

gardenctl | Gartensteuerung

Inhaltsverzeichnis

Gartensteuerung

Einleitung	2
Beispiel gardenctl	2
Hardware	2
Schaltplan	3
Bedienung	3
Programm starten	3
Fernwartung	3
Dauerbetrieb	3
Wartung	3
Stromausfall	3

Programmstart

Startoptionen	4
Konfigurationsdateien	4

Programm bedienen

Event-Loop	5
Blockanzeige	5
Fortlaufende Anzeige	5
Menu	5
Devices	6
Events	7
Testprogramm devtest	8

Datei- und Ordnerübersicht

Linkbibliotheken	9
Dateien	9

GNU General Public License

Gartensteuerung

Einleitung

Das Programmbeispiel `gardenctl` zeigt wie aus vordefinierten Modulen Steuerungen zusammengestellt und betrieben werden können. Die Steuerung wird aus Events, Devices und einer Loop zusammengestellt. Events und Devices werden durch Konfigurationsdateien beschrieben. Damit können zum Beispiel Sensoren einfach durch Änderung einer Konfigurationsdateien ausgetauscht werden.

Zum Testen der Funktionen von Events und Devices während der Entwicklung und zur Fehlersuche gibt es ein umfangreiches Testprogramm `devtest` das direkt die verwendeten Konfigurationsdateien einlesen kann.

Das Grundkonzept der Steuerungen und deren Konfiguration sind in der Dokumentation zu `devtest` genau beschrieben. Siehe: : [2_devtest.pdf](#)

Beispiel `gardenctl`

Das Programm `gardenctl` zeigt den Aufbau einer Steuerung für den Garten.

Das Programm kann zeit- und/oder programmgesteuert:

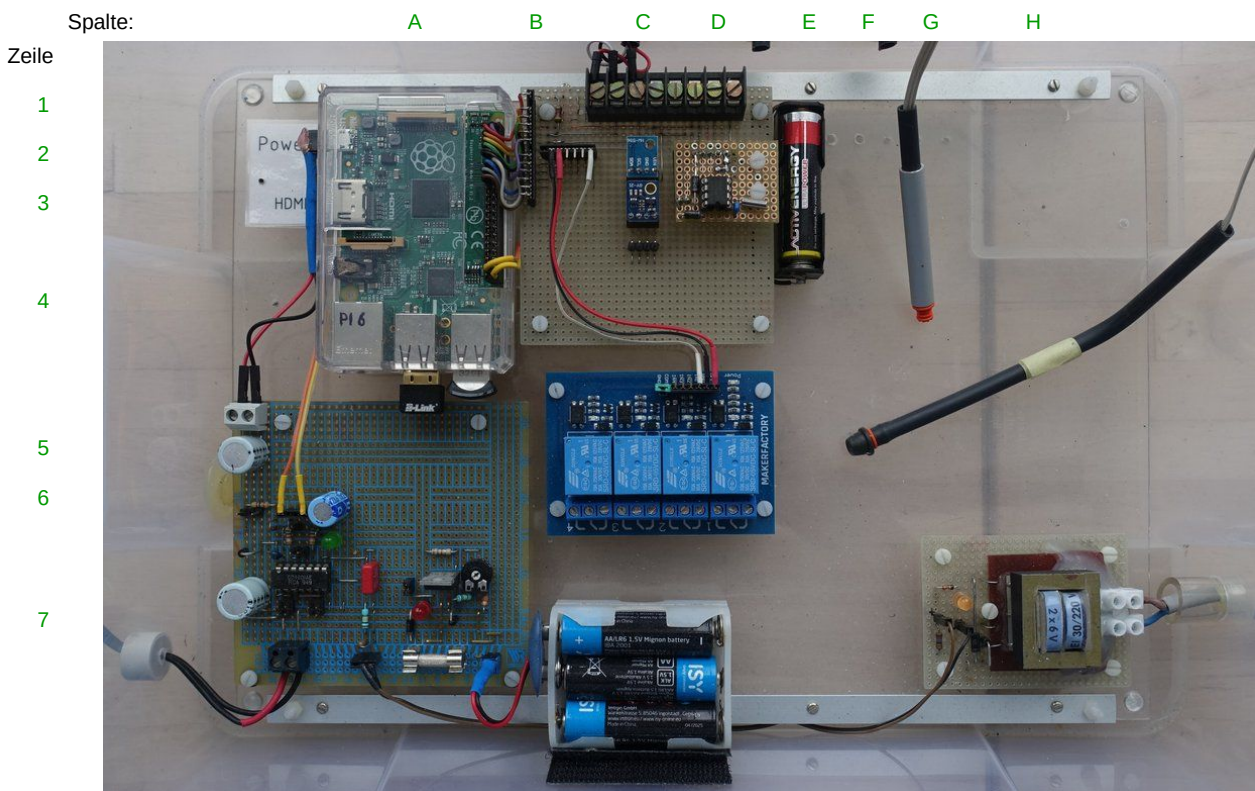
- ▷ Sensoren lesen
- ▷ Relays schalten
- ▷ Steuerprogramme ausführen

Weitere Programmfunktionen:

- ▷ Automatischer Betrieb mit Fernbedienung über Lan/Wlan
- ▷ Ergebnisse im Terminal anzeigen
- ▷ Ergebnisse und Ereignisse in eine Logdatei schreiben
- ▷ ...
- ▷ Beispiel: Heizung im Frühbeet steuern
- ▷ Beispiel: Zeitgesteuerte Bewässerung

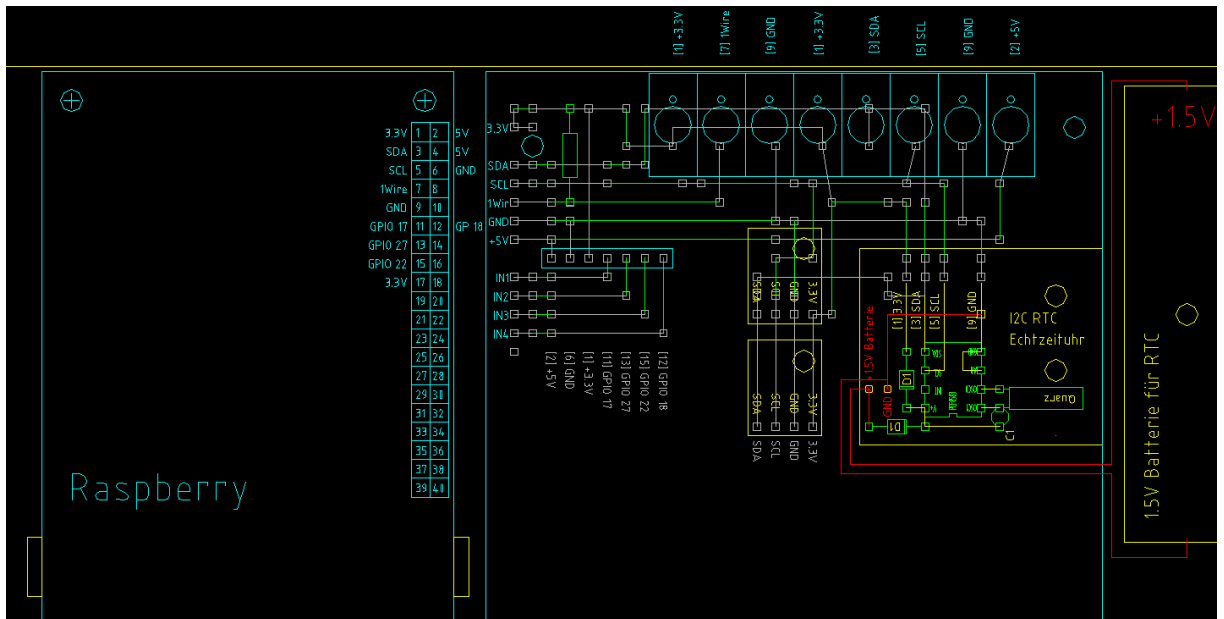
Alle Hardware-/Treibermodule für Sensoren und Relais können leicht getauscht werden, ohne Änderungen an der Steuerung.

Hardware



Spalte/Zeile	Beschreibung
A3	Raspberry Pi
A7	In Arbeit: Umschalter von Netz auf Notstrom
C1	1-Wire und I2C Anschlüsse
C1 und C3	I2C Sensoren BMP180 und Si7021
C5	Relais
C7	Batterien zum Herunterfahren bei Netzausfall
D3	Echtzeituhr PCF8583 I2C
E3	Echtzeituhr Batterie
F5 und G4	1-Wire Temperatursensoren
H7	In Arbeit: Stromsensor (Trafo) für Netzspannung

Schaltplan



Bedienung

Auf dem Raspberry Pi wird das Programm `gardenctl` nach dem Einschalten automatisch gestartet und kann dann von einem beliebigen Netzwerk-Terminal aus mit `ssh` bedient werden. Siehe Dauerbetrieb.

Programm starten

Das Programm wird auf jedem Rechner - remote oder lokal - immer mit Befehl `gardenctl` gestartet. Dabei wird automatisch der richtige Programmmodus gewählt. Eine auf Pi laufende Steuerung wird dabei niemals unterbrochen. Auf einem remote PC wird die Verbindung mit dem Terminalmultiplexer `screen` hergestellt.

Fernwartung

Für den Fernzugriff mit `ssh` wird der Terminalumschalter `screen` verwendet. Damit kann das Programm am Pi im Hintergrund auch ohne Ein- und Ausgabe-Terminal betrieben werden.

Zur einfachen Fernsteuerung von `gardenctl` wird der Terminalumschalter `screen` über das Hilfsprogramm `screenstart` aus `c/bin/screenstart` aufgerufen. Damit gibt es sicher nur eine Programminstanz von `gardenctl`.

Dauerbetrieb

Zur Sicherheit ruft ein cron - Job jede Minute den Startbefehl `'screenstart'` auf. Der Befehl prüft, ob die Steuerung `gardenctl` wirklich läuft. Im Fehlerfall wird eine neue `'screen'`-Sitzung mit `'gardenctl -1'` neu gestartet.

Wartung

Der Austausch von Sensoren/Aktoren ist durch modularen Aufbau der Steuersoftware sehr einfach. Es muss nur die Konfigurationsdatei für Devices angepasst werden. Siehe: [2_devtest.pdf](#)

Stromausfall

An einer sauberen Lösung wird noch gearbeitet!

Programmstart

Startoptionen

Zum Testen wird das Programm auf PC oder Raspberry Pi mit

```
gardenctl -2
```

aufgerufen. Auf dem PC werden die Hardwarefunktionen der Devices simuliert.

Hilfe anzeigen:

```
gardenctl -h
Aufruf: gardenctl [OPTIONS] [STARTOPT]
OPTIONS:
-h Help
-m mtrace on. Speicherüberwachung
-d Programm im Debugmodus starten
-x runTest() rufen

Der normale Aufruf von 'gardenctl' erfolgt ohne [STARTOPT].
Das Programm wird mit Hilfe von 'screenstart' neu gestartet.
'screenstart' startet zuerst den Terminalmultiplexer 'screen'
und dann das Programm mit 'gardenctl -1'.

STARTOPT:
-1 Startmodus -1: Reserviert für 'screen'
-2 Startmodus -2: Nur Testmodus ohne 'screen'
```



Konfigurationsdateien

gardenctl.conf : Basiskonfiguration
garden.events : Eventeinstellungen für Programm gardenctl. Beschreibung siehe: [2_devtest.pdf](#)
garden.devices : Geräteeinstellungen für die Events Beschreibung siehe: [2_devtest.pdf](#)

Das Programm sucht nach der Konfiguration gardenctl.conf:

1. Versuch im Homeordner: `~/config/gardenctl/gardenctl.conf`
2. Versuch Programmordner `_gardenctl`: `./_gardenctl/gardenctl.conf`

- Syntax: einfaches C
- 'gardenctl.conf.bak' ist eine Vorlage für 'gardenctl.conf'

Für gardenctl sollte die Option 1.Versuch verwendet werden. Damit wird verhindert, dass bei Programmupdates die Konfigurationen überschrieben werden.

```
// -----
// Konfiguration für gardenctl
// Version min 0.59
// -----
VarSetGrpFlt(0,1); // Alle Konfigurationsdaten in VarSubGrp 1 speichern

// Konfiguration für Events und Devices
DirDat = ConfigDir; // DirDat ist ConfigDir
EventsDat = "garden.events"; // Dateiname
DevicesDat = "garden.devices"; // Dateiname

// Herunterfahren bei Stromausfall mit GardenDownCheck()
UseDownCheck = 0; // 0/1: GardenDownCheck() verwenden ja/nein

// Logdatei -----
LogLabel = "Scandisk"; // Name des USB-Datentägers für Logdatei
LogDirOpt = ""; // Optionales Dir für Logdatei

LogOn = 1; // 1: Logdatei schreiben
LogErr = 1; // 1: Fehler in Logdatei schreiben

// Watchdog -----
WatchdogSec = 20; // 0: kein Watchdog
// >WATCHDOGMinSec: Bei Unterbrechung der Steuerung von
// mehr als WatchdogSec Programm beenden

// GardenDownCheck -----
DownCheck = 0; // 1: GardenDownCheck()

DeviceGpio = "/dev/gpiochip0"; // Devicepfad GPIO-Device
```

Programm bedienen

Event-Loop

Nach erfolgreichem automatischem Start von **gardenctl** befindet sich das Programm in der Funktion Event-Loop. Im Testmodus kann die Event-Loop mit Befehl **Steuerung fortsetzen** gestartet werden.

In dieser Endlosschleife werden zeit- oder programmgesteuert die konfigurierten Events behandelt. Das Device **DISPLAY** erzeugt automatisch eine Blockanzeige der Events.

Blockanzeige

Exit Terminal	Terminal verlassen. Im automatischen Modus läuft die Steuerung ohne Terminal weiter!
Menu	Das Wartungsmenu aufrufen
Refresh	Anzeige neu aufbauen
Anzeige	Umschalten auf fortlaufende Anzeige

```

gardenctl
Loop[2025-01-23 16:03:03 Uhr]

T1: 14.5 C      | Temp Fruehbeet
RH: 0FF        | Relay Heizung
TA: 15.0 C     | Temp Aussen
P1: 69.9 Pa    | Luftdruck
RW: 0FF        | Relay Wasser

Exit Terminal | Menu | Refresh | Anzeige
  
```

Fortlaufende Anzeige

Zum Testen kann von der Blockanzeige auf eine fortlaufende Anzeige umgeschaltet werden. Dann werden alle Checks der Event-Loop fortlaufend angezeigt.

Der Debugwert bestimmt die Ausführlichkeit der Informationen zu den EventExec() Funktionen.

Das Bild zeigt den Verlauf von EventExec() **Heizung** im Debugmodus 1.

Time:	Echtzeit oder Simulationszeit einstellen
Uhr	Uhr bedeutet Echtzeit
Weiter mit Taste	Simulationszeit
Anzeige	Umschalten auf Blockanzeige
Check	Liste der wartenden Events
Debug	Modus 1,2: Ausführliche Ausgaben
Write Com	Device-Write: Simulationswert schreiben
Stopp	Anzeige anhalten

In EventGetWaitSec() wird die kleinste **CheckTime** alle Events bestimmt. Damit ergibt sich die Wartezeit **WaitSec** für den nächsten Check in Loop().

```

gardenctl
>>> Exec RH CheckTime:00:00:00 > skip
>>> Exec Heizung CheckTime:2025-01-23 17:21:05
EventExec Heizung Cmd=0
HeizungExec Sensor DevLst[8] Relay DevLst[2]
DsRead(2)
T1000=14500
DsDisplayStr(14550)->" 14.5 C"
Sensor=14500 TmpOn=1000 TmpOff=1200
RelayWrite: Pin=12 > 0FF
RelayRead()
HeizungExec > Relay: 0FF
EventSetCheck Heizung Nxt=1 > CheckTime:2025-01-23 17:21:20 CheckCmd:0
>>> Exec TA CheckTime:2025-01-23 17:21:10 > skip
>>> Exec P1 CheckTime:2025-01-23 17:21:10 > skip
>>> Exec RW CheckTime:00:00:00 > skip
>>> Exec TimerRW CheckTime:2025-01-24 00:00:00 > skip

EventGetWaitSec TimeSec:2025-01-23 17:21:05 WaitSecMax:10
Display CheckTime: 2025-01-23 17:21:07
Wait 2 sec auf Taste

Time: Uhr | Anzeige | Check | Debug: 1 | Write Com | Stopp
  
```

Beschreibung siehe: [2_devtest.pdf](#)

Menu

Der Befehl **Menu** stoppt die laufende Steuerung und ruft das Hauptmenu auf. Die Befehle **1-2** bieten einfache Debugfunktionen.

Mit Befehl **3** kann Programm **devtest** gestartet für umfassende Tests werden. Dabei wird Programm **gardenctl** vollständig durch das Testprogramm **devtest** ersetzt. Die Konfigurationen für Events und Devices werden dabei übernommen.

Steuerung fortsetzen	Loop() fortsetzen
1 Devices	Infos zu den Devices
2 Events	Infos zu den Events
3 Testprogramm devtest	Testprogramm mit den Devices und Events starten
Logdateien anzeigen	Logdateien anzeigen oder löschen
Konfiguration bearbeiten	
Crontab bearbeiten	Automatischen Start oder Neustart einstellen
Booteinstellungen bearbeiten	Bootparameter anzeigen und einstellen

```

gardenctl
gardenctl[0.59] Gartensteuerung

Info | Help | + Debug: 0 | Watchdog: 0

1 Steuerung fortsetzen
  1 Devices
  2 Events
  3 Testprogramm devtest

Logdateien anzeigen
Konfiguration bearbeiten
Crontab bearbeiten
Booteinstellungen bearbeiten

Quit | Steuerung anhalten | Programm beenden
  
```


Devices

Devices sind Steuerprogramme für Sensoren, Aktoren oder Programme für Events. Sie werden zugeordnete Events aufgerufen.

- Used Devices Devices anzeigen
- Device Infos Device-Info aufrufen
- Module Deklarierte Device-Module
- Filter Anzeige filtern

```
gardenctl
Devices
Modul      : Modulname des Devices
DevId     : Device-Id
Mode      : Gewünschter Device-Modus: -1, nicht verwenden
Status    : Tatsächlicher Device-Modus: -1, nicht bereit
Setup     : Setupstring für das Device. Setup[0] ist der Interface-Typ
C-Header  : Header der Devicesoftware.
Rem       : Remark aus der Konfigurationsdatei.

n  Modul      | DevId      | Mode | St | Setup      | C-Header      | Rem
0  DISPLAY    | PROG       | 0    | 0  | p          | devdisp.h     | Anzeige
1  LOGDATEI   | PROG       | 0    | 0  | p          | devlog.h      | Logdatei schreiben
2  GPIOCHIP   | Relays     | 12   | 12 | Co0       | relayc.h      | Relay Heizung
3  HEIZUNG    | PROG       | 0    | 0  | p          | devprog.h     | Prog Heizung
4  GPIOCHIP   | Relays     | 15   | 15 | Co0       | relayc.h      | Relay Wasser
5  TIMER      | PROG       | 1    | 0  | t test.timer | devtimer.h    | Timer Wasser
6  BMP180     | 0x77       | 1    | 1  | 2         | bmp180i.h     | 1 Temperatur
7  BMP180     | 0x77       | 2    | 2  | 2         | bmp180i.h     | 2 Luftdruck
8  DS1820     | 10-000802e45d3c | 0    | 2  | S         | ds1820s.h     | Temp Sensor 1-Wire

Used Devices | Device Infos | Module | Filter: Alle, Bereit, Gewünscht
```

Used Devices

Es werden die die konfigurierten Felder, die Laufzeitinformationen und das Event angezeigt.

```
gardenctl
Devices
Config; /home/guenther/c/pi/bin/gardenctl/bin/_gardenctl/garden.devices

Die grünen Feldewerte wurden aus der Configdatei gelesen. In Device-Open werden dunkelgrünen Felder bestimmt. Used Devices haben Status>1.

DevLst | Used
DevLst[0]
Modul : DISPLAY      | Device-Modul. "Modul,DevId,Modus" bestimmt das Device!
DevId : PROG         | Device-Id: PROG, Chip-ID oder I2c-Device-Id
Modus : 0            | Gewünschter Modus, tatsächlicher Modus ist 'Status'
Setup : "p"         | Setupstring. Setup[0] ist der Interface-Typ
Rem   : Anzeige     | Rem Modul

Header: devdisp.h    | Header/Beschreibung der Steuerfunktion
RemCtl: Terminalanzeige | Rem Steuerfunktion, DEVCtlRem
DevCtl: 0x55ac3951b4a6 | Steuerfunktion FnkDevCtl() aus Moduliste von 'devices.m'
Status: 0            | Nach Open: Tatsächlicher Modus, Close:-1
ComInt: 0            | Last DevCom Parameter: int32 t
ComStr: devdisp.h   | Last DevCom Parameter: String
Event : Display, Anzeige | Besitzer des Devices
```

Device Infos

Die Infos werden durch den Aufruf der Device-Info Funktion ermittelt.

Für open Devices können damit die aktuellen Informationen von Sensoren und Aktoren abgefragt werden.

```
gardenctl
Device[1 ]
Obj Log      : Daten in Logfile schreiben
Device      : LOGDATEI,PROG,0      | Rem: Logdatei
Header      : devlog.h
Log         : ON
LogOk       : 1 | Var(LogOn ):1 | Logdatei schreiben
LogErr      : 1 | Var(LogErr):1 | Fehler loggen
Path        : /home/guenther/c/pi/bin/gardenctl/bin/_gardenctl/20250125.log | Logdatei
fLog        : 0x55ac6b04bfb0 | Filepointer der Logdatei
LogStart    : 1737793021 2025-01-25 09:17:01 | Startzeit
LogEnd      : 1737845999 2025-01-25 23:59:59 | Endzeit
LogEV[]     : (nil) | Eventliste aus Open

Weiter mit Taste
```

Module

Es werden die deklarierten Programm-Header und die DevCtl Funktion für Devices angezeigt.

Die Header aus der Linkbibliothek werden über die Datei devicesm.c in gardenctl deklariert.

Das erste Zeichen im Setupstring eines Devices legt den Interfacetyp (IfTyp) des Devices fest.

```
gardenctl
Device Module
Module für Programm gardenctl in Sourcedatei devicesm.c

Diese Devices werden kompiliert und können verwendet werden.
Im Device-Setup ist das erste Zeichen der Interface-Typ.

Device | IfTyp | C-Header      | DevCtl-Funktion
DISPLAY | p     | devdisp.h     | ok
LOGDATEI | p    | devlog.h      | ok
BMP180   | I    | bmp180s.h     | ok
BMP180   | 2    | bmp180i.h     | ok
DS1820   | S    | ds1820s.h     | ok
SI7021   | 2    |               | ok
GPIOCHIP | C    | relayc.h      | ok
HEIZUNG  | p    | devprog.h     | ok
TIMER    | t    | devtimer.h    | ok

Option IfTyp: Es ist nur ein Interface möglich.
p Programm
t Timer
C gpiochip
W 1Wire
S 1Wire sysfs
2 i2c-1 ioctl()
I i2c sysfs
```

Events

Events beschreiben, welche Devices wann und wie in der Steuerungs-Loop() aufgerufen werden. Jedes Event ist dazu mit genau einem Device verlinkt.

Die Events werden von Loop() fortlaufend alle WHSec geprüft. Events mit mit WHSec=0 können von anderen Events/Devices aufgerufen werden.

Für termingesteuerte Events gibt es ein programmierbares **TIMER** Device.

Used Events Eventliste anzeigen
Loop: Warteschlange Wartende Events anzeigen

```
gardenctl
Event : Eindeutige Event-Id
Typ : Art des Events
Device : DeviceLink: "Modul,DevId,Modus"
DevN : Device-Index
Cmd : Event-Exec Wunschparameter, -1 not used
CmdExec : Nächster Event-Exec Parameter, -1 not used
Rem : Remark aus der Konfigurationsdatei
```

n: Event	Typ	Device	Dev	Cmd	CmdE	Rem
0: Display	d	DISPLAY, PROG, 0	0	0	0	Anzeige
1: Log	l	LOGDATEI, PROG, 0	1	0	0	Logdatei
2: T1	s	DS1820, 10-000802e45d3c, 0	8	0	0	Temp Fruehbeet
3: RH	r	GPIOCHIP, Relays, 12	2	0	0	Relay Heizung
4: Heizung	p	HEIZUNG, PROG, 0	3	0	0	Steuerung Heizung
5: TA	s	BMP180, 0x77, 1	6	0	0	Temp Aussen
6: P1	s	BMP180, 0x77, 2	7	0	0	Luftdruck
7: RW	r	GPIOCHIP, Relays, 15	4	0	0	Relay Wasser
8: TimerRW	t	TIMER, PROG, 1	5	0	0	Timer

Used Events | Loop: Warteschlange

Used Events

Liste der momentan verwendeten Events. Felder mit ':' stammen aus der Konfigurationsdatei. Felder mit '=' zeigen berechnete Laufzeitinfos.

CheckTime: Bei Time>=**CheckTime** wird die Devicefunktion Exec mit Parameter **CheckCmd** aufgerufen. Danach werden für das Event die nächsten Werte **CheckTime** und **CheckCmd** berechnet.

```
gardenctl
Event-Liste
Event[0]
Name : Display | Eindeutige Event-Id
Rem : Anzeige | Remark
Typ : d, Display | Art des Events
Device : "DISPLAY, PROG, 0" | DeviceLink: "Modul,DevId,Modus"
Open : " | Device Open-String
Cmd : 0, used | Wunschparameter für Event-Exec, -1 not used
WHSec : 2 | Event-Exec alle WHSec wiederholen
```

```
CmdExec = 0 | Parameter für Event-Exec, -1 not used
DevN = 0 | Device[0], Rem: 'Terminalanzeige'
CheckTime = 2025-01-25 10:07:21 | Nächste Check-Time für Loop oder 0
CheckCmd = 0 | Event-Exec Parameter bei Check
ErrCount = 0 | Fehlerzähler des Events
Debug = 0 | Debugausgabe für das Event
```

Loop: Warteschlange

Die Liste zeigt die wartenden Events in der Warteschlange von Loop().

Zum Zeitpunkt Time>=**CheckTime** wird die Device-Exec Funktion des Events mit dem Parameter **CheckCmd** aufgerufen.

Events mit **CheckTime**==0 werden von Loop() immer übersprungen.

Beispiel: Event '**RH**' und '**RW**'. Diese Relais werden von den Programmdevices '**Heizung**' und '**TimerRW**' aufgerufen.

```
gardenctl
Events
Event : Eindeutige Event-Id
Typ : Art des Events
Device : DeviceLink: "Modul,DevId,Modus"
DevN : Device-Index
Cmd : Event-Exec Wunschparameter, -1 not used
CmdExec : Nächster Event-Exec Parameter, -1 not used
Rem : Remark aus der Konfigurationsdatei
CheckTime: Nächster Prüfzeitpunkt
CheckCmd : Event-Exec Parameter
```

```
Loop | Warteschlange der Events
Time: 2025-01-25 10:08:42 Uhr | CheckTime und CheckCmd anzeigen.
```

Loop	Event	T CmdE	CheckTime	CheckCmd
Event[0]	Display	d 0	2025-01-25 10:08:42	0
Event[1]	Log	l 0	2025-01-25 10:08:42	0
Event[2]	T1	s 0	2025-01-25 10:08:42	0
Event[3]	RH	r 0	00:00:00	0
Event[4]	Heizung	p 0	2025-01-25 10:08:42	0
Event[5]	TA	s 0	2025-01-25 10:08:42	0
Event[6]	P1	s 0	2025-01-25 10:08:42	0
Event[7]	RW	r 0	00:00:00	0
Event[8]	TimerRW	t 0	2025-01-25 10:08:42	1

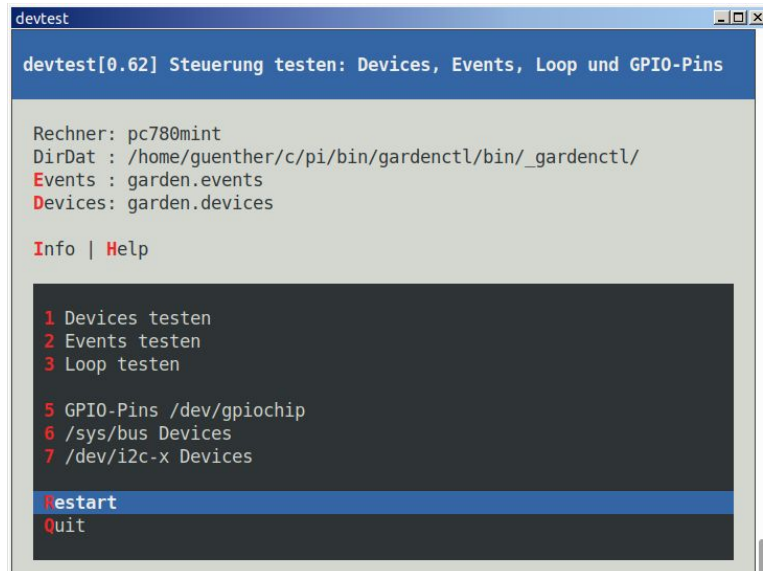
Testprogramm devtest

Mit Befehl '3 Testprogramm devtest' aus dem Hauptmenu kann in das Testprogramm `devtest` umgeschaltet werden. Programm `devtest` ersetzt das laufende `gardenctl` und wird automatisch mit den aktuellen Konfigurationsdateien für `Events` und `Devices` gestartet.

Mit `devtest` können Einstellungen für Devices, Events, Loop und GPIO-Pins von Steuerungen einzeln getestet werden. Am PC ohne angeschlossene Hardware werden die Geräte simuliert.

Dokumentation siehe: [2_devtest.pdf](#)

Mit den Befehlsoptionen `5,6,7` kann die angeschlossene Hardware getestet oder ausgelesen werden. Auf dem PC werden die Geräte simuliert.



```
devtest
devtest[0.62] Steuerung testen: Devices, Events, Loop und GPIO-Pins

Rechner: pc780mint
DirDat : /home/guenther/c/pi/bin/gardenctl/bin/_gardenctl/
Events : garden.events
Devices: garden.devices

Info | Help

1 Devices testen
2 Events testen
3 Loop testen

5 GPIO-Pins /dev/gpiochip
6 /sys/bus Devices
7 /dev/i2c-x Devices

Restart
Quit
```


Datei- und Ordnerübersicht

Linkbibliotheken

Das Programm `gardenctl` verwendet Funktionen aus der Linkbibliothek `c/pi/bininc/`. Die Header und Sourdateien der Funktionen werden dabei immer über relative symbolische Links eingebunden. Siehe Dokumentation [2_devtest.pdf](#).

Beispiel: Einen relativen symbolischen Link für die Linkbibliothek `loop.c` anlegen.

```
cd c/pi/bin/gardenctl          in den Programmordner wechseln
ls ../../bininc/events/loop.c Linkpfad testen
ln -si ../../bininc/events/loop.c symbolischen Link interaktiv anlegen
```

Dateien

Die aktuellste Dateiübersicht findet man in `1_read.me`

```
c/pi/bin/
├── gardenctl/
│   ├── bin/
│   │   ├── gardenctl/
│   │   │   ├── 1_read.me
│   │   │   ├── 20220329.log
│   │   │   ├── gardenctl.conf
│   │   │   ├── gardenctl.devices
│   │   │   ├── gardenctl.events
│   │   │   └── ...
│   │   └── gardenctl
│   └── gardenctl
├── gardenctl.h
├── gardenctl.c
├── run.c
├── devicesm.c
├── makefile
├── Linkbibliotheken aus 'c/pi/bin/bininc/...' Die Dateien dieser
│   Bibliothek werden über relative Links in die Programme eingebunden.
├── @loop.h
├── @loop.c
├── @events.h
├── @events.c
├── @eventsedit.c
├── @devices.h
├── @devices.c
├── @devdisp.h
├── @devdisp.c
├── @devlog.h
├── @devlog.c
├── @devprog.h
├── @devprog.c
├── @devtimer.h
├── @devtimer.c
├── Linkbibliotheken aus 'c/pi/bin/bininc/' zur Ansteuerung der
│   Raspberry Pi Hardware. Diese Interfaces werden von den Aktoren
│   und Sensoren verwendet.
├── @gpinci.h
├── @gpinci.c
├── @gpinci_pi.h
├── @gpiosb.h
├── @gpiosb.c
├── @gpioi2.h
├── @gpioi2.c
├── Linkbibliotheken aus 'c/pi/bin/bininc/' für Aktoren und Sensoren.
│   Diese Devices verwenden die GPIO-Interfaces zur Ansteuerung.
├── @relayc.h
├── @relayc.c
├── @ds1820s.h
├── @ds1820s.c
├── @bmp180i.h
├── @bmp180i.c
├── @bmp180s.h
├── @bmp180s.c
├── ...
└── gardenctl.geany
```

Projektverzeichnis
Ordner Programm

Ordner Konfiguration

Hilfdatei

Logdateien, Option

Konfiguration für gardenctl

Konfiguration für Devices

Konfiguration für Events

fertiges Programm

Globale Definitionen

init(), main(), Exit()

Dialog- und Hilfsfunktionen

Modulverzeichnis. Liste der verfügbaren C-Programme für Devices aus der Linkbibliothek

makefile

Zentrale Steuerungsschleife für Events

Verwaltungsfunktionen für Events

Dialogfunktionen für Events

Verwaltungsfunktionen für Devices

Device DISPLAY: Anzeige im Terminal

Device LOGDATEI: Logdatei schreiben

Device HEIZUNG: Steuerprogramm für Heizungen

Device TIMER: Verwaltet Timerprogramme für Events

GPIO-Interface für Paspberry I/O-Pins. Verwendet das Chip-Interface /dev/gpiochip0 zum Steuern. Kopie von <linux/gpio.h> vom Raspberry Pi für die Simulation am PC

GPIO-Interfaces im sysfs unter /sys/bus - i-Wire Interface mit SBUS /sys/bus/w1 steuern - i2c-Interface mit SBUS /sys/bus/i2c steuern

GPIO-Interface: i2c-Interface /dev/i2c-X mit ioctl()

Device: Relais mit Raspberry GPIO steuern

Device: 1-Wire Temperatursensore vom Typ DS1820

Device: Sensor BMP180. i2c Feuchte und Temperatursensor i2c-Interface /dev/i2c-X mit ioctl()

Device: Sensor BMP180. i2c Feuchte und Temperatursensor SBus i2c-Interface /sys/bus/i2c

Starter für IDE geany

GNU General Public License

```
/*
 * Copyright 2022-25 Günther Schardinger <v.schardinger@gmx.net>
 *
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation; either version 2 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
 * MA 02110-1301, USA.
 */
```