Tcl-Erweiterungen selbst entwickeln

Weltenwanderer

Tcl-Erweiterungen in C oder C++ sind leichter zu entwickeln, als viele Programmierer denken. Mit wenigen Zeilen C-Code sind zeitkritische Funktionen oder Hardware-nahe Routinen implementiert. Tcl-Skripte nutzen diese Ergänzungen ebenso bequem wie vorhandene Routinen oder eigene Prozeduren. Carsten Zerbst

Programmierer trennen ihre Sprachen grob in zwei Welten auf: Skriptsprachen und Compilersprachen. Diese Teilung setzt sich bei den Entwicklern selbst fort, die meisten fühlen sich nur in einer der beiden Sprachwelten heimisch und scheuen die andere.

Dank der enormen Menge an Tcl-Erweiterungen ist die Skriptwelt ziemlich groß, fast alle Aufgaben lassen sich mit reinem Tcl oder einer fertigen Erweiterung erledigen. Hin und wieder gibt es aber ein Problem, das sich nicht so leicht lösen lässt. Doch jeder Tcl-Entwickler darf den Sprung in die kompilierende Parallelwelt wagen. Schon ein paar Funktionen aus der C-API von Tcl genügen, um eine Erweiterung (in Form einer Shared Library) zu entwickeln.

Zudem sind viele Probleme, mit denen sich die C-Programmierer herumschlagen müssen, in der Tcl-API schon gelöst – sogar plattformübergreifend. Das betrifft den kompletten Bereich der Stringbearbeitung, die Speicherverwaltung sowie das Dateisystem.

Die Dokumentation der API liegt als Sammlung von Manualseiten vor: »man Tcl_SetVar« zeigt beispielsweise die Funktion, mit der C-Code eine Tcl-Variable setzt. Ein kompletter Überblick über die API findet sich auf [1]. Für den Einstieg genügen aber wenige, gängige Routinen.

Objekte statt Strings

Vor der Tcl-Version 8.0 galt der Grundsatz „Alles ist ein String“, der Interpreter verarbeitete ausschließlich Zeichenketten. Der Performance war dies allerdings sehr abträglich, beim Übergang von Tcl zu C und zurück zu Tcl wurde eine Zahl jedes Mal zwischen String und Integer konvertiert. Um diese ständigen Umwandlungen zu vermeiden, führte Tcl 8.0 das »Tcl_Obj« als zentralen Datentyp ein. Nach außen hin verhält sich das Objekt wie ein String, es enthält jedoch auch einen Long, Double oder Pointer auf andere Datenstrukturen.

Bei Berechnungen mit »Tcl_Obj« reicht die einmalige Umwandlung von String in den jeweiligen Typ, danach kann der C-Code ohne Umschweife auf den gewünschten Typ zugreifen. Das Objekt speichert auch, welche der Darstellungen auf dem aktuellen Stand ist, und erzeugt beispielsweise die String-Form erst wieder, wenn das Programm sie auch benötigt. Die Tcl-Entwickler haben die alten String-basierten Funktionen aber nicht abgeschafft, sondern jeweils eine neue Version mit »Obj« im Namen hinzugefügt. Aus Performancegründen sind die neuen Funktionen zu bevorzugen.

Grundstruktur einer Erweiterung

Der Aufbau einer Tcl-Erweiterung ist reicht einfach. Sie muss sich initialisieren, die Tcl-Kommandos definieren und Code enthalten, der die neuen Kommandos in die Tat umsetzt. Ein einfaches Hallo-Welt-Beispiel findet sich in Listing 1. Es stellt das neue Tcl-Kommando »hallo« zur Verfügung, das den String »Hallo Welt!« zurückgibt.

Einstiegspunkt der Erweiterung ist die »Hallo_Init()«-Funktion, hier hinein gehört der gesamte Code zur Initialisierung von Datenstrukturen und Kommandos. Die Init-Funktion wird beim Laden der Erweiterung automatisch aufgerufen. Ihr Name setzt sich aus dem Namen der Bibliothek (mit großem Anfangsbuchstaben) plus »_Init«-Anhängsel zusammen.

Tabelle 1: Wichtige Tcl-C-Funktion

Kommando	Erklärung
Tcl_InitStubs (*Interp, Version, spätere-Version-möglich*)	Initialisiert die Stubs-Bibliothek; die Funktion gibt vor, welche Tcl-Version sie erwartet
Tcl_CreateObjCommand (*Interp, Name, Funktion, Client-Daten-Zeiger, Löschfunktion*)	Erzeugt ein Tcl-Kommando: Die Client-Daten werden bei jedem Aufruf an die C-Funktion durchgereicht; die optionale Löschfunktion räumt auf, wenn das Kommando aus dem Interpreter entfernt wird
Tcl_SetObjResult (*Interp, Objekt-Zeiger*)	Setzt den Rückgabewert des Kommandos, den das Tcl-Skript erhält
Tcl_WrongNumArgs (*Interp, Objekt-Anzahl, Objekt-Wert-Array, Meldung*)	Behandelt Fehler: Wenn das neue Tcl-Kommando mit falschen Optionen benutzt wurde, übergibt »Tcl_WrongNumArgs« die benutzten Parameter als Objekt sowie eine ergänzende Meldung
Tcl_Obj* Tcl_NewStringObj (*Char-Array, Länge*)	Erzeugt ein String-Objekt
char* Tcl_GetStringFromObj (*Objekt-Zeiger, Länge-Zeiger*)	Liest die String-Darstellung aus einem Tcl-Objekt
Tcl_UniChar* Tcl_GetUnicodeFromObj (*Objekt-Zeiger, Länge-Zeiger*)	Liest die String-Darstellung in Unicode kodiert aus dem Tcl-Objekt
Tcl_Obj* Tcl_NewDoubleObj (*Double-Wert*)	Erzeugt ein Double-Objekt
int Tcl_GetDoubleFromObj (*Interp, Objekt-Zeiger, Double-Zeiger*)	Liest einen Double-Wert aus einem Tcl-Objekt

Für das Beispiel muss die Bibliothek also »libhallo.so« heißen.

Früher funktionierten Erweiterungen nur zusammen mit jener Tcl-Version, für die sie kompiliert wurden. Obwohl Erweiterungen als Shared Library ausgeführt sind, benötigen sie auch Funktionen aus der Tcl-API; diese Bibliothek hat der Tcl-Interpreter aber bereits gelinkt. Damit entsteht eine gegenseitige Abhängigkeit, die sich nur auflösen lässt, wenn Interpreter und Erweiterung identische API-Versionen verwenden.

Saubere Trennung

Mit Version 8.1 brachte Tcl eine wichtige Neuerung, die Stubs-Library. Sie sorgt für eine klare Trennung: Stubs ersetzen das direkte Linken von Erweiterung und Tcl-Bibliothek durch eine Sprungtabelle (Array mit Funktionszeigern).

Hält sich eine Erweiterung daran, läuft sie problemlos mit jeder neueren Tcl-Version. Der genaue technische Hintergrund ist in der Manualseite zu »Tcl_InitStubs()« beschrieben. In der Praxis genügt es, einfach den Bereich zwischen den Zeilen 18 und 22 zu kopieren, beim Compiler-Aufruf das Symbol »USE_TCL_STUBS« zu setzen und die Stub-Bibliothek statt der Tcl-Library zu linken. Die Tcl-Header sind so programmiert, dass der Entwickler sich nicht um weitere Details kümmern muss.

Die Funktion

In Zeile 25 (Listing 1) vereinbart »Tcl_CreateObjCommand()« die neue »hallo«-Funktion. In diesem einfachen Fall genügt es schon, dem Create-Object-Kommando drei Parameter zu übergeben: den Zeiger zum Tcl-Interpreter, den Namen des neuen Tcl-Kommandos sowie den Zeiger zur C-Funktion, die das Tcl-Kommando implementiert. Sie soll den String »Hallo Welt!« zurückgeben. Mit »Tcl_NewStringObj()« erzeugt sie in der Zeile 40 ein »Tcl_Obj« für den Rückgabestring. Die New-String-Funktion nimmt einen Parameter für den String sowie einen für seine Länge entgegen. Statt die Länge explizit anzugeben, soll sich der Interpreter darum kümmern, deshalb setzt das Beispielprogramm den Wert »-1« ein.

Die C-Funktion, die das Tcl-Kommando implementiert, muss dem Interpreter einen Exit-Status als Integer-Wert zurück-

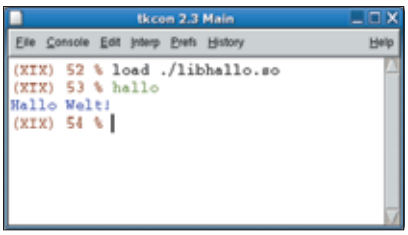


Abbildung 1: Die selbst entwickelte Tcl-Erweiterung »hallo« ist sehr einfach einsetzbar: Bibliothek laden und das neue Kommando ausführen. Der Code dazu ist in Listing 1 zu sehen.

Listing 1: Einfache Tcl-Erweiterung

```
01 /* Hello World als Tcl-Erweiterung */
02
03 #include <tcl.h>
04
05 /* Vorwärtsdeklaration des Kommandos */
06 int Hallo_Kommando (ClientData cdata,
07   Tcl_Interp *interp, int objc,
08   Tcl_Obj * CONST objv[]);
09
10 /* Erweiterung initialisieren; diese
11  * Funktion wird beim Laden vom Interpreter
12  * aufgerufen.
13  * @param interp, Pointer auf den Interpreter
14  * @return Status, TCL_OK oder TCL_ERROR
15  */
16 int Hallo_Init (Tcl_Interp *interp)
17 {
18 #ifdef USE_TCL_STUBS
19   if (Tcl_InitStubs(interp, "8.1", 0) == 0L) {
20     return TCL_ERROR;
21   }
22 #endif
23
24   /* Das hallo-Kommando erzeugen */
25   Tcl_CreateObjCommand (interp, "hallo",
26     Hallo_Kommando, NULL, NULL);
27   return TCL_OK;
28 }
29
30 /* Das hallo-Kommando ausführen
31  * @param interp, der Interpreter
32  * @param objc, Anzahl der Eingabeobjekte
33  * @param objv[], Array mit Eingabeobjekten
34  * @return Status, TCL_OK oder TCL_ERROR
35  */
36 int Hallo_Kommando (ClientData cdata,
37   Tcl_Interp *interp, int objc,
38   Tcl_Obj * CONST objv[])
39 {
40   Tcl_Obj* retval = Tcl_NewStringObj(
41     "Hallo Welt!", -1);
42   Tcl_SetObjResult (interp, retval);
43   return TCL_OK;
44 }
```

geben. Dieser Status entscheidet über Erfolg oder Fehler, er ist nicht zu verwechseln mit dem Rückgabewert (oder der Fehlermeldung) der Funktion, den das Tcl-Skript sieht. Diesen Rückgabewert setzt die C-Funktion in Zeile 42 per »Tcl_SetObjResult()«.

Damit ist die Erweiterung fertig. Neben dem GCC sind zum Übersetzen noch das komplette Tcl-Entwicklungspaket oder mindestens die Headerdateien aus den Tcl-Quellen nötig. Folgender Aufruf erledigt das Kompilieren:

```
gcc -shared -DUSE_TCL_STUBS listing1.c ➥
  -o libhallo.so -ltclstub8.4
```

Die entstandene Bibliothek »libhallo.so« kann man nun in den Interpreter laden, am einfachsten mit dem Tcl-Kommando »load *Bibliothek*«. Danach steht dem Tcl-Skript das neue »hallo«-Kommando zur Verfügung (Abbildung 1).

Diese Erweiterung ist nur ein akademisches Beispiel – es gibt genug Aufgaben, die nicht mit reinem Tcl zu erledigen sind. Ein Beispiel ist das in der letzten Folge [9] erwähnte Messprogramm für Scheinwerfer. Sensor und Computer sind per serieller Schnittstelle verbunden, die beiden Schrittmotoren für den Scheinwerfer hängen am Parallelport.

Zugriff auf die parallele Schnittstelle

Die Ansteuerung der seriellen Schnittstelle ist mit Tcl kein Problem, der Interpreter bringt dafür schon die passenden Kommandos mit. Beim Parallelport ist aber noch Arbeit nötig. Als Vorlage für ein passendes Erweiterungsmodul dient das Parcon-Programm von Drew Pertulla [2]. Es besteht aus lediglich 42 Zeilen C-Code. Mit ihm kann man die Spannung an jedem Kontakt der parallelen Schnittstelle einzeln einstellen.

Die meisten Schrittmotor-Steuerungen benötigen aber einen Impuls pro Schritt. Beim gegebenen Messaufbau sind 360 Aufrufe pro Winkelgrad nötig. Das ausführbare Programm ist mit 9 KByte zwar klein, aber ein Aufruf dauert gut 1100 Mikrosekunden. Es bietet sich eine Tcl-Erweiterung an, die das langwierige Aufrufen eines komplette Programms durch ein flinkes Tcl-Kommando ersetzt.

Der Quelltext der Parcon-Erweiterung ist in Listing 2 zu sehen. Der Aufbau gleicht dem von Listing 1, wieder gibt es die Initialisierung (ab Zeile 19) und eine Kommandofunktion (ab Zeile 45). Zum Lesen und Beschreiben des Parallelports verwendet die Erweiterung die C-Funktionen »inb()« und »outb()«. Mit ihnen kann ein Programm seit Unix-Urzeiten direkt auf Ports zugreifen. Direkter Hardwarezugriff ist jedoch gefährlich, weshalb er Root vorbehalten bleibt.

In der Initialisierung prüft die Tcl-Erweiterung per »ioperm()« (Zeile 28), ob ihr der Zugriff möglich ist. Ist dies nicht der Fall, erfolgt eine Fehlermeldung. Das un-

Listing 2: Parallele Schnittstelle

```
01 /* Mit dieser einfachen Erweiterung kann ein
02  * Tcl-Programm unter Unix auf die parallele
03  * Schnittstelle schreiben. Die Erweiterung
04  * benötigt Root-Rechte, sie benutzt inb und
05  * outb auf Port 888.
06  */
07
08 #include <asm/io.h>
09 #include <stdio.h>
10 #include <sys/ioctl.h>
11 #include <tcl.h>
12
13 /* Vorwärtsdeklaration */
14 int parcon_Cmd (ClientData cdata,
15   Tcl_Interp *interp, int objc,
16   Tcl_Obj * CONST objv[]);
17
18 /* Erweiterung initiliasieren und Rechte prüfen */
19 int Parcon_Init (Tcl_Interp *interp)
20 {
21 #ifdef USE_TCL_STUBS
22   if (Tcl_InitStubs(interp, "8.1", 0) == 0L) {
23     return TCL_ERROR;
24   }
25 #endif
26
27   /* Zugriffsrecht auf parallele Schnittstelle?
   */
28   if (ioperm(888,1,1)) {
29     /* Fehlermeldung erzeugen */
30     Tcl_Obj *retval = Tcl_NewStringObj (
31       "Kann parallele Schnittstelle nicht
   öffnen.", -1);
32     Tcl_AppendStringsToObj (retval,
33       "\nBitte als Root benutzen", (char*) NULL
   );
34     Tcl_SetObjResult (interp, retval);
35     return TCL_ERROR;
36   }
37
38   /* parcon-Kommando erzeugen */
39   Tcl_CreateObjCommand(interp, "parcon",
40     parcon_Cmd, NULL, NULL);
41   return TCL_OK;
42 }
43
44 /* Das parcon-Kommando ausführen */
45 int parcon_Cmd (ClientData cdata,
46   Tcl_Interp *interp, int objc,
47   Tcl_Obj * CONST objv[])
48 {
49   /* entweder kein Argument oder eines */
50   if (objc > 2) {
51     Tcl_WrongNumArgs (interp, 1, objv,
   "?bitmap?");
52     return TCL_ERROR;
53   }
54
55   /* Eingabe überprüfen und Status setzen */
56   if (objc == 2) {
57     /* Länge prüfen */
58     if (Tcl_GetCharLength(objv[1]) != 8) {
59       Tcl_Obj *retval = Tcl_NewStringObj(
60         "Falsche Eingabe, benötige 8bit (
   \"00101010\" )", -1);
61       Tcl_SetObjResult (interp, retval);
62       return TCL_ERROR;
63     }
64
65     /* fiese Bitshifterei */
66     char* bitmap = Tcl_GetString (objv[1]);
67     int wert=0, i;
68     for (i=0; i<8; i++) {
69       if (strncmp("1", bitmap, 1) == 0) {
70         wert |= 1<<(7-i);
71       } else {
72         wert &= ~(1<<(7-i));
73       }
74       bitmap++;
75     }
76     //printf ("werte %i\n", wert);
77     /* Parallelport setzen*/
78     outb (wert,888);
79   }
80
81   /* Parallelport auslesen */
82   unsigned char bitmap = inb (888);
83
84   /* .. und Ergebnis formatieren */
85   Tcl_Obj *result = Tcl_NewStringObj ("", -1);
86   int i;
87   for (i=7; i>=0; i--) {
88     Tcl_AppendStringsToObj (result,
89       (bitmap&(1<<i)) ? "1":"0", (char*) NULL);
90   }
91
92   /* Ergebnis setzen */
93   Tcl_SetObjResult(interp, result);
94   return TCL_OK;
95 }
```

bequeme Zusammenfügen von Strings mit normalen C-Funktionen kann dank der Tcl-API entfallen: Zeile 30 erzeugt einen String, Zeile 32 fügt mit »Tcl_AppendStringsToObj()« weiteren Text hinzu. Die Funktion »Tcl_SetObjResult()« übergibt dem Interpreter das Ergebnis (hier die Fehlermeldung). Der Rückgabewert der C-Funktion »TCL_ERROR« signalisiert dem Interpreter, dass die Funktion auf einen Fehler gestoßen ist. Das Tcl-Programm erfährt davon direkt beim »load«-Aufruf.

Das Parcon-Kommando

War die Initialisierung erfolgreich, erzeugt Zeile 39 (Listing 2) das neue Kommando »parcon«. Alle Aufrufe des Kommandos leitet Tcl nun an die C-Funktion »parcon_Cmd()« weiter. Diese Funktion soll den Status der parallelen Schnittstelle abfragen und auf Wunsch ändern. Sie prüft daher zunächst die Übergabeparameter.

Die C-Funktion erhält ein Array mit »Tcl_Obj«-Objekten. Ähnlich wie bei einer »main()«-Funktion beginnt das Array mit dem Tcl-Kommandonamen als erstem Eintrag. Die weiteren Objekte enthalten dann die auf der Tcl-Seite angegebenen Kommandoparameter. Die Erweiterung prüft in Zeile 50 die Anzahl der Parameter sowie gegebenenfalls in Zeile 58 die Länge der Eingabe. Sie benutzt dazu wiederum String-Funktionen aus der Tcl-API.

Wenn das Ergebnis gültig ist, muss die Funktion noch das in einem String abgelegte Bitmuster (etwa »"00101010"«) in den jeweiligen Integer-Wert (hier 42) umwandeln. Die Bit-Schieberei dazu stammt aus dem Quelltext von Drew Pertulla. Der »outb()«-Aufruf in Zeile 78 versetzt mit dem eben ermittelten Wert die parallele Schnittstelle in den gewünschten Zustand.

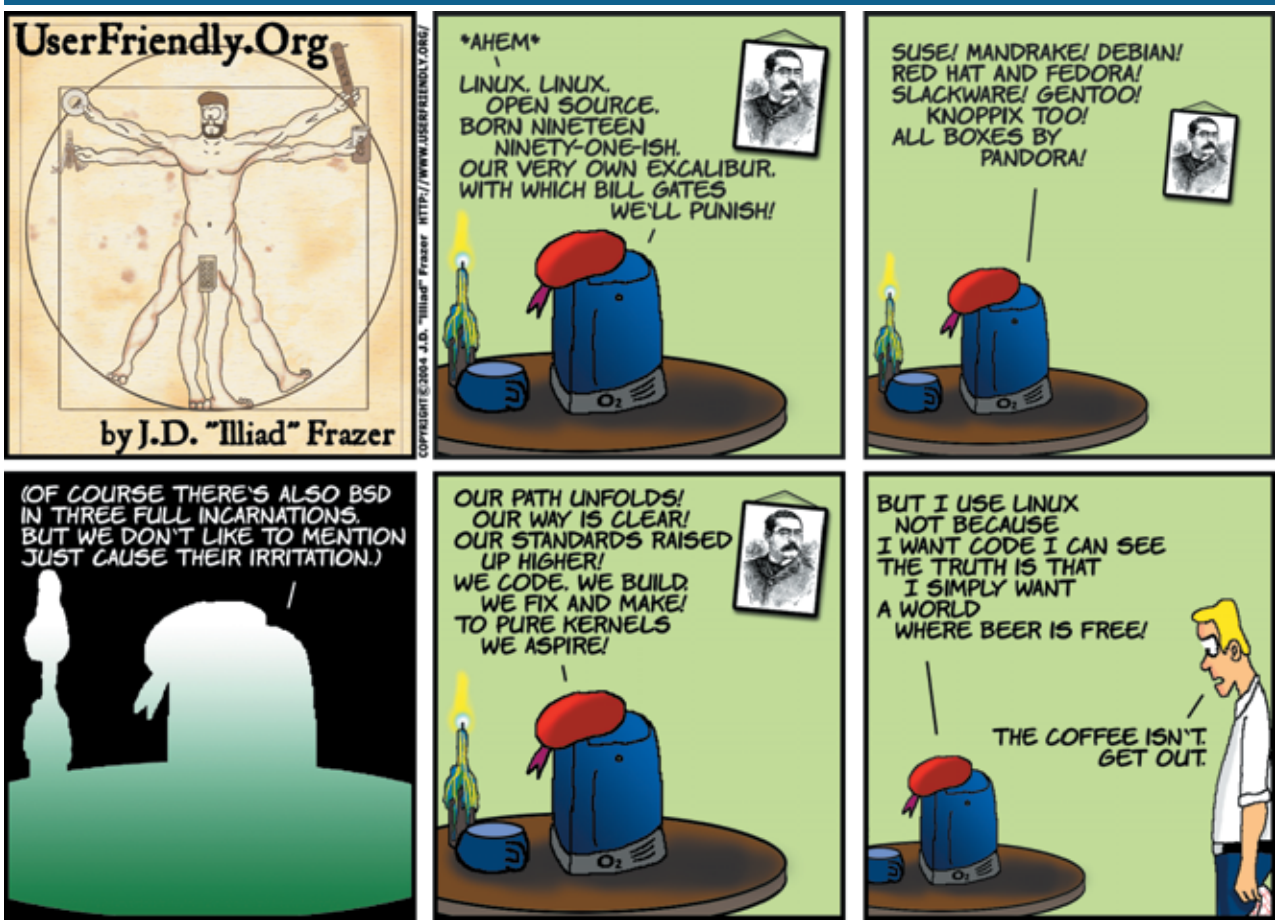
Als Nächstes liest »inb()« den aktuell gesetzten Wert des Parallelports, er ist für die Rückgabe in einen String zu verwandeln. Dabei kommt wieder die Tcl-API zum Zuge, sie ist bequemer als reines C. Bis auf das Bit-Schieben ist diese Tcl-Erweiterung nicht weiter schwierig – und das Ergebnis überzeugt: Statt 1100 Mikrosekunden für das externe Programm dauert ein Parcon-Aufruf nur noch 12 Mikrosekunden.

Plattformübergreifender Code ist aufwändiger

Diese Beispiele sind bewusst einfach gehalten, so läuft die zweite Erweiterung wegen »inb()« und »outb()« nur auf Unix-ähnlichen Systemen. Für plattformunabhängige Tcl-Module gibt die TEA-Spezifikation (Tcl Extension Architecture, [3]) den besten Weg vor. Auf der Webseite ist auch gleich ein Standardgerüst zu finden, das als Grundlage für eigene Erweiterungen dienen kann.

Wer nur vorhandene C-Bibliotheken von Tcl aus verwenden muss, kann sich das manuelle Schreiben der Kommandos so-

User Friendly – der monatliche Comic im Linux-Magazin



gar ganz sparen: Hierfür bietet sich das Werkzeug SWIG [4] an. Das Programm erzeugt auf Basis einer Spezifikation (Headerdateien mit optionalen Ergänzungen) fertige Wrapper um C- und C++-Bibliotheken. Mit dem Wrapper können Tcl und viele andere Skriptsprachen die Funktionen der Bibliothek verwenden.

Findet sich in der Fülle der verfügbaren Bibliotheken und Tcl-Erweiterung nichts Passendes für eine Aufgabe, greift ein gestandener Entwickler zur Selbsthilfe und schreibt sich eine Erweiterung. *(fjl)* ■

Infos

[1] Funktionen der C-API von Tcl: [http://www.tcl.tk/man/tcl8.4/TclLib/]
[2] Parcon: [http://bigasterisk.com/parallel]
[3] Tcl Extension Architecture, TEA: [http://www.tcl.tk/doc/tea/]
[4] SWIG: [http://www.swig.org]
[5] Tile: [http://tktable.sourceforge.net]
[6] Tk-Look: [http://tcllib.sourceforge.net/TkLook/]
[7] Ceptcl, Communications Endpoints for Tcl: [http://www.fivetones.net/software/]
[8] Scotty: [http://wwwhome.cs.utwente.nl/~schoenw/scotty/]
[9] Carsten Zerbst, „Bildhafte Kurven – Datenreihen mit Tcl-Programmen visualisieren“: Linux-Magazin 05/04, S. 106

Das Neueste

In der Tcl-Szene geht die Diskussion um das Aussehen von Tk unter Unix weiter. Die Entwickler verfolgen zwei Techniken, um die Darstellung zu verbessern: Joe English entwickelt das ambitionierte Tile-Paket [5], andere Programmierer begnügen sich mit neuen Basiseinstellungen. Das Tile-Paket ist eine Theme-Engine ähnlich der von GTK oder QT, die eigene Themes auf vorhandene Widgets anwendet. Unter Linux ist der große Vorteil von Tile, dass es auch fertige GTK-Themes verwenden kann.

In Abbildung 2 ist eine Demo-Anwendung mit dem Aquativ-Theme zu sehen. Eine einfache Alternative schlägt Jeffry Hobbs für die Tklib vor [6]. Er ändert nur die Standardeinstellung für Farben und Ränder in Tk und nähert sich so dem GTK-Standardtheme.

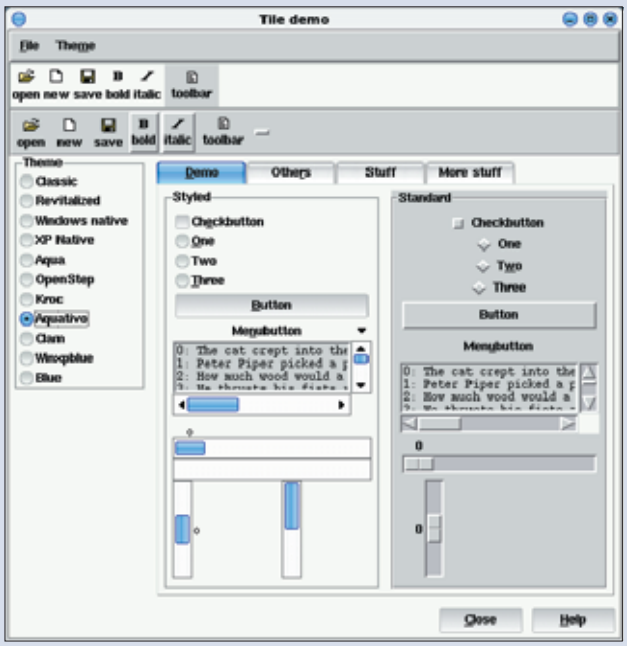


Abbildung 2: Mit dem Tile-Paket wird Tk anpassungsfähig: Dieses Demo-Programm zeigt das GTK-Theme Aquativ, angewendet auf eine Tk/Tile-Oberfläche.

Wie bereits im letzten Feder-Lesen angekündigt [9], ist das Diagrammpaket von Arjen Markus inzwischen in der Tklib enthalten. Die reine Tcl-Erweiterung zeichnet viele Diagrammtypen bis hin zu ansehnlichen 3D-Plots (Abbildungen 3a bis 3c).

Unter der Haube

Tcl unterstützt seit jeher die TCP/IP-Kommunikation. Mit der Ceptcl-Erweiterung [7] lernt es auch Verbindungen über Unix-Domain-Sockets und IPv6 aufzubauen. Über IP-Pakete kann ein Tcl-Programm dank Ceptcl nicht nur TCP-Verbindungen, sondern auch UDP-Datagramme und rohe Daten senden. Alternativ steht für Datagramme auch Scotty [8] zur Verfügung. Scotty wurde vor allem als leistungsfähiges SNMP-Paket bekannt.

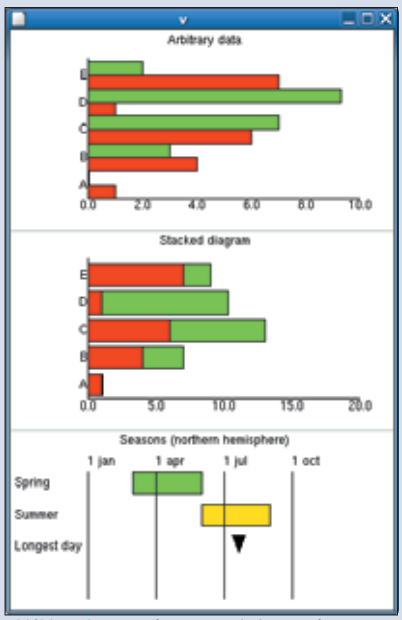


Abbildung 3a: Das Diagrammpaket von Arjen Markus stellt einfache Balkendiagramme ...

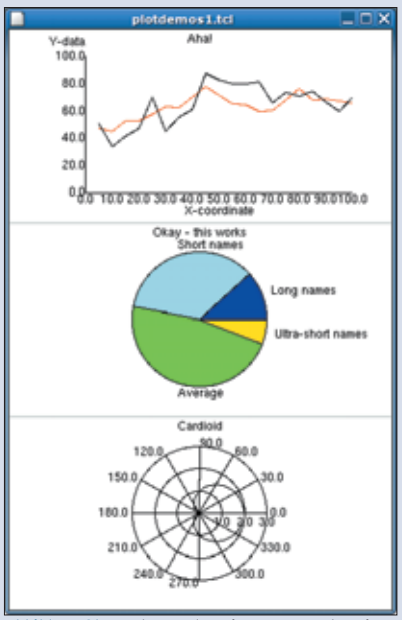


Abbildung 3b: ... ebenso dar wie Kurven und Kreise. Obwohl das Paket in reinem Tcl geschrieben ist, ...

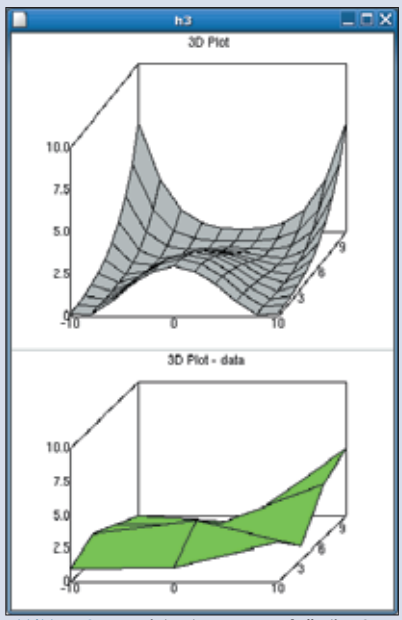


Abbildung 3c: ... zeichnet es sogar aufwändige 3D-Diagramme in bester Qualität.