Hallo Ihr da draußen, hier gibt's eine weitere Programmiersprache für den Raspberry Pi. Ich hoffe, dass Ihr die nächsten Ausgaben dieses Tutorials genießen werdet, die Euch etwas darüber vermitteln, wie man damit Programme entwickelt. Obwohl wir reichlich seltsame und komplexe Dinge machen werden (dazu gehört das Compilieren des Compiler-Quellcodes) ist es eine Schritt- für-Schritt-Anleitung für absolute Anfänger, eine neue Programmiersprache zu erlernen.

1 Die Programmiersprache Icon

1.1 Einführung – Was ist Icon?

Icon ist eine benutzerfreundliche prozedurale und gut-lesbare Allzweck-Progammiersprache, die Merkmale von Sprachen wie Pascal und C vereint und in der Lage ist, in C geschriebene Bibliotheken oder Objektdateien zu nutzen. Du kannst kleine Programme in kurzer Zeit erzeugen und wirklich große Anwendungen ebenfalls. Üblicherweise wirst Du keine Zeit vergeuden, irgendwelche Strukturen oder Objekte zu füllen oder tief verschachtelte Methoden aufzurufen. Es gibt keine Zeiger – denk' an die Probleme zurück, z.B. in C nahm jeder Zeiger auf NIL Dein Programm gleich mit ins Nirgendwo, wieder und immer wieder – bis Du den Fehler entdeckt hast!

Es gibt Merkmale, die in anderen Programmiersprachen nicht implementiert sind. Erfahrungsgemäß resultiert die Verwendung all dieser Merkmale in Quellcode, der um etwa ein Drittel kürzer ist als Code in Pascal oder C, der die gleiche Anwendung realisiert. Allerdings ist Icon momentan (noch) eine der weniger bekannten Programmiersprachen, die zur Entwicklung von Software-Lösungen berücksichtigt werden sollten.

Icon wurde von Ralph E. Griswold an der University of Arizona in den letzten Jahrzehnten entwickelt und wurde zuletzt im Juni 2013 aktualisiert – davor zuletzt im Jahre 2010. Icon wird eher selten aktualisiert – so kannst Du das einmal Gelernte über eine wesentlich längeren Zeit anwenden als in anderen Sprachen.

Icon war eine der ersten kostenlosen Programmiersprachen mit offenem Quellcode, die erlaubten, dass der gleiche Programmcode ohne Änderungen auf unterschiedlichen Betriebssystemen compiliert werden konnte. Schau Dir mal die Icon-Webseite an: Icon läuft auf 14 Betriebssystemen!

Ich mag Icon!

1.2 Herunterladen & Entpacken

Die folgenden Schritte habe ich auf einem Raspberry Pi Modell B mit einer frischen, aktualisierten Version von Raspbian Wheezy auf einer 8 GB SDCard durchgeführt. Die gleiche Vorgehensweise sollte erfolgreich auf anderen Betriebssystemen zum Ziel führen – vorausgesetzt dass dort ein C-Compiler installiert ist.

Vorsicht: Verwende nicht die Version v9.4.3, die von Synaptic für die armhf-Architektur angeboten wird – es ist nicht die aktuelle Version und resultiert in einen Compiler, der wie ein Skript aufgerufen werden muss (mit ./) - die compilierten Programme genauso.

Alle Informationen (Bücher, Zeitschriften, Technische Berichte, Handbücher für viele verschiedene Betriebssysteme) sind auf der Icon Webseite www.cs.arizona.edu/icon (Abb. 1-1) erhältlich. Tabelle 1-1 enthält eine Vorgehensweise zum Herunterladen und Entpacken des Quellcodes des Icon-Compilers.

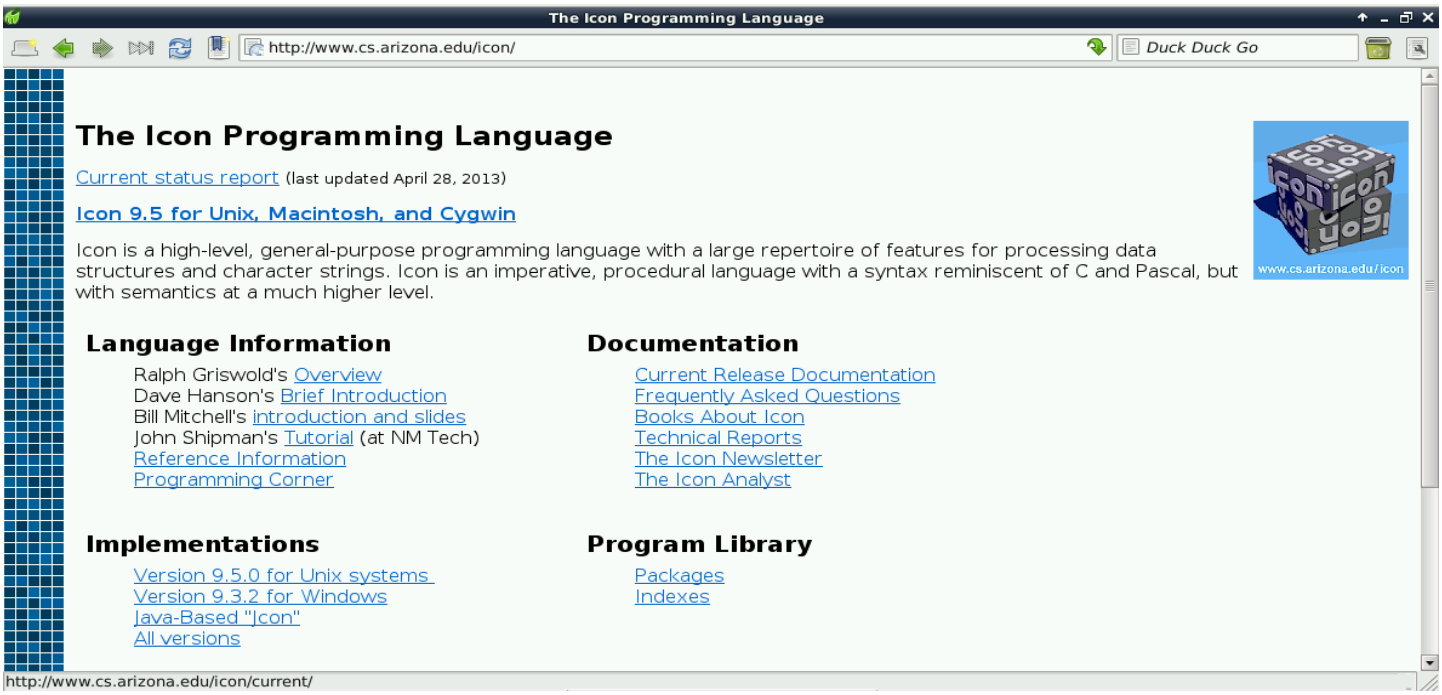

Fig. 1-1: Die Icon-Webseite

Tab. 1-1: Herunterladen und Entpacken der komprimiertenn Datei

Ins Terminal eingeben	Beschreibung
`wget www.cs.arizona.edu/icon/ftp/packages/unix/icon-v951src.tgz`	Den Quellcode herunterladen
`cd /home/pi/Downloads`	In Dein Download-Verzeichnis wechseln
`tar -x -v -f icon-v951src.tgz`	Die tgz-Datei entpacken
`mkdir /home/pi/icon9_51`	Ein Arbeitsverzeichnis für Icon erzeugen
`mv -v icon-v951src/* /home/pi/icon9_51`	Alle entpackten Verzeichnisse & Dateien in das Arbeitsverzeichnis kopieren
`sudo apt-get install build-essential libx11-dev libxt-dev libxaw7-dev`	Installiert für die Compiliierung des Quellcodes benötigte Dateien – Eine Zeile
`cd /home/pi/icon9_51`	Zum Arbeitsverzeichnis wechseln
`dir`	Den Inhalt des Verzeichnisses anzeigen
`bin config doc ipl lib` `Makefile man README src tests`	Ausgabe des Kommandos `dir` (nichts eingeben)

`dir bin`
`cd src`
`dir`
`common h icont Makefile`
`preprocrtt runtime wincap xpm`
	Zeigt den Inhalt des Verzeichnisses `bin` an – leider ohne Inhalt. In `src/icont` ist der C-Quellcode des Icon-Compilers enthalten, wir dürfen also den Icon-Compiler aus dem C-Quellcode aufbauen!

1.3 Konventionen

```
Ins Terminal eingeben
```

```
Quellcode in einen Texteditor eingeben
```

Listing: Pfad/Dateiname.Erweiterung

`In eine Zeile eingeben`

Syntax-Erklärung

`Anmerkungen und Übersichten`

`Schlüsselwörter, Funktionsnamen` um sie von Text zu unterscheiden

`Vorzunehmende Änderungen in Dateien`

1.4 Aufbauen der Version 9.51 vom Quellcode

Da sich noch niemand die Mühe gemacht, den Icon-Compiler auf den RaspberryPi zu übertragen, zeige ich euch, wie das geht.

Als Erstes bearbeiten wir einige Konfigurationsdateien. Dazu wechseln wir in das Verzeichnis `config` und fertigen dort eine Kopie einer vorhandenen Konfiguration an:

```
cd /home/pi/icon9_51/config
cp -v -r posix RaspberryPi
cd RaspberryPi
dir
define.h     Makedefs     status
```

Es gibt drei Dateien, die wir bearbeiten wollen. Ich bin ein Fan von Geany, so hoffe ich, dass Du es auch installiert hast – oder Du machst es gleich jetzt. Natürlich kannst Du aber jeden Texteditor Deiner Wahl nutzen.

Diese drei Dateien sollten geöffnet und nach der Vorgabe von Tabelle 1-2 geändert und gespeichert werden.

```
cd ../..
make Status name=RaspberryPi
```

Du wirst die gleiche Information erhalten, die Du in der Datei `status` festgelegt hast. Unsere Vorgehensweise, Icon auf dem Raspberry Pi einzurichten, scheint zu funktionieren!

Tab. 1-2: Erforderliche Änderungen an den drei Dateien im Verzeichnis `/home/pi/icon9_51/config`

define.h	Makedefs (no changes)	status
/* * Icon configuration file for RaspberryPi system with X window */ #define UNIX 1 #define RaspberryPi 1 #define LoadFunc	# CC C compiler # CFLAGS flags for building C files # CFDYN additional flags for dynamic functions # RLINK flags for linking run-time system # RLIBS libraries to link with run-time system # TLIBS libraries to link for POSIX threads # XLIBS libraries to link for graphics # XPMDEFS definitions for building XPM library # GDIR directory of graphics helper library CC = gcc CFLAGS = -O CFDYN = -fPIC RLINK = -Wl,-E RLIBS = -lm -ldl TLIBS = -lpthread XLIBS = -lX11 XPMDEFS = -DZPIPE GDIR = xpm	System configuration: RPi (Raspbian Wheezy) Latest Icon version: Version 9.5.1 Installer: Andreas Schulz Icon Project Missing features: Mutual colors Known bugs: None Comments: Tested on ARM V6 systems running Raspbian Wheezy on Raspberry Pi (RPi) Other operating systems are not tested yet. Raspbian operating systems may require package installation in order to build: $ sudo apt-get install build-essential libx11-dev libxt-dev libxaw7-dev Date: January 3, 2013

Gib die Zeilen ein

```
make X-Configure name=RaspberryPi
make X-Configure=RaspberryPi
make status=RaspberryPi
make Icont
make Samples
make Test
make Benchmark
```

Der Raspberry Pi compiliert die Quelldateien mittels `gcc` (GNU C Compiler), und erzeugt dabei den Compiler `icont`, den Interpreter `iconx` und einige Bibliotheken. Sei geduldig – es wird einige Zeit dauern... der Bildschirm füllt sich, etwaige Fehlermeldungen bitte ignorieren.

1.4 Erster Test: Der Compiler selbst

Starte einen Texteditor und gib Dein erstes Icon-Programm ein – natürlich ohne die Zeilennummern.

Listing: home/pi/icon9_51/bin/hallo.icn

```
1 procedure main()
2    write("Hallo, Ihr da draußen!")
3 end
```

#	Erläuterung
1	Da Icon eine prozedurale Programmiersprache ist, wirst Du nicht erstaunt sein, dass es ein reserviertes Schlüsselwort `procedure` gibt. Jedes Icon-Programm muss eine Prozedur `main` haben. Es bildet den Eintrittspunkt, um das Programm zu starten. Du kannst innerhalb der Klammern `()` Parameter übergeben – aber das werden wir ein anderes Mal machen.
2	`write()` ist eine eingebaute Function – na ja, Du hast Recht, natürlich ist es eine eingebaute Pozedur – die Du in Deinen eigenen Programmen verwenden kannst. `write()` erhält einen Parameter, in diesem Beispiel sollte die Zeichenkette "Hallo, Ihr da draußen!" auf dem Bildschirm ausgegeben werden.
3	`end` wird das Programm beenden – was auch sonst?

Anmerkung: Du benötigst kein einziges Semikolon ";" hinter jeder oder besonderen Zeilen wie in C oder Pascal. Du kannst sie verwenden, wenn Du es möchtest – aber da es nicht erforderlich ist, warum solltest Du das tun?

Speichere die Datei unter dem Namen „hallo.icn" in das Verzeichnis `bin` Deines Icon-Arbeitsverzeichnisses, öffne wieder einmal das Terminal und betrete dieses Verzeichnis durch

```
cd /home/pi/icon9_51/bin
```

Du findest dort den Compiler `icont`, den Interpreter `iconx` und Deine erste Icon-Quelldatei `hallo.icn`.

Um diese zu compilierem gib ein

```
icont hallo
iconx hallo
```

`icont` erzeugt das ausführbare Programm `hallo` und `iconx` startet `hello`. Du kannst Dein Programm auch direkt nach dem Compilierenn starten, wenn Du eingibst `icont hello -x`

Du kannst es auch einfach durch Eingabe von `hello` starten.

Das Ergebnis ist in jedem Fall das Gleiche: Die Zeichenkette "Hallo, Ihr da draußen" wird auf dem Terminal angezeigt. Dein Compiler-Paket funktioniert!

1.5 Zweiter Test: Die graphische Umgebung

Gib ein weiteres Programm ein:

Listing: home/pi/icon9_51/bin/hello_window.icn

```
1 link graphics
2
3 procedure main()
4    WOpen("label=Mein 1. Icon-Fenster")
5    WWrite("Hallo, Ihr da draußen")
6    WDelay(5000)
7    WClose()
9 end
```

Auf eine Erläuterung der einzelnen Programmzeilen verzichte ich dieses Mal noch. Es geht mir hier nur darum, die Funktionsfähigkeit des Icon-Compilers zur Erstellung von Programmen mit einer graphischen Benutzeroberfläche zu zeigen.

Speichere es ebenfalls in das Verzeichns `bin`,

wobei Du den Namen `hello_window.icn` verwendest. Compiliere es und lasse es auf die bewährte Weise laufen. Du erhältst ein Fenster mit dem festgelegten Titel, das die Zeichenkette “Hallo, Ihr da draußen“ anzeigt, 5 Sekunden wartet und sich danach schließt. Ja? Glückwunsch! Du hast eine korrekt funktionierende Icon-Compiler-Umgebung!

1.6 Umgebungsvariablen

Es gibt einige sog. Umgebungsvariablen, die definiert werden müssen, damit Quelldateien compiliert werden können, die sich nicht im Icon `bin` Verzeichnis befinden, z.B. falls Du an einem Projekt mit Dutzenden an Quelldateien arbeitest – wenn Du dies möchtest.

Nun ja, LPATH muss alle Verzeichnisse innerhalb IPL (Icon Programming Library) abdecken, die einschließbare *.icn-Dateien enthält. IPATH muss alle Verzeichnisse innerhalb IPL abdecken, die vorcompilierte ucode-Dateien (*.u1, *.u2) enthalten. Tabelle 1-3 enthält eine Übersicht über die IPL-Verzeichnisse und eine Zuordnung, welches Verzeichnis von LPATH oder IPATH abgedeckt werden sollte.

Öffne die Datei `/home/pi/.bashrc` und füge ein paar Zeilen an das Ende dieser Datei an. Die unterstrichene Definition für LPATH muss in einer einzigen Zeile eingegeben werden. Speichere diese Datei und führe einen Reboot durch. Ab jetzt kannst Du beliebige Dateien in `ipl/progs` oder `ipl/gprogs` compilieren.

Box 1-1: Umgebungsvariablen für den Icon-Compiler icont,Voreinstellungen in ()

BLKSIZE (500000)	Die Anfangsgröße des allozierten Speicherbereiches "Block", in Bytes. Ich bin mir sicher, dass wir diesen Wert nicht zu ändern brauchen – nur dann, wenn Probleme beim Compilieren sehr großer Programme auftreten.
COEXPSIZE (2000)	Die Größe jeder Co-expression (in Bytes). Wir werden den voreingestellten Wert nur dann ändern, wenn wir in Schwierigkeiten geraten, Quelldateien zu compilieren, die Co-expressions enthalten, womit eine Art von Parallel-Prozessen ermöglicht werden.
IPATH (nicht definiert)	Der Ort der ucode-Dateien, der in Link-Deklarationen für den Icon-Compiler festgelegt wird. IPATH ist eine durch Leerzeichen getrennte Auflistung von Verzeichnissen. Das aktuelle Verzeichnis wird immer als erstes durchsucht, trotz des Wertes von IPATH.
LPATH (nicht definiert)	Der Ort der Quelldateien, die in der Präprozessor-Anweisung `$include` angegeben werden. Ansonsten ist LPATH ähnlich zu IPATH.
MSTKSIZE (10000)	Die Größe des Haupt-Interpreter-Stapelspeichers. Ich bin mir sicher, dass wird diesen Wert nicht zu ändern brauchen – nur falls wir Probleme bekommen, sehr große Programme zu compilieren.
NOERRBUF (nicht definiert)	Voreinstellungsgemäß wird die Standardfehlerausgabe in `&errout` gepuffert. Falls diese Variable gesetzt ist, wird die Standardfehlerausgabe `&errout` nicht gepuffert.
STRSIZE (500000)	Die Anfangsgröße des Speicherbereiches "String", in Bytes.
TRACE (nicht definiert)	Der Anfangswert von `&trace`. Falls diese Variable einen Wert hat, überschreibt es die Option -t von `icont`.

ACHTUNG: Verwechsle LPATH nicht mit „Pfad zu Libraries“ und IPATH nicht mit „Pfad zu Include-Files“. Es ist genau umgekehrt, vielleicht ein Fehler der allerersten Installation, der niemals geändert wurde?

Script: /home/pi/.bashrc Am Ende hinzufügen

```
echo "PATH to Icon erweitern"
PATH=$PATH:/home/pi/icon9_51/bin
echo $PATH
export PATH

echo "IPATH setzen"
IPATH="/home/pi/icon9_51/ipl/cfuncs"
echo $IPATH
export IPATH

echo "LPATH setzen"
LPATH="/home/pi/icon9_51/ipl/cfuncs /home/pi/icon9_51/ipl/gincl
/home/pi/icon9_51/ipl/gprocs /home/pi/icon9_51/ipl/incl /home/pi/icon9_51/ipl/procs"
echo $LPATH
export LPATH
```

Tab. 1-3: Übersicht über die Verzeichnisse in IPL

Verzeichnis	Inhalt	Abzudecken durcho	
		LPATH	IPATH
cfuncs	Umgebung, um Funktionen, die in C geschrieben wurden aufzurufen	X	X
data	Datenverzeichnis für andere Programme in der IPL	-	-
docs	Dokumentation für einige Programme in der IPL	-	-
gdata	Datenverzeichnis für andere Programme mit einer GUI (graphische Benutzeroberfläche)	-	-
gdocs	Dokumentation für Graphikbeschreibungen		-
gincl	Include-Dateien für Anwendungen mit einer GUI	X	-
gpacks	Anwendungspakete (wie Tetris, GED und VIB) mit einer GUI	-	-
gprocs	Include-Dateien, die Unterstützungsfunktionen für Anwendungen mit einer GUI bereitstellen	X	
gprogs	Kleine Anwendungen mit einer GUI (gut geeignet, um daraus zu lernen)	-	-
incl	Include-Datei für Anwendungen	X	
packs	Einige Anwendungen mit einer GUI	-	-
procs	Include-Dateien, die Unterstützungsfunktionen für Anwendungen bereitstellen	X	-
progs	Kleine Anwendungen ohne GUI	-	-

Das war's für heute! Genießt Euer Leben bis zum nächsten Mal!

Was kommt das nächste Mal?

2 Konfiguration von Geany

- Wie erzeugt man in Geany Menüs, die die Auswahl der Programmiersprache Icon erlauben?
- Wie erhält man ein Syntax-Highlighting für Icon-Schlüsselworte?
- Wie compiliert, linkt und führt man Programme direkt aus Geany heraus?
- Wie richtet man eine funktionsfähige Icon-Vorlage ein?
- Wie erhält man Unterstützung durch Code-Schnipsel?
- Wie passt man das Programm VIB (visual interface builder) an zeitgemäße Bildschirmauflösungen an?