

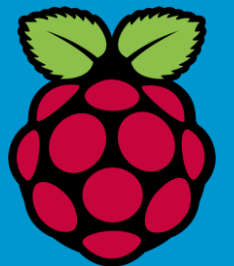


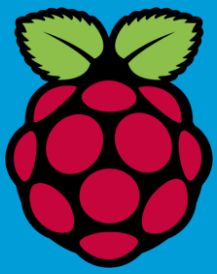
The
#NOSSUED
Software Entwicklungs Open Space - Karlsruhe 29/30 Juni 2019

SOFTWARE ENTWICKLUNGS OPEN SPACE

Using .NET with the Raspberry Pi

.NET User Group Karlsruhe 2018
Frank Pfattheicher





Ich

Frank Pfattheicher

Freier Softwareentwickler

Automatisierungstechnik, Azure, Embedded

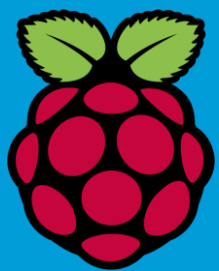
.NET UserGroup Karlsruhe

mail fpf@dotnet-ka.de

mobil 0172-7207196

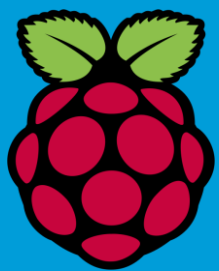
twitter [@fpf_baden](https://twitter.com/fpf_baden)

Skype [fpf@itbaden.de](https://www.skype.com/en/contacts/itbaden.de/fpf/)



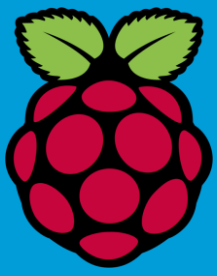
Bestellt – Da 😊





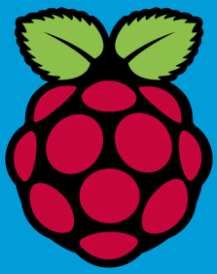
Jetzt kann ich loslegen...





...aber habe „gerade“ keine Zeit

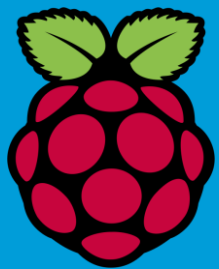




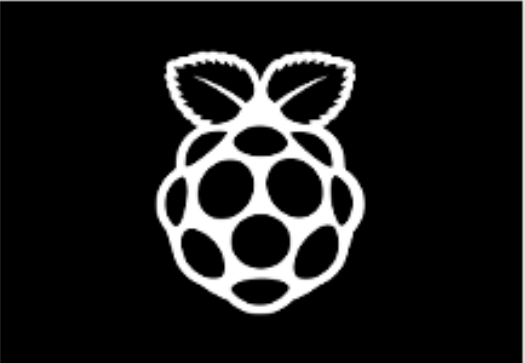
Keine Ausreden – los geht's !

Was brauche ich noch?


- Hardware (Raspberry Pi)
 - SD-Karte, Netzteil
 - HDMI-Kabel, Monitor
 - Tastatur, Maus
 - Optional USB-Hub
- Hardware (Entwicklungssystem)
 - Windows- oder Linux-PC
 - SD-Kartenleser
- Software
 - Win32 DiskImager (Windows)
 - Image für SD-Karte
 - Entwicklungsumgebung



Was nehme ich?



NOOBS



RASPBIAN

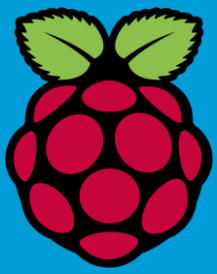
Windows 10 IoT Editions

Windows 10 IoT for industry devices
Desktop Shell, Win32 apps, Universal apps and drivers
Minimum: 1 GB RAM, 16 GB storage
x86/x64

Windows 10 IoT for mobile devices
Modern Shell, Mobile apps, Universal apps and drivers
Minimum: 512 MB RAM, 4 GB storage
ARM

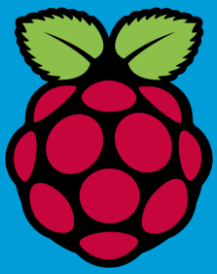
Windows 10 IoT Core
Universal Apps and Drivers
No shell or MS apps
Minimum: 256MB RAM, 2GB storage
x86/x64 or ARM

+



Nichts vorbereitet, alles live !

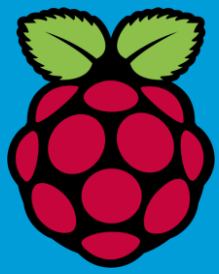
- Image auf SD-Karte übertragen
(dauert ca. 5 bis 15 Minuten)
- Hardware aufbauen



Warum nicht Windows 10 IoT Core ?

Runtime Image. Subject to the requirements in Section 3 and restrictions in Section 4, Microsoft hereby grants to you a royalty-free, worldwide, non-exclusive, personal, non-transferable, non-assignable, limited license to install a Runtime Image into an Embedded System and distribute your Embedded System to End Users.

No Distribution of Software as Stand-alone Product. You must not advertise, provide a separate price for, or otherwise market or distribute the Software, or any part of the Software, as a separate item from an Embedded System.



Raspbian

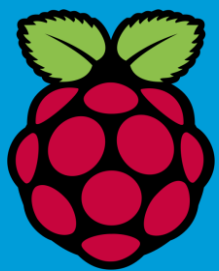
Versionen

Wheezy – Debian 7

Jessie – Debian 8 (September 2015)

Stretch – Debian 9 (August 2017)

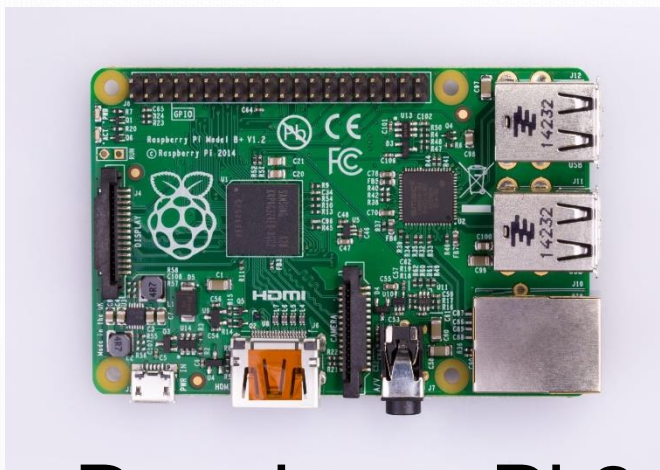
<https://en.wikipedia.org/wiki/Raspbian>



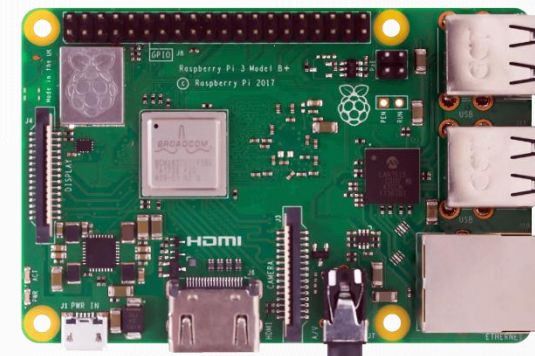
Hardware-Varianten



Raspberry Pi 1



Raspberry Pi 2



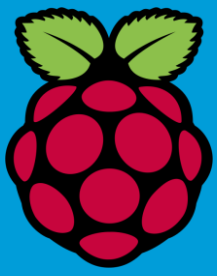
Raspberry Pi 3



Zero

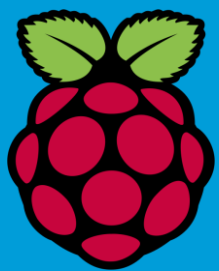


Compute Module



Erster Start

- SD-Karte einsetzen, Stromversorgung anstecken / einschalten
- Resized root filesystem. Rebooting in 5 seconds...
- Vier Himbeeren :-)
- Welcome to the Raspberry Pi Desktop – Setup Assistant
- Land, Sprache, Tastatur und Zeitzone einstellen
- Neues Passwort vergeben
- Netzwerk verbinden (optional)
- Updates und gewählte Sprache installieren (dauert etwas...)
- Fertig. Neustart



Und jetzt?

Grundlegende Einstellungen

Raspberry-Pi-Konfiguration

System Schnittstellen Leistung Lokalisierung

Passwort: Passwort ändern...

Hostname: raspberrypi

Boot: ☒ Zum Desktop ☐ Zum CLI

Automatische Anmeldung: ☐ Als Benutzer 'pi' anmelden

Netzwerk beim Booten: ☐ Auf Netzwerk warten

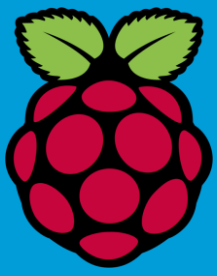
Startbildschirm: ☒ Aktiviert ☐ Deaktiviert

Auflösung: Auflösung festlegen...

Übertastung: ☒ Aktiviert ☐ Deaktiviert

Pixelverdopplung: ☐ Aktiviert ☒ Deaktiviert

Abbrechen OK



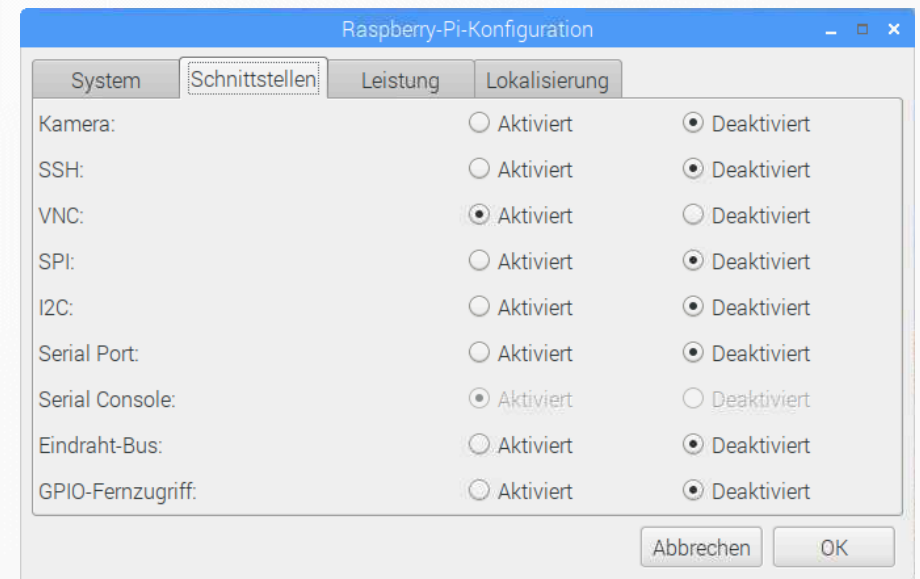
Schnittstellen

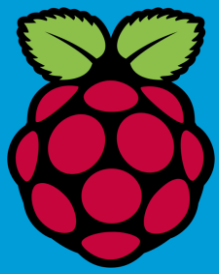
Externer Zugriff

- **SSH** – Secure Shell – Remotezugriff auf Kommandozeile
- **VNC** – Remotezugriff auf GUI

Geräteinterne Kommunikation zwischen Schaltungsteilen

- **SPI** – Serial Peripheral Interface
- **I2C** – Inter-Integrated Circuit
- **Eindraht-Bus (OneWire)**

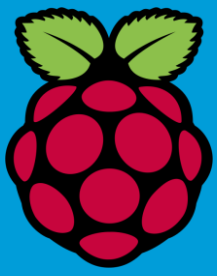




Pi-Complete

Der Raspberry Pi ist jetzt grundsätzlich bereit

Jetzt kommt die Entwicklungsumgebung



Entwicklungsumgebung

Windows

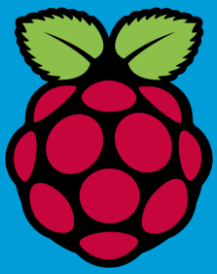
- VisualStudio 2017
- Visual Studio Code
- JetBrains Rider

Linux

- Visual Studio Code
- JetBrains Rider

Raspberry Pi

- Visual Studio Code
- Mono Develop (nur Mono = Full Framework V4.5)



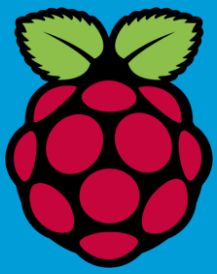
Full Framework vs. Mono vs. Core

Mono

- Kompatibel zu Full Framework \leq V4.5
- Über MonoDevelop direkt auf dem Pi
- Full Framework ist ***deprecated***

Core

- Identisch mit Windows-Version
- Keine GUI Unterstützung



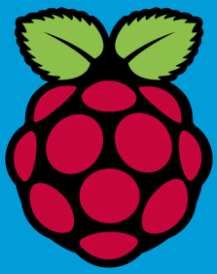
Windows - VisualStudio 2017

Projekt erstellen (Console)

File – New – Project – Visual C# / -NET Core – Console App

„F5“

Läuft – was muss jetzt getan werden?

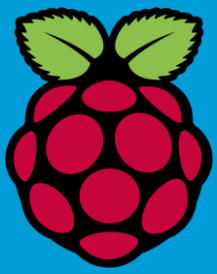


Windows - VisualStudio 2017

Anpassung der Zielplattform

Edit csproj – RuntimeIdentifiers hinzufügen

```
<PropertyGroup>  
  <OutputType>Exe</OutputType>  
  <TargetFramework>netcoreapp2.1</TargetFramework>  
  <RuntimeIdentifiers>linux-arm</RuntimeIdentifiers>  
</PropertyGroup>
```

Publish...

Anwendung packen

```
dotnet publish -c Release -r linux-arm --self-contained
```

-c Release

Gibt die Build-Konfiguration an

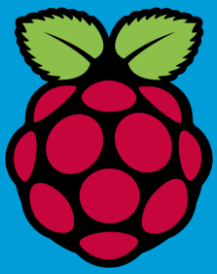
-r linux-arm

Zielplattform

--self-contained

Das Framework wird lokal hinzugefügt

Keine Installation notwendig



„Deployment“

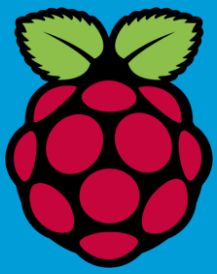
VNC Dateiübertragung

- Zielordner auf dem Raspberry Pi einstellen !
- Programm als ausführbar markieren

`chmod +x Programm`

Alternativen

- Ordner auf dem Pi freigeben (Samba muss installiert werden)
- Ordner auf dem PC freigeben



Windows - VisualStudio Code

Projekt erstellen (Console)

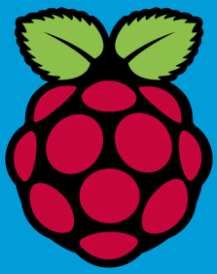
Projekt-Ordner anlegen – in VSCode öffnen

Ansicht - Integriertes Terminal

Im Terminalfenster:

```
dotnet new console
```

„F5“ - Läuft

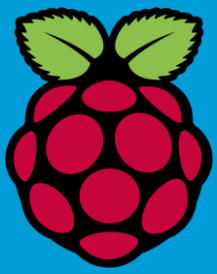


Windows - Rider

Projekt erstellen (Console)

New Solution - .NET Core – Console Application

„F5“ - Läuft

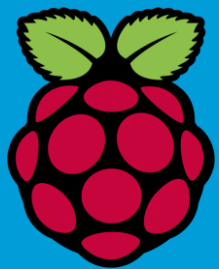


Essen

Hacksteak ?

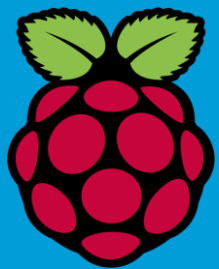
Pause

Noch Fragen?



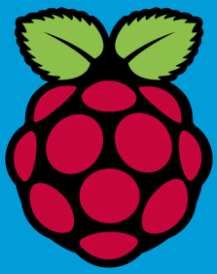
Euphorie

Starten – Läuft ...



Remote-Debugging

... doch nicht ☹



Remote-Debugging

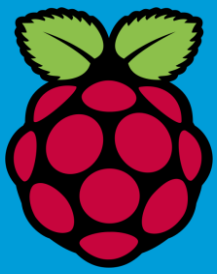
Voraussetzungen auf dem Raspberry Pi

- SSH aktivieren

```
raspi-config
```

- Debugger für linux-arm installiert (VsCode)

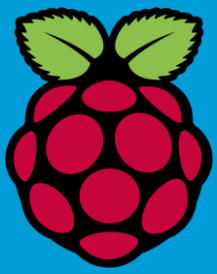
```
curl -sSL https://aka.ms/getvsdbgsh |  
  bash /dev/stdin -r linux-arm -v latest -l ~/vsdbg
```



Remote-Debugging

Voraussetzungen auf dem Entwicklungsrechner

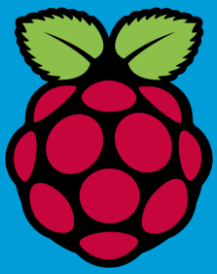
- Visual Studio 2017
 - Nichts weiter 😊
- VsCode
 - Windows: PuTTY installieren
 - Linux: SSH-Key eintragen
 - Neue Konfiguration in launch.json erstellen
 - Debugger muss auf dem Raspberry Pi installiert sein
- Rider
 - Geht NICHT 😞
(RIDER-738 Add support for remote debugging)



Remote-Debugging

Linux - VsCode Konfiguration in launch.json

```
{
  "name": ".NET Core Remote Attach",
  "type": "coreclr",
  "request": "attach",
  "processId": "${command:pickRemoteProcess}",
  "pipeTransport": {
    "pipeCwd": "${workspaceFolder}",
    "pipeProgram": "/usr/bin/ssh",
    "pipeArgs": [
      pi@<IpAddr>
    ],
    "debuggerPath": "~/vsdbg/vsdbg"
  }
},
```



Remote-Debugging

Linux – SSH Credentials anlegen

```
$ ssh-keygen -t rsa -b 2048
```

Generating public/private rsa key pair.

Enter file in which to save the key (/home/username/.ssh/id_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

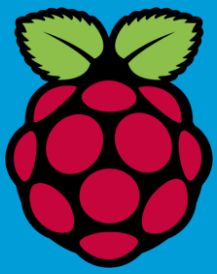
Your identification has been saved in /home/username/.ssh/id_rsa.

Your public key has been saved in /home/username/.ssh/id_rsa.pub.

Copy your keys to the target server:

```
$ ssh-copy-id pi@<RaspPiAddress>
```

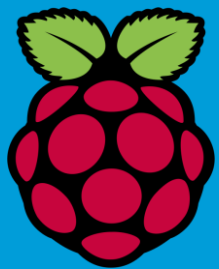
id@server's password: raspberry



Remote-Debugging

Windows - VsCode Konfiguration in launch.json

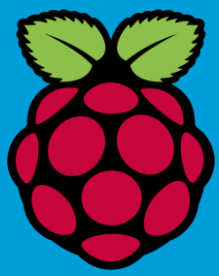
```
{
  "name": ".NET Core Remote Attach",
  "type": "coreclr",
  "request": "attach",
  "processId": "${command:pickRemoteProcess}",
  "pipeTransport": {
    "pipeCwd": "${workspaceFolder}",
    "pipeProgram": "c:\\Program Files\\PuTTY\\plink.exe",
    "pipeArgs": [
      "-pw", "raspberry",
      pi@<IpAddr>
    ],
    "debuggerPath": "~/vsdbg/vsdbg,,
  },
},
```



Ready

Das WIE ist jetzt geklärt.

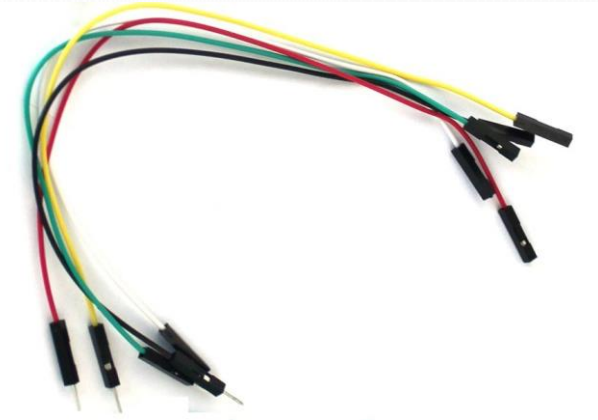
Jetzt kommt das WAS !



Projektideen

Hardware und das mit dem Löten

- Oder Minimalinvasiv über



Hardware aber bitte ohne Löten

- GrovePi

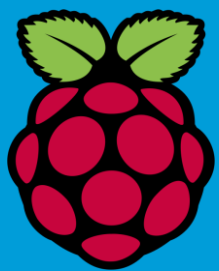
Oder ganz ohne zusätzliche Hardware

- Infotafel (Kiosk-Display)



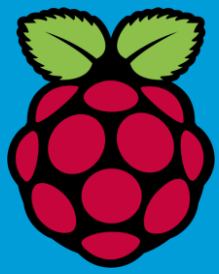


Alternate Function							Alternate Function		
	3.3V PWR	1			2	5V PWR			
I2C1 SDA	GPIO 2	3			4	5V PWR			
I2C1 SCL	GPIO 3	5			6	GND			
OneWire	GPIO 4	7			8	GPIO 14	UART0 TX		
	GND	9			10	GPIO 15	UART0 RX		
	GPIO 17	11			12	GPIO 18			
	GPIO 27	13			14	GND			
	GPIO 22	15			16	GPIO 23			
	3.3V PWR	17			18	GPIO 24			
SPI0 MOSI	GPIO 10	19			20	GND			
SPI0 MISO	GPIO 9	21			22	GPIO 25			
SPI0 SCLK	GPIO 11	23			24	GPIO 8	SPI0 CS0		
	GND	25			26	GPIO 7	SPI0 CS1		
	Reserved	27			28	Reserved			
	GPIO 5	29			30	GND			
	GPIO 6	31			32	GPIO 12			
	GPIO 13	33			34	GND			
SPI1 MISO	GPIO 19	35			36	GPIO 16	SPI1 CS0		
	GPIO 26	37			38	GPIO 20	SPI1 MOSI		
	GND	39			40	GPIO 21	SPI1 SCLK		



GPIO – Was ist denn das ?





Strom und Spannung

Warum 5 Volt und 3,3 Volt?

5 Volt

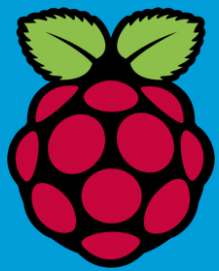
- Stromversorgung über USB

3,3 Volt

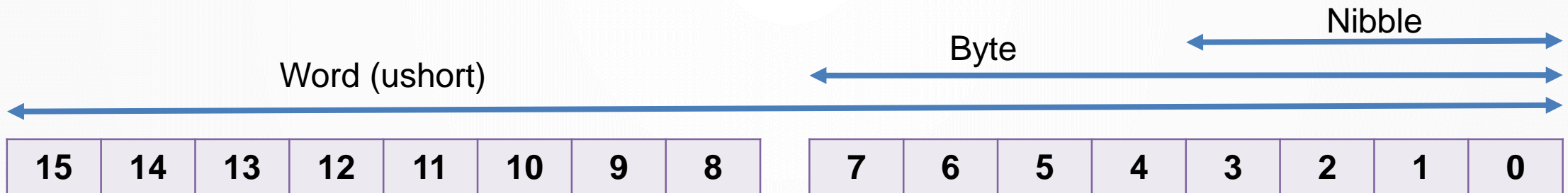
- Stromverbrauch
- Geschwindigkeit

Lösung

- Pegelwandler
- Relais-Boards

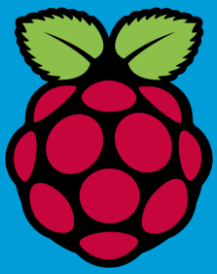


Bits und Bytes

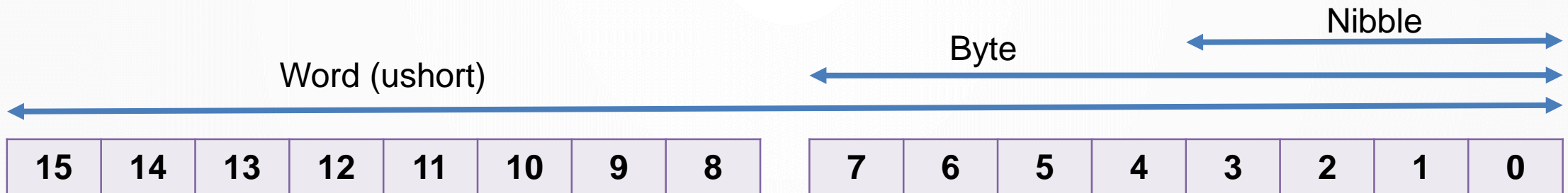


```
public bool IsSet(ushort word, int bit)
{
}

public ushort Set(ushort word, int bit, bool value)
{
}
}
```

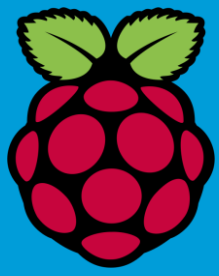


Bits und Bytes



```
public bool IsSet(ushort word, int bit)
{
    return (word & (ushort)(1 << bit)) != 0;
}

public ushort Set(ushort word, int bit, bool value)
{
    return value
        ? (ushort) (word | (ushort) (1 << bit))
        : (ushort) (word & ~(ushort) (1 << bit));
}
```

Interop

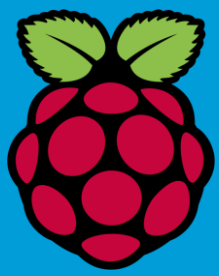
Wie unter Windows

```
[DllImport("libc.so.6")]
public static extern int open(string file, int mode /*, int permissions */);

[DllImport("libc.so.6")]
public static extern int close(int file);

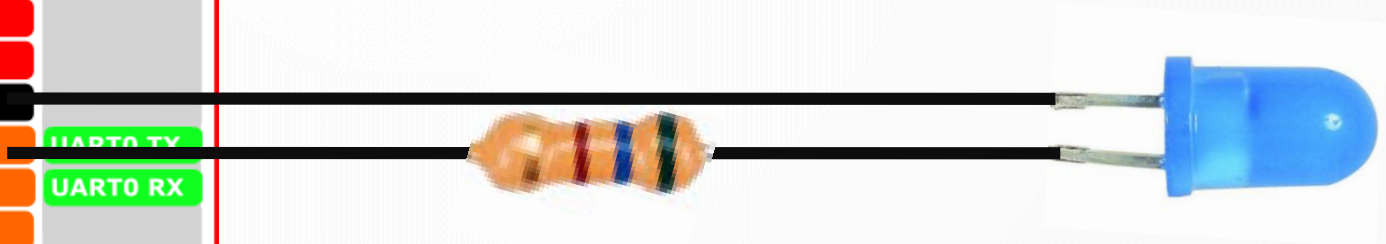
[DllImport("libc.so.6", SetLastError = true)]
public static extern int read(int file,
                             [MarshalAs(UnmanagedType.LPArray)] byte[] addr, int count);

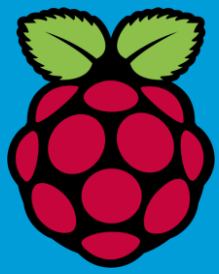
[DllImport("libwiringPi.so", EntryPoint = "pinMode")] //Uses Gpio pin numbers
public static extern void PinMode(int pin, int mode);
```



IoT Hello World

Alternate Function					Alternate Function
	3.3V PWR	1		2	5V PWR
I2C1 SDA	GPIO 2	3		4	5V PWR
I2C1 SCL	GPIO 3	5		6	GND
OneWire	GPIO 4	7		8	GPIO 14
	GND	9		10	GPIO 15
	GPIO 17	11		12	GPIO 18
	GPIO 27	13		14	GND
	GPIO 22	15		16	GPIO 23
	3.3V PWR	17		18	GPIO 24
SPI0 MOSI	GPIO 10	19		20	GND
SPI0 MISO	GPIO 9	21		22	GPIO 25
SPI0 SCLK	GPIO 11	23		24	GPIO 8
	GND	25		26	GPIO 7
	Reserved	27		28	Reserved
	GPIO 5	29		30	GND
	GPIO 6	31		32	GPIO 12
	GPIO 13	33		34	GND
SPI1 MISO	GPIO 19	35		36	GPIO 16
	GPIO 26	37		38	GPIO 20
	GND	39		40	GPIO 21





Autostart

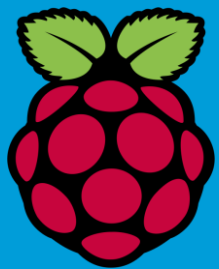
Autostart (Windows shell:startup)

```
crontab -e  
@reboot /usr/pi/name
```

Daemon (Windows Service)

Servicebeschreibung erstellen: `/lib/systemd/system/name.service`

```
[Service]  
Type=simple  
ExecStart=/usr/pi/name  
[Install]  
WantedBy=multi-user.target  
  
systemctl start name
```



Tipps und Hinweise

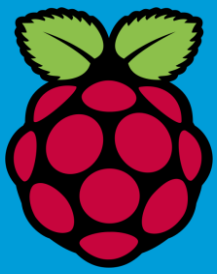
Ausschalten ☹

SD-Karten

Dateifreigabe (Samba)

Stromversorgung

Zubehör



Danke

Pinvoke

sudo

armhf

apt-get

Docker

PWM

Fragen

--self-contained

Mono

3v3 vs. 5V

Links

Folien / Skripte

<https://github.com/FrankPfattheicher/RaspiDotnet>

Raspberry Pi

<https://www.raspberrypi.org/>

Win32DiskImager

<https://sourceforge.net/projects/win32diskimager>

elinux.org

https://elinux.org/RPi_Hub

EXP TECH

<https://www.exp-tech.de/module/raspberry-pi/>

PiXtend

<https://www.pixtend.de/>

Revolution Pi

<https://revolution.kunbus.de/>