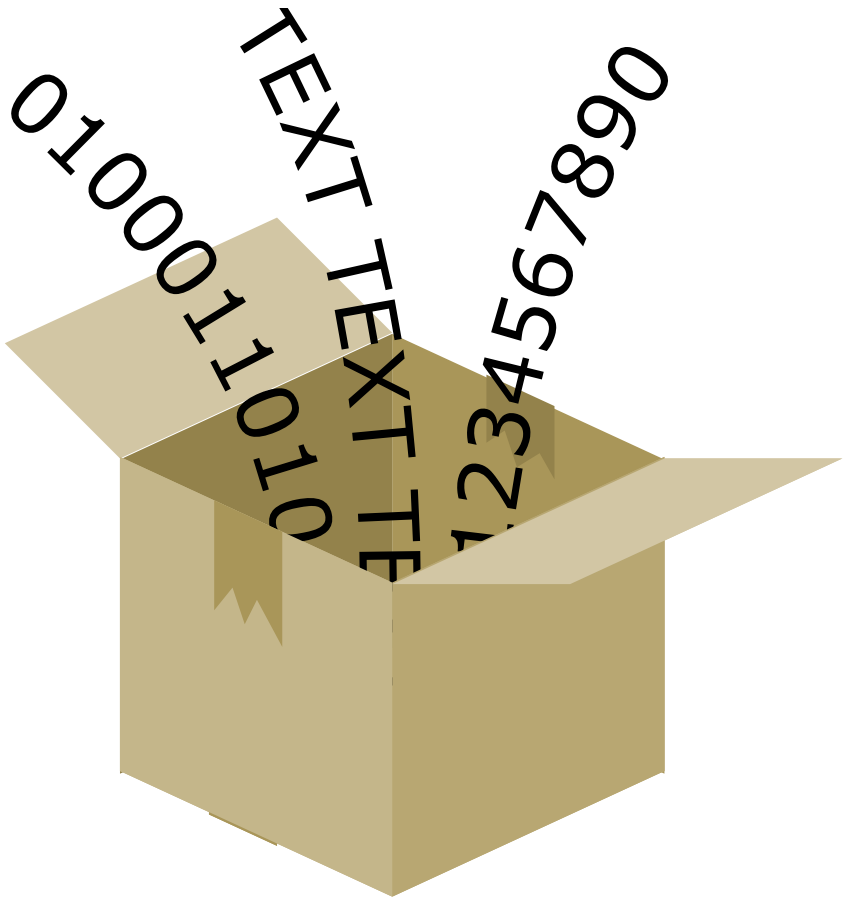
   Falls die Datei noch nicht existiert, wird diese automatisch erstellt

Datei

Schreiben von Daten

1. Datei zum Schreiben öffnen

   Falls die Datei noch nicht existiert, wird diese automatisch erstellt
2. Inhalt in Datei schreiben

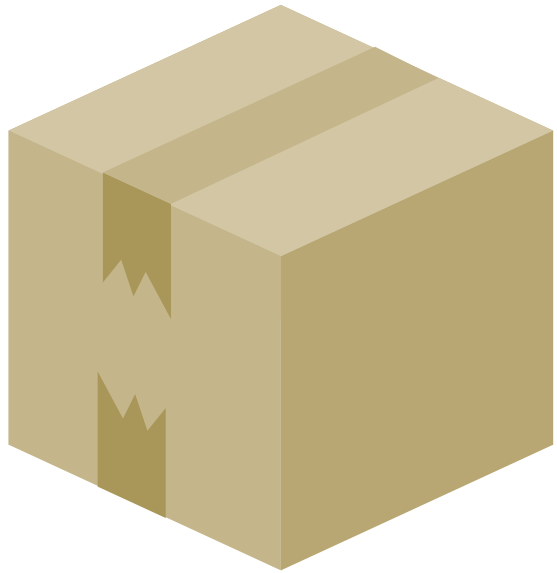


Datei

Schreiben von Daten

1. Datei zum Schreiben öffnen

   Falls die Datei noch nicht existiert, wird diese automatisch erstellt
2. Inhalt in Datei schreiben
3. Datei schließen



Datei

Schreiben von Daten

Beispiel

Listing 1: code/write.cpp

```
1  #include <fstream>
2  using namespace std;
3
4  int main () {
5    ofstream dateiObjekt; //Anlegen einer Instanz der Klasse ofstream
6    // ofstream steht fuer output file stream
7    dateiObjekt.open ("beispiel.txt");
8    dateiObjekt << "API2017";
9    dateiObjekt.close();
10   return 0;
11 }
```

Wichtig

Der Inhalt der geöffneten Datei wird gelöscht!

Schreiben von Daten

Ergebnis

Binäre Werte

A	P	I	2	0	1	7
0100 0001	0101 0000	0100 1001	0011 0010	0011 0000	0011 0001	0011 0111

Dezimale Werte

A	P	I	2	0	1	7
65	80	73	50	48	49	55

Anzeige im Texteditor

beispiel.txt

```
1  API2017
```

ASCII-Tabelle

HEX	...0	...1	...2	...3	...4	...5	...6	...7	...8	...9	...A	...B	...C	...D	...E	...F
0...	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1...	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2...	SP	!	"	#	$	%	&	’	(	)	*	+	,	-	.	/
3...	0	1	2	2	4	5	6	7	8	9	:	;	<	=	>	?
4...	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5...	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6...	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7...	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Anfügen von Daten

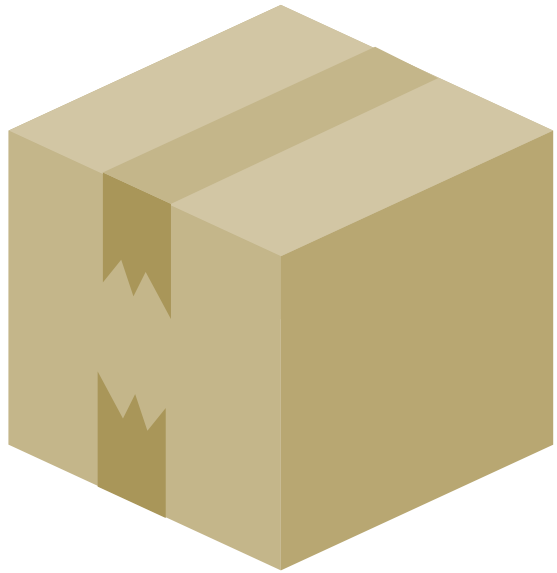
Beispiel

Daten werden an den bestehenden Inhalt angefügt:

Listing 2: code/append.cpp

```
1  #include <fstream>
2  using namespace std;
3
4  int main () {
5    ofstream dateiObjekt; //Anlegen einer Instanz der Klasse ofstream
6    // ofstream steht fuer output file stream
7    dateiObjekt.open ("beispiel.txt", ios::app);
8    dateiObjekt << "API2017\n";
9    dateiObjekt.close();
10   return 0;
11 }
```

Lesen von Daten



Datei

Lesen von Daten

1. Datei zum Lesen öffnen

Datei

Lesen von Daten

1. Datei zum Lesen öffnen
2. Prüfen, ob die Datei geöffnet werden konnte

Datei

Lesen von Daten

01000110101

1. Datei zum Lesen öffnen
2. Prüfen, ob die Datei geöffnet werden konnte
3. Datei zeilenweise auslesen

Datei

Lesen von Daten

1. Datei zum Lesen öffnen
2. Prüfen, ob die Datei geöffnet werden konnte
3. Datei zeilenweise auslesen

Datei

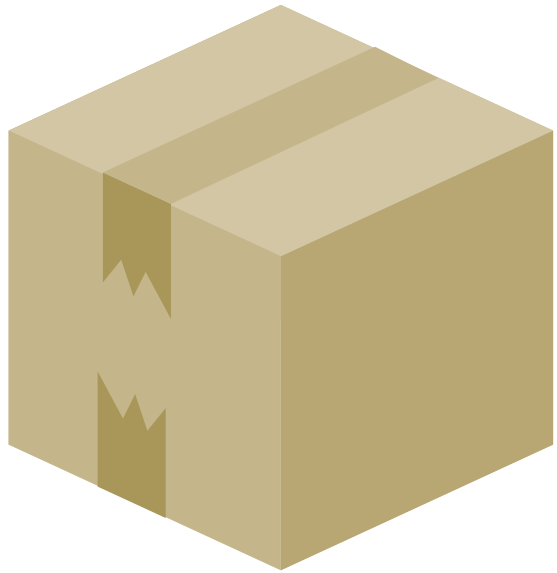
Lesen von Daten

1. Datei zum Lesen öffnen
2. Prüfen, ob die Datei geöffnet werden konnte
3. Datei zeilenweise auslesen

Datei

Lesen von Daten

1. Datei zum Lesen öffnen
2. Prüfen, ob die Datei geöffnet werden konnte
3. Datei zeilenweise auslesen
4. Datei schließen



Datei

Lesen von Daten

Beispiel 1/2

Listing 3: code/read.cpp

```
1 #include <string>
2 #include <iostream>
3 #include <fstream>
4
5 using namespace std;
6 // ...
```

Lesen von Daten

Beispiel 2/2

Listing 4: code/read.cpp

```
1  int main () {
2    string line;
3    ifstream dateiObjekt ("beispiel.txt");
4    if (dateiObjekt.is_open())
5    {
6      while ( getline (dateiObjekt,line) )// Zeile auslesen
7      {
8        cout << line << '\n';
9      }
10     dateiObjekt.close();
11   }
12   cin.getline(NULL, 0); // Verhindert Schliessen der Konsole
13
14   return 0;
15 }
```

Comma-separated values *.csv

Eigenschaften

- Werte sind tabellarisch angeordnet
- Eine Zeile in der csv-Datei entspricht einer Zeile der Tabelle
- Erste Zeile wird häufig zur Beschriftung der Spalten genutzt
- Trennzeichen separieren die Spalten

Verwendung

- Messwerte
- Weiterverarbeitung in Programmen möglich

Comma-separated values

Beispiel

Listing 5: code/csvDatei.csv

```
1  Spalte1,Spalte2,Spalte3,Spalte4
2  1,0.6,0,-5
3  2,0.7,0,-4.5
4  3,0.8,0,-4
5  4,0.9,0,-3.5
6  5,1,0,-3
7  6,1.1,0,-2.5
8  7,1.2,0,-2
9  8,1.3,0,-1.5
10 9,1.4,0,-1
```

Comma-separated values — Vor- und Nachteile

Vorteile

- Werte sind im Texteditor einsehbar und interpretierbar
- Sehr einfaches Format
- Spezifikation in der ersten Zeile möglich

Nachteile

- Daten können lediglich als Tabelle abgespeichert werden
- Höherer Speicheraufwand

Initialisierungsdatei *.ini

Eigenschaften

- Werte werden zu Schlüsseln zugeordnet (Wertepaar)

  `Schlüssel = Wert`

- Gruppierung von Wertepaaren möglich

  `[Gruppenname]`

- Pro Zeile ein Wertepaar oder eine Gruppendefinition
- Schlüssel müssen innerhalb einer Gruppe eindeutig sein
- Gruppennamen müssen eindeutig sein

Verwendung

- Speicherung von Einstellungen und Parametern

Initialisierungsdatei Beispiel

Listing 6: code/iniDatei.ini

```
1 [SimConnect]
2 level=verbose
3 console=1
4 RedirectStdOutToConsole=1
5 OutputDebugString=1
6 ;file=c:\simconnect%03u.log
7 ;file_next_index=0
8 ;file_max_index=9
```

Initialisierungsdatei Vor- und Nachteile

Vorteile

- Werte sind im Texteditor einsehbar und interpretierbar
- Einfaches Format
- Wenig Overhead

Nachteile

- Inhalte können nur als Wertepaare gespeichert werden
- Nur eine Gruppenhierarchieebene möglich

Extensible Markup Language *.xml

Eigenschaften

- Werte werden in Tags oder Attributen dargestellt

  ```
  <Tag>Wert</ Tag>
  ```

  ```
  <Tag Attribut="WertAttribut">WertTag</ Tag>
  ```

- Baumstruktur durch Verschachteln von Tags
- Validierung der Datenstruktur möglich

Verwendung

- Webinhalte
- Geographische Daten (OpenStreetMap)

Extensible Markup Language Beispiel

Listing 7: code/xmlDatei.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" >
 <bounds minlat="52.2780700" minlon="10.5269700"/>
 <node id="48999923" visible="true" >
  <tag k="railway" v="switch"/>
  <tag k="railway:local_operated" v="yes"/>
  <tag k="railway:switch" v="default"/>
  <tag k="railway:turnout_side" v="right"/>
 </node>
</osm>
```

Extensible Markup Language Vor- und Nachteile

Vorteile

- Werte sind im Texteditor einsehbar und interpretierbar
- Intuitives Format
- Flexibles Format innerhalb der Baumstruktur
- Validierung der Datenstruktur möglich

Nachteile

- Aufwendiger beim Auslesen als csv- oder Initialisierungsdateien

→ Parser erforderlich

- Leicht erhöhter Speicheraufwand durch schließende Tags

JavaScript Object Notation *.json

Eigenschaften

- Daten können über fest definierte Zeichen zu Objekten gruppiert werden

```
{
  Schlüssel:Wert
}
```

- Trennung der Wertepaare und Unterobjekte erfolgt über ein Komma
- Objekte werden als Baumstruktur angelegt

Verwendung

- Einstellungen
- Datenaustausch

JavaScript Object Notation

Beispiel

Listing 8: code/jsonDatei.json

```
1  {
2  "time":1495458880,
3  "states":
4    [
5      [
6      "a0cfbd",
7      "AAL653  ",
8      "United States",
9      1495458879,
10     1495458879,
11     -78.7932,
12     "..."
13     ]
14   ]
15 }
```

JavaScript Object Notation Vor- und Nachteile

Vorteile

- Unterstützung von 6 Datentypen:

  Nullwerte, Bool-Werte, Zahlen, Zeichenketten, Arrays und Objekte
- Flexibles Format innerhalb der Baumstruktur

Nachteile

- Aufwendiger beim Auslesen als csv- oder Initialisierungsdateien

→ Parser erforderlich

- Etwas schwierigerer Einstieg

Parser

Werkzeug zur Umwandlung von Eingaben

- Wandelt die Eingabe in einer definierten Art und Weise um
- Größere Bibliotheken stellen Parser für gängige Formate bereit
- Beispiele für Parser in Software
  - Webbrowser – HTML-Parser, XML-Parser
  - RSS-Reader
  - Compiler

Für C++ verfügbare Parser — Für Profis

json-Parser `https://github.com/nlohmann/json`

xml-Parser `http://stackoverflow.com/questions/9387610/what-xml-parser-should-i-use-in-c`

Teil III

Serielle Schnittstelle

Serielle Schnittstelle

- Datenübertragung zwischen verschiedenen Geräten
- Bits werden nacheinander übertragen (seriell)
- Bekannte Standards:
  - RS-232
  - Serial ATA (SATA)
  - Universal Serial Bus (USB)

TX/RX-Ports des Arduino sind für maximal 5V ausgelegt. Bei RS232 kann die Spannung bis zu 12 V hoch sein.

Baudrate

- Die Baudrate gibt an, mit welcher Schrittgeschwindigkeit Daten übermittelt werden.
- Einheit: $\frac{\text{Symbol}}{\textit{Sekunde}}$
- Bei Übertragungen mit 2 Spannungen gilt die Einheit $\frac{\text{Bits}}{\text{Sekunde}}$

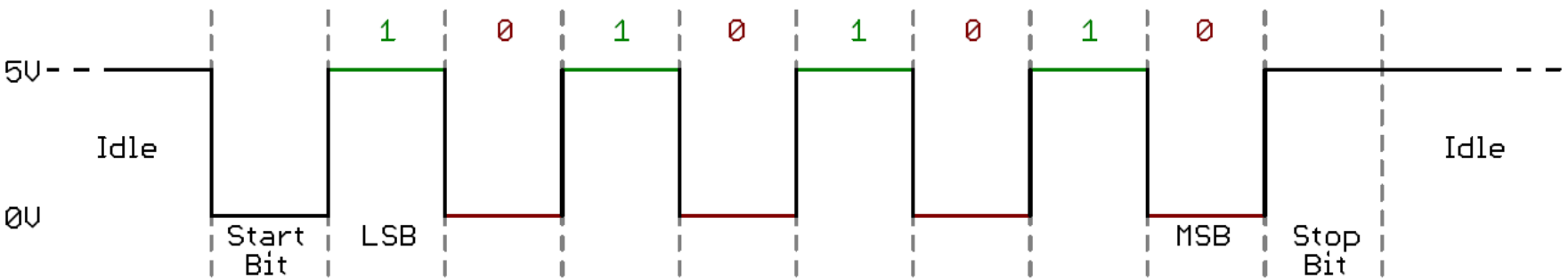
Baud-Rate	max. Länge
2.400	900 m
4.800	300 m
9.600	152 m
19.200	15 m
57.600	5 m
115.200	< 2 m

Wichtig:

Empfänger und Sender benötigen die gleiche Baudrate

Start-/Stop-Bit

- Asynchrone Kommunikation
- Start-Bit kündigt Nachricht an
- Stop-Bit schließt Nachricht ab
- Dazwischen wird ein Datenset mit 5-9 Bits übertragen
  Ein Wort stellt ein Zeichen dar
- Häufig wird eine Länge von 8 Bit verwendet



Paritätsbit (engl. Parity Bit)

- Das Paritätsbit folgt an die Nachricht
- Gibt an, ob die Anzahl der Bits gerade oder ungerade ist
- Vergleich durch Zählen der Bits und der Prüfung, ob eine gerade oder ungerade Anzahl vorliegt

Daten-Bits	Anzahl 1-Bits	Gerade	Ungerade
0000 0000	0	0000 0000 **1**	0000 0000 **0**
0000 0111	3	0000 0111 **0**	0000 0111 **1**
0100 0111	4	0100 0111 **1**	0100 0111 **0**

Arduino Initialisierung serielle Schnittstelle

Listing 9: code/ArduinoSerialInput/ArduinoSerialInput.ino

```
1 void setup() {
2   Serial.begin(9600); // Beim Oeffnen wird die Baudrate festgelegt
3 }
```

Tabelle 1: Standardwerte

Parameter	**Einstellung**
Datenset	8 Bits
Paritätsbit	Keins (engl. none)
Stop-Bit	1 Bit
$\Rightarrow$	8-N-1

Arduino Initialisierung serielle Schnittstelle

Listing 10: code/ArduinoSerialInput/ArduinoSerialInput.ino

```
1  char incomingByte = '\0';   // Speicher fuer das ankommende Byte
2
3  void setup() {
4    Serial.begin(9600); // Beim Oeffnen wird die Baudrate festgelegt
5  }
6
7  void loop() {
8    if (Serial.available() > 0){
9      incomingByte = Serial.read();
10     Serial.print(incomingByte);
11   }
12 }
```

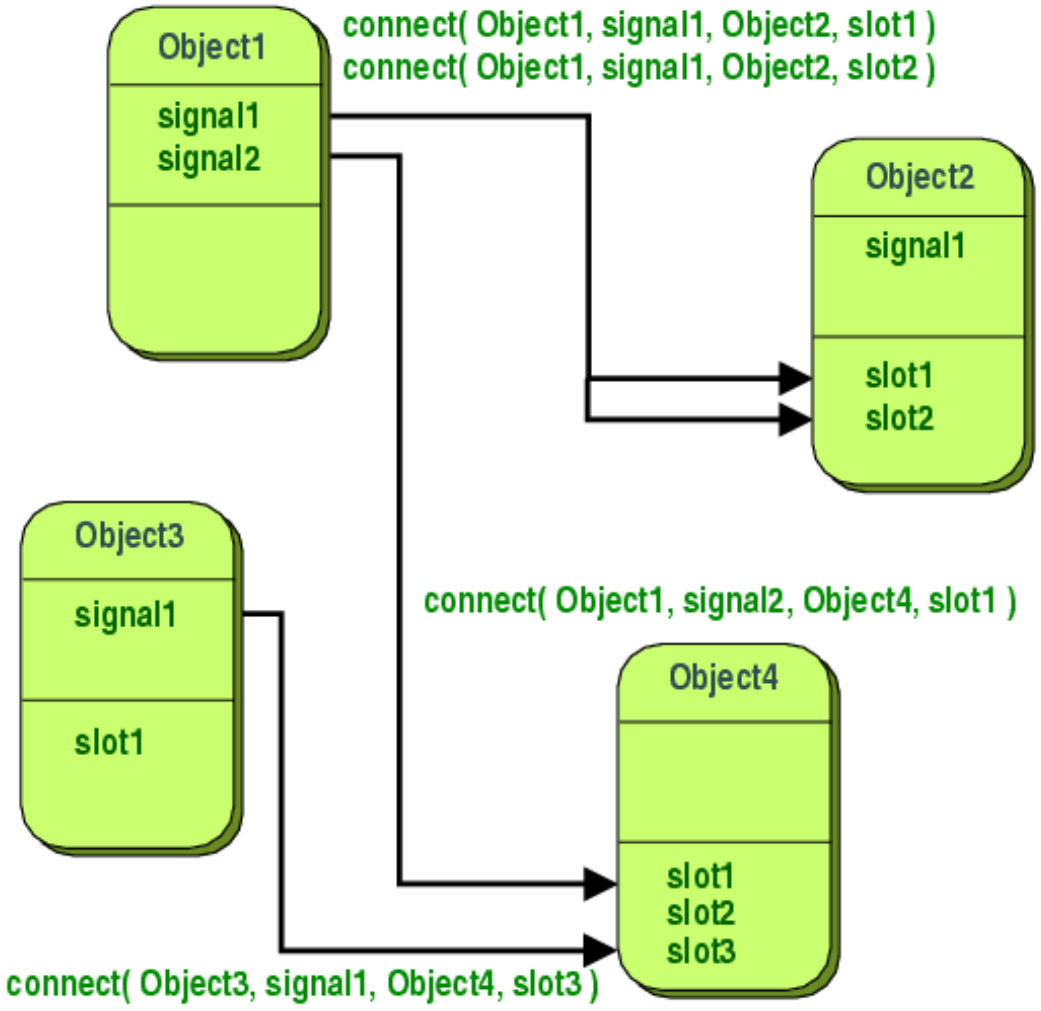
Signale und Slots in Qt

- Signale und Slots können für die Kommunikation zwischen Objekten genutzt werden
- Sehr mächtiges Feature in Qt
- Gut geeignet für die Interaktion mit graphischen Objekten
- Benachrichtigung bei Ereignissen

  Zum Beispiel: Ein Button wurde gedrückt
- Viele Klassen in Qt besitzen bereits Signale und Slots, die miteinander oder mit eigenen Signalen und Slots verbunden werden.

Signale und Slots

Schema



Signale und Slots — Textuelles Beispiel

- Signale zeigen an, dass sich etwas im Objekt verändert hat
  Signal: Daten sind verfügbar

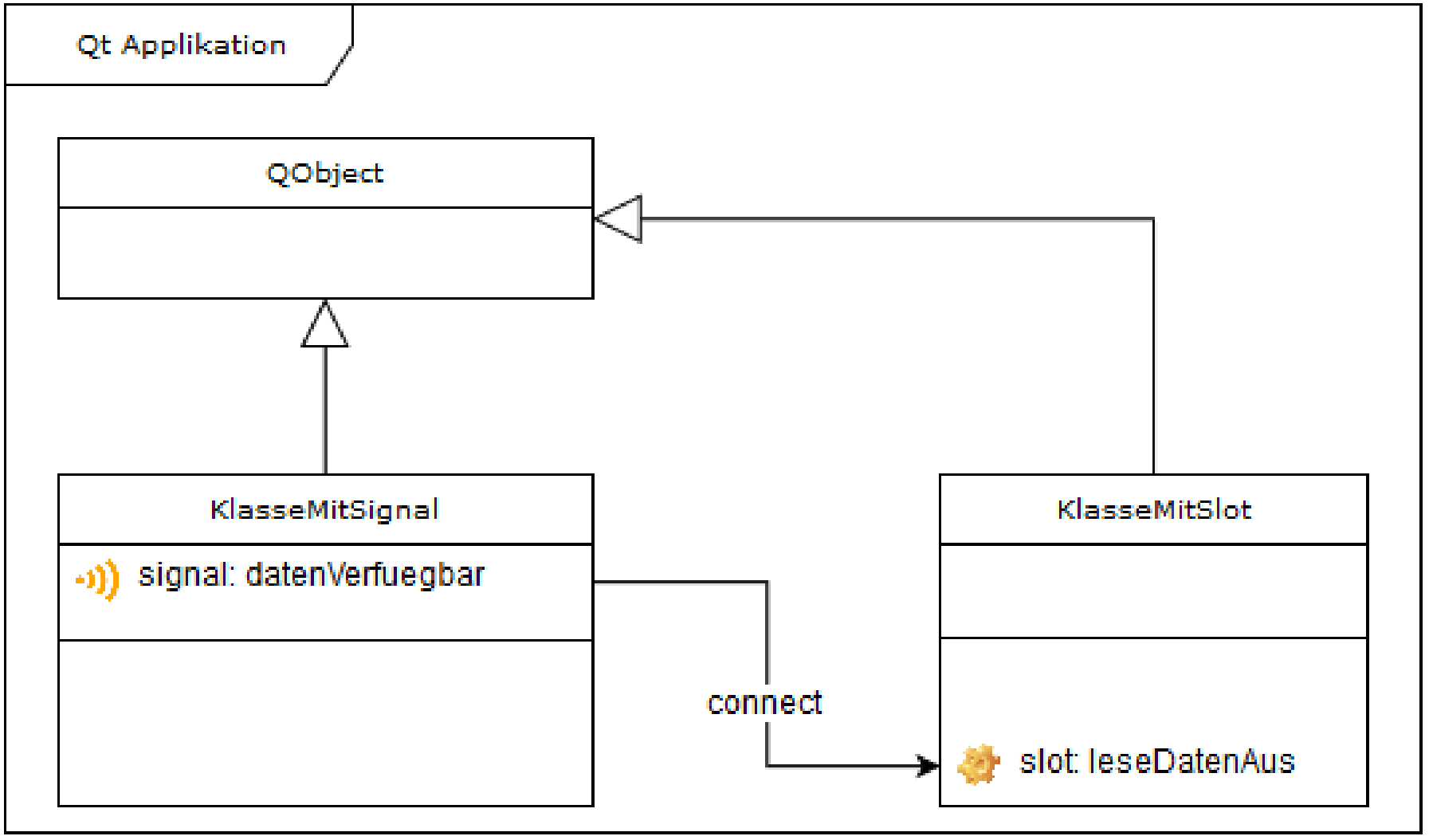
- Slots sind Funktionen bzw. Methoden, die mit Signalen verbunden werden können
  Slot: Lese Daten aus

- Mit dem Befehl `QObject::connect(...)` können Signale mit Slots verbunden werden
  Verbinde das Signal „Daten sind verfügbar“ mit dem Slot „Lese Daten aus“

→ Wenn das Signal: Daten sind verfügbar emittiert wird, wird der Slot „Lese Daten aus“ ausgeführt

Signale und Slots

Klassendiagramm



Signale und Slots

Klasse mit Signal

Listing 11: code/SignalSlot/klassemitsignal.h

```
1  #ifndef KLASSEMITSIGNAL_H
2  #define KLASSEMITSIGNAL_H
3
4  #include <QObject>
5
6  class KlasseMitSignal : public QObject
7  {
8    Q_OBJECT
9  public:
10   explicit KlasseMitSignal(QObject *parent = 0);
11
12 signals:
13   void datenVerfuegbar();
14 };
15 #endif // KLASSEMITSIGNAL_H
```

Signale und Slots

Klasse mit Slot

Listing 12: code/SignalSlot/klassemitslot.h

```
1  #ifndef KLASSEMITSLOT_H
2  #define KLASSEMITSLOT_H
3
4  #include <QObject>
5
6  class KlasseMitSlot : public QObject
7  {
8    Q_OBJECT
9  public:
10   explicit KlasseMitSlot(QObject *parent = 0);
11
12 public slots:
13   void leseDatenAus(); //Implementierung in klassemitslot.cpp
14 };
15 #endif // KLASSEMITSLOT_H
```

Signale und Slots

Klasse mit Slot

Listing 13: code/SignalSlot/klassemitslot.cpp

```
1  #include "klassemitslot.h"
2  #include <iostream>
3
4  KlasseMitSlot::KlasseMitSlot(QObject *parent) : QObject(parent)
5  {
6
7  }
8
9  void KlasseMitSlot::leseDatenAus()
10 {
11   std::cout << "leseDatenAus wurde aufgerufen";
12   std::cout.flush();
13 }
```

Signale und Slots — Verbindung

Auszug der main.cpp:

Listing 14: code/SignalSlot/main.cpp

```
1                    // Vorher nicht vergessen:
2   KlasseMitSignal instanzMitSignal; // #include "klassemitsignal.h"
3   KlasseMitSlot instanzMitSlot;   // #include "klassemitslot.h"
4
5   //Verbinden von Signal und Slot
6   QObject::connect(&instanzMitSignal, SIGNAL(datenVerfuegbar()),
7                    &instanzMitSlot, SLOT(leseDatenAus()));
8
9   //Signal aufrufen
10  emit instanzMitSignal.datenVerfuegbar();
```

Signale und Slots

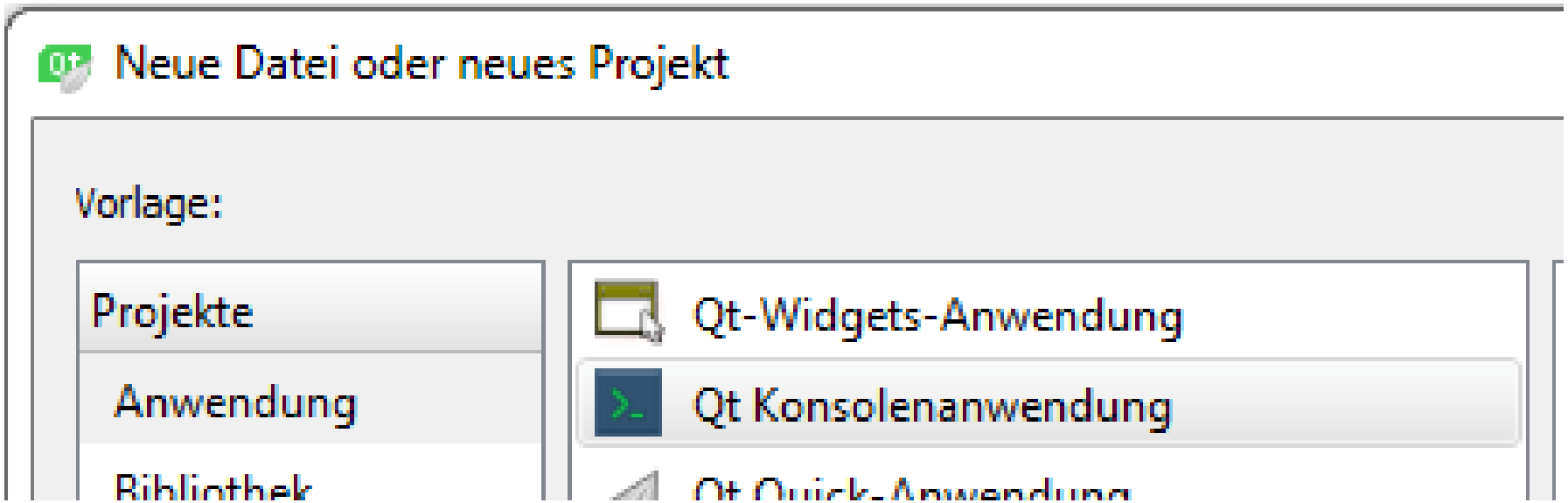
Regeln

Wichtige Regeln

- Die Qt-Applikation muss gestartet sein
- Objekte, die Signale und Slots verwenden möchten, müssen…
  - …Ableitungen des Objekts **`QObject`** sein
  - …das Makro **`Q_OBJECT`** enthalten
- Die verbundenen Signale und Slots müssen die gleichen Ein- und Ausgabeparameter besitzen

Anlegen eines Projekts

1. Neues Projekt
2. Qt Konsolenanwendung
3. Projekt einrichten
   - 3.1 Projektname wählen
   - 3.2 Zielverzeichnis auswählen
   - 3.3 Falls erforderlich, Compiler auswählen



Qt-Bibliothek einbinden

- Bibliothek von Qt ist Modular
- In der pro-Datei können Qt-Bibliotheken hinzugefügt werden:

  QT += *Name der Qt-Bibliothek*
- oder ausgeschlossen werden:

  QT -= *Name der Qt-Bibliothek*

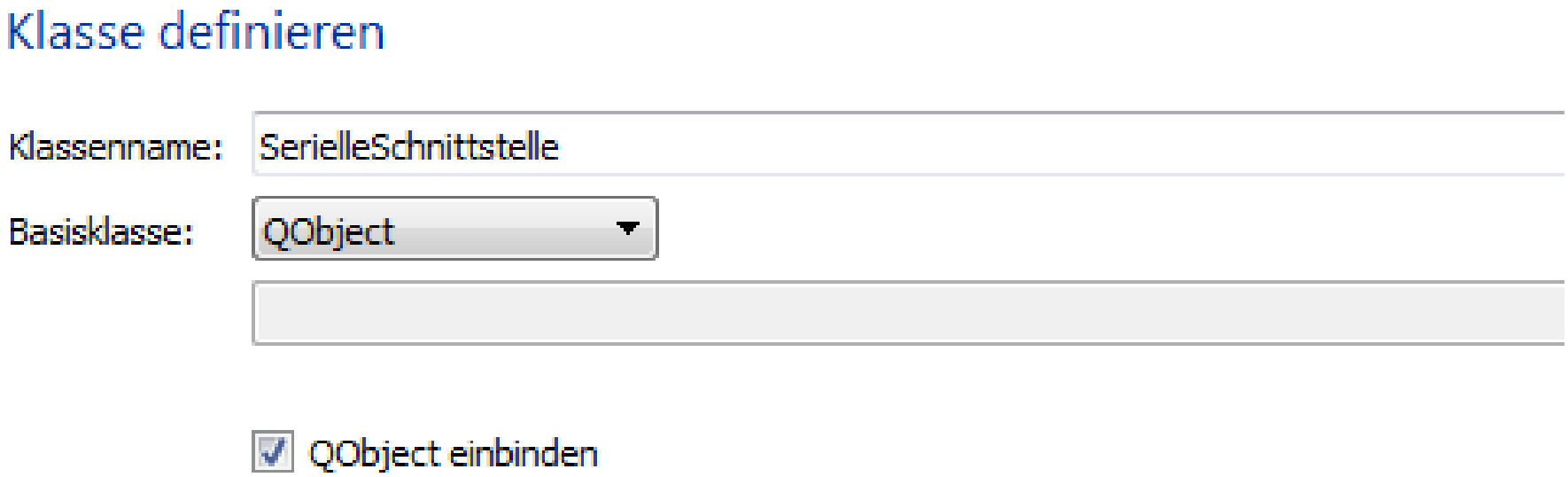
Für die Benutzung der seriellen Schnittstelle ist der folgende Eintrag in der Projektdatei (*.pro) erforderlich:

Listing 15: code/Serial/Serial.pro

```
1 QT += serialport #Bibliothek fuer serielle Schnittstelle
```

Eigene Klasse in Qt erstellen

- Klassen können automatisiert in Qt erstellt werden
- Datei→Neu…→C++-Klasse
- Als Basisklasse: QObject auswählen



Klasse erweitern

Klasse `QSerialPort` hinzufügen:

Listing 16: code/Serial/serielleschnittstelle.h

```
1 private:              // Nicht vergessen:
2   QSerialPort m_serialPort; // #include <QSerialPort>
```

Eigene Slots deklarieren:

Listing 17: code/Serial/serielleschnittstelle.h

```
1 public slots:
2   void readFromSerialPort();
3   void writeToSerialPort(const char *dataToWrite);
```

Implementierung von readFromSerialPort()

Listing 18: code/Serial/serielleschnittstelle.cpp

```
1  void SerielleSchnittstelle::readFromSerialPort()
2  {
3    char speicher[100]; // Speicher anlegen
4    memset(speicher, 0, sizeof(speicher));
5
6    m_serialPort.read(speicher, 100);
7
8    //etwas mit dem Inhalt von speicher machen
9    std::cout << speicher; // Zum Beispiel ausgeben
10   std::cout.flush();
11 }
```

Implementierung von writeToSerialPort()

Listing 19: code/Serial/serielleschnittstelle.cpp

```
1 void SerielleSchnittstelle::writeToSerialPort(const char*
      dataToWrite)
2 {
3   m_serialPort.write(dataToWrite);
4   m_serialPort.flush();
5 }
```

Einrichten des SerialPort-Objekts

Listing 20: code/Serial/serielleschnittstelle.cpp

```
1  SerielleSchnittstelle::SerielleSchnittstelle(QObject *parent) :
       QObject(parent)
2  {
3    m_serialPort.setPortName("COM5");
4    m_serialPort.setBaudRate(9600);
5
6    connect(&m_serialPort, SIGNAL(readyRead()),
7        this, SLOT(readFromSerialPort()));
8
9    if(!m_serialPort.open(QIODevice::ReadWrite)){
10     std::cout << "Could not open Interface\n";
11     std::cout.flush();
12   }
```

Teil IV

Projektarbeit

Anmeldung Projektmappe per E-Mail

Listing 21: Betreff der E-Mail

```
1 Anmeldung API-Projektmappe
```

Listing 22: Inhalt der E-Mail

```
1 Vorname:
2 Nachname:
3 Matrikelnummer:
4 GitHub-Benutzer:
5 GitHub-Team:
6 GitHub-Repository-URL: .git
7 GitHub-Wiki-URL: .wiki.git
```

An die folgende E-Mail-Adresse:
api-iff@tu-braunschweig.de

Anmeldung Projektarbeit per E-Mail

Anmeldung

- Jeder muss sich bis zum 30.06.2017 anmelden.
- Für die E-Mail sollen die Inhalte von Listing 21 und Listing 22 verwendet werden.
- Oder direkt den Link verwenden: api-iff@tu-braunschweig.de

Abmeldung

Frist	11. Juli 2017
E-Mail-Adresse	api-iff@tu-braunschweig.de
Betreff	Abmeldung API-Projektmappe
Inhalt	Siehe Listing 22

2. Zyklus vom API-Spiralmodell beginnen

Aufgabe 1

Beginnen Sie den zweiten Zyklus des API-Spiralmodells.

Fragen?

Gibt es noch Fragen?