gardenctl | Gartensteuerung

Inhaltsverzeichnis

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:

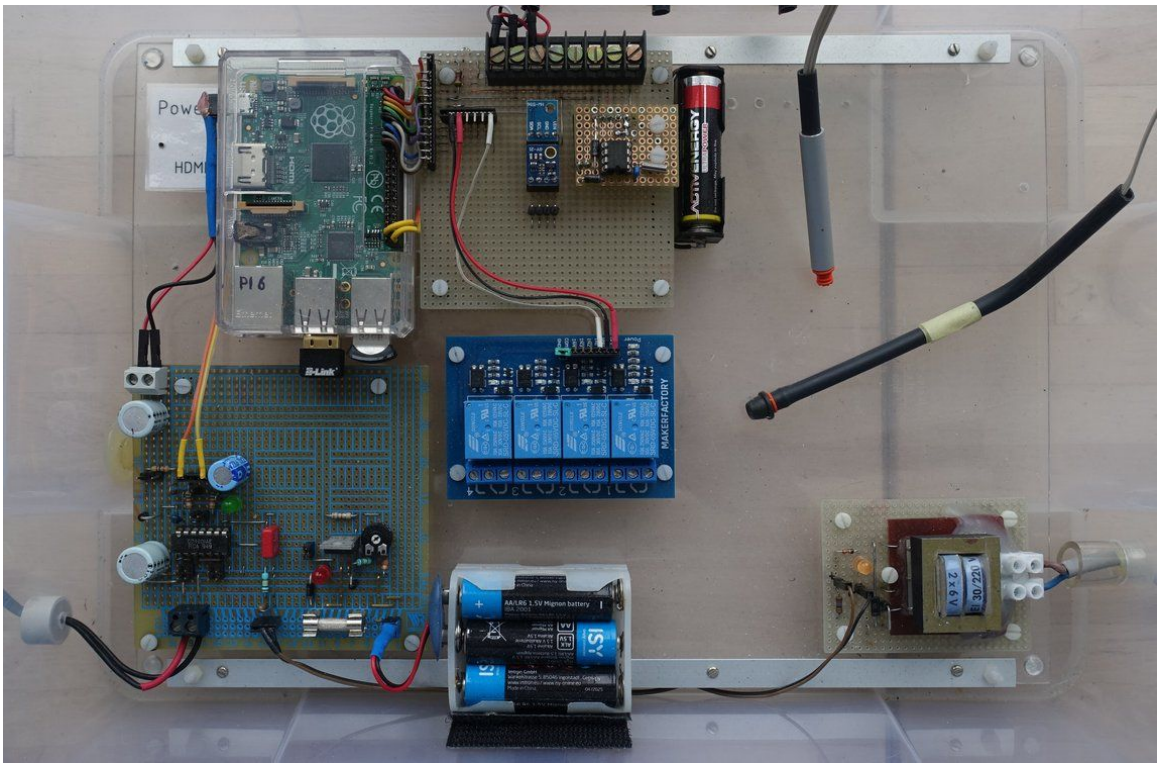
- ▷ Automatischer Betrieb mit Fernbedienung über Lan/Wlan
- ▷ Ergebnisse im Terminal anzeigen
- ▷ Ergebnisse und Ereignisse in eine Logdatei schreiben
- ▷ ...
- ▷ Beispiel: Heizung im Frühbeet steuern
- ▷ Beispiel: Zeitgesteuerte Bewässerung

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

Hardware

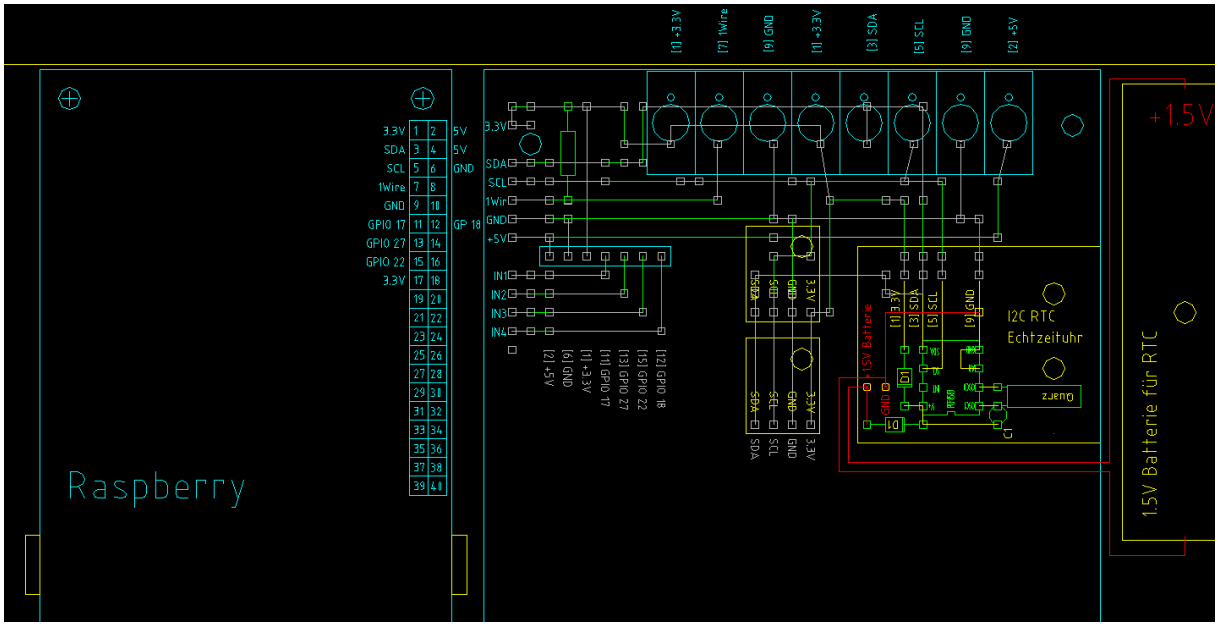
Spalte: A B C D E F G H

Zeile 1 2 3 4 5 6 7



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 wird 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 Fernzugriffe 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-Datenträgers 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 erfolgreichen 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

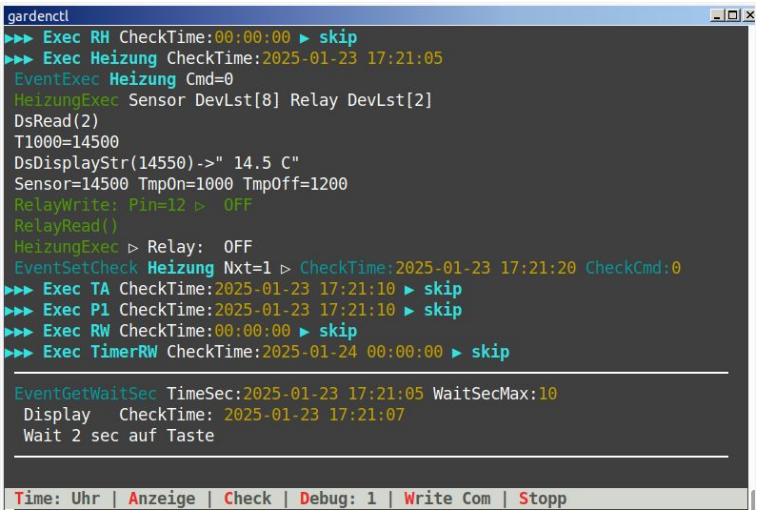
Exit Terminal	Terminal verlassen. Im automatischen Modus läuft die Steuerung ohne Terminal weiter!
Menu	Das Wartungsmenu aufrufen
Refresh	Anzeige neu aufbauen
Anzeige	Umschalten auf fortlaufende Anzeige

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 EventExex() Funktionen.

Das Bild zeigt den Verlauf von EventExex() 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().

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.

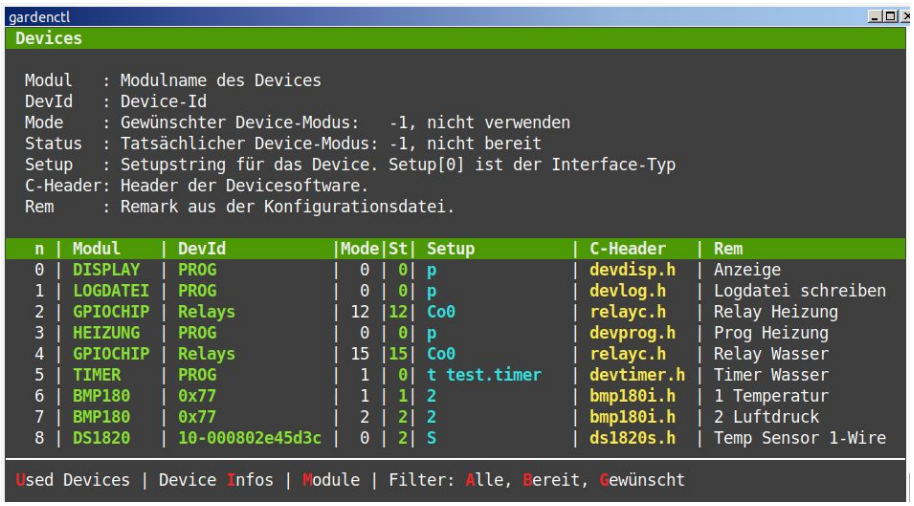
Mit Befehl 3 kann Programm devtest gestartet für umfassende Tests werden. Dabei wird Programm gardenctl vollständig durch das Testprogramm devtest ersetzt. Die Konfigurationen für Events und Devices werden dabei übernommen.

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

Devices

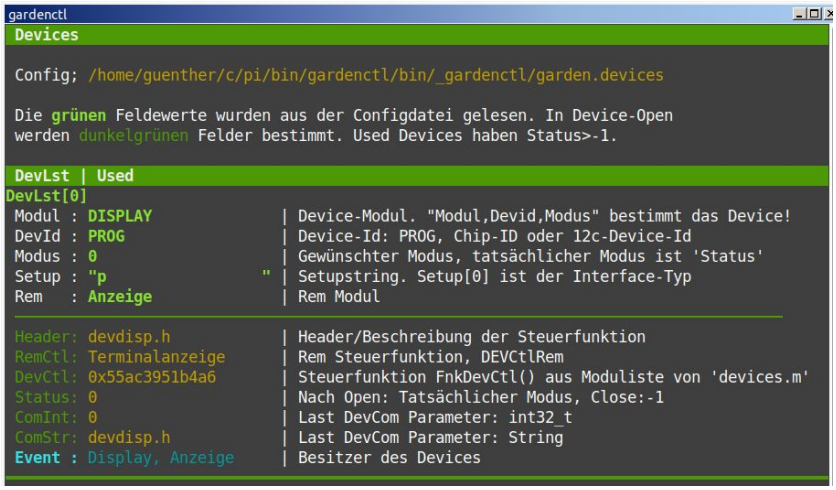
Devices sind Steuerprogramme für Sensoren, Aktoren oder Programme für Events. Sie werden zugeordnete Events aufgerufen.

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



Used Devices

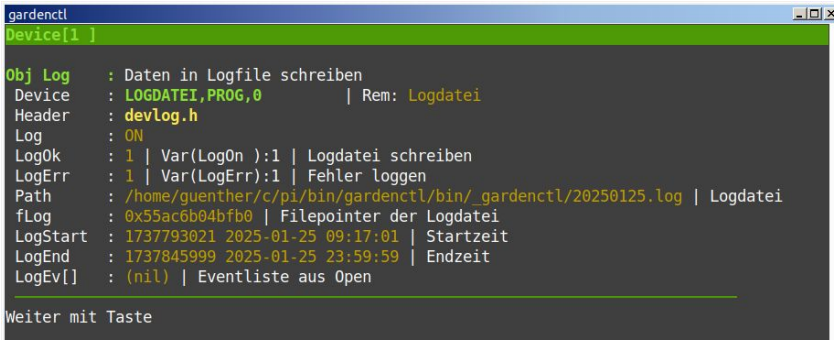
Es werden die die konfigurierten Felder, die Laufzeitinformationen und das Event angezeigt.



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.

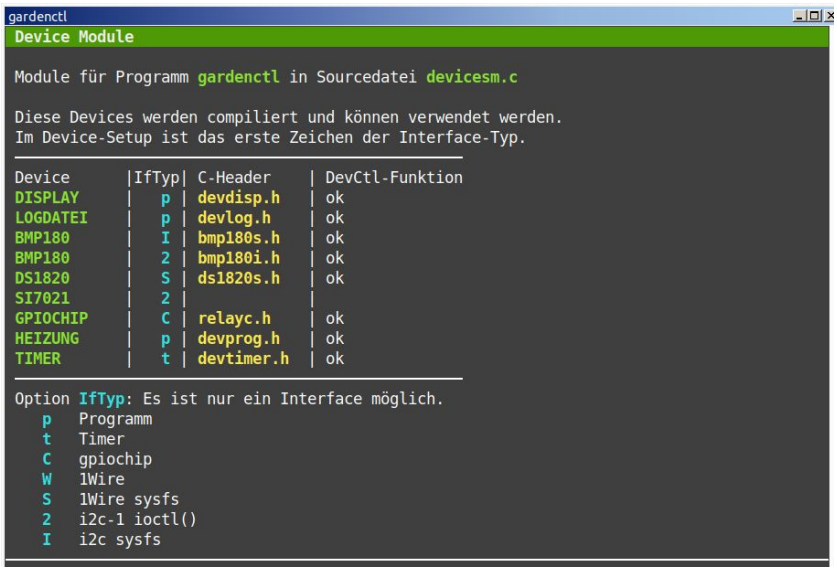


Module

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.



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.

```
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	Eventliste anzeigen
Loop: Warteschlange	Wartende Events anzeigen

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.

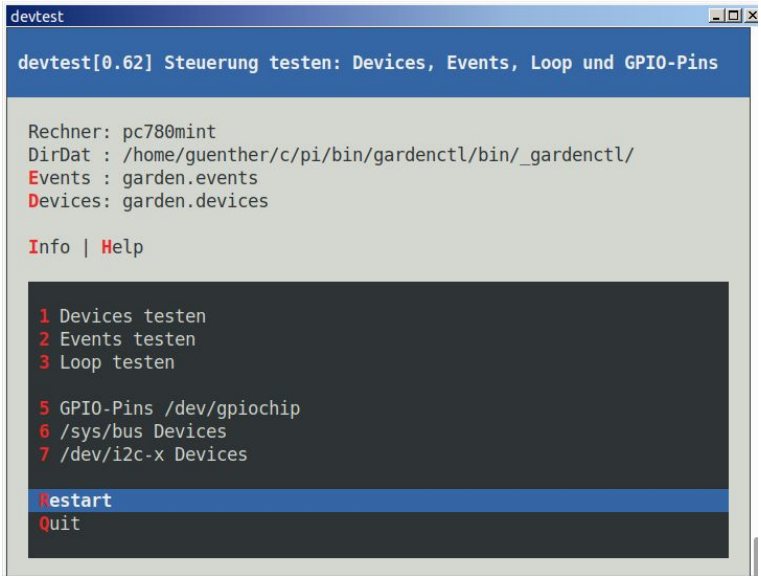
```
gardenctl
 Events

 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.

Datei- und Ordnerübersicht

Linkbibliotheken

Das Programm gardenctl verwendet Funktionen aus der Linkbibliothek c/pi/bininc/. Die Header und Sourcedateien 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/                      Projektverzeichnis
    │   ├── bin/                        Ordner Programm
    │   │   ├── _gardenctl/             Ordner Konfiguration
    │   │   │   ├── 1_read.me           Hilfedatei
    │   │   │   ├── 20220329.log        Logdateien, Option
    │   │   │   ├── gardenctl.conf      Konfiguration für gardenctl
    │   │   │   ├── gardenctl.devices   Konfiguration für Devices
    │   │   │   ├── gardenctl.events    Konfiguration für Events
    │   │   │   └── ...
    │   │   └── gardenctl               fertiges Programm
    │   ├── gardenctl.h     Globale Definitionen
    │   ├── gardenctl.c     init(), main(), Exit()
    │   ├── run.c           Dialog- uns Hilfsfunktionen
    │   ├── devicesm.c      Modulverzeichnis. Liste der verfügbaren
    │   │                   C-Programme für Devices aus der Linkbibliothek
    │   ├── makefile        makefile
    │   │   ____________________________________________________________
    │   │   Linkbibliotheken aus 'c/pi/bin/bininc/...' Die Dateien dieser
    │   │   Bibliothek werden über relative Links in die Programme eingebunden.
    │   ├── @loop.h         Zentrale Steuerungsschleife für Events
    │   ├── @loop.c
    │   ├── @events.h       Verwaltungsfunktionen für Events
    │   ├── @events.c
    │   ├── @eventsedit.c   Dialogfunktionen für Events
    │   ├── @devices.h      Verwaltungsfunktionen für Devices
    │   ├── @devices.c
    │   ├── @devdisp.h      Device DISPLAY: Anzeige im Terminal
    │   ├── @devdisp.c
    │   ├── @devlog.h       Device LOGDATEI: Logdatei schreiben
    │   ├── @devlog.c
    │   ├── @devprog.h      Device HEIZUNG: Steuerprogramm für Heizungen
    │   ├── @devprog.c
    │   ├── @devtimer.h     Device TIMER: Verwaltet Timerprogamme für Events
    │   ├── @devtimer.c
    │   │   ____________________________________________________________
    │   │   Linkbibliotheken aus 'c/pi/bin/bininc/' zur Ansteuerung der
    │   │   Raspberry Pi Hardware. Diese Interfaces werden von den Aktoren
    │   │   und Sensoren verwendet.
    │   ├── @gpioci.h       GPIO-Interface für Paspberry I/O-Pins. Verwendet
    │   ├── @gpioci.c       das Chip-Interface /dev/gpiochip0 zum Steuern.
    │   ├── @gpioci_pi.h    Kopie von <linux/gpio.h> vom Raspberry Pi
    │   │                   für die Simulation am PC
    │   ├── @gpiosb.h       GPIO-Interfaces im sysfs unter /sys/bus
    │   ├── @gpiosb.c       - i-Wire Interface mit SBUS /sys/bus/w1 steuern
    │   │                   - i2c-Interface mit SBUS /sys/bus/i2c steuern
    │   ├── @gpioi2.h       GPIO-Interface: i2c-Interface /dev/i2c-X mit ioctl()
    │   ├── @gpioi2.c
    │   │    ___________________________________________________________
    │   │    Linkbibliotheken aus 'c/pi/bin/bininc/' für Aktoren und Sensoren.
    │   │    Diese Devices verwenden die GPIO-Interfaces zur Ansteuerung.
    │   ├── @relayc.h       Device: Relais mit Rasperry GPIO steuern
    │   ├── @relayc.c
    │   ├── @ds1820s.h      Device: 1-Wire Temperatursensore vom Typ DS1820
    │   ├── @ds1820s.c
    │   ├── @bmp180i.h      Device: Sensor BMP180. i2c Feuchte und Temperatursensor
    │   ├── @bmp180i.c              i2c-Interface /dev/i2c-X mit ioctl()
    │   ├── @bmp180s.h      Device: Sensor BMP180. i2c Feuchte und Temperatursensor
    │   ├── @bmp180s.c              SBus i2c-Interface /sys/bus/i2c
    │   └── ...
    ├── gardenctl.geany         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.
*/
```