Extensions

Scripts von C aus starten

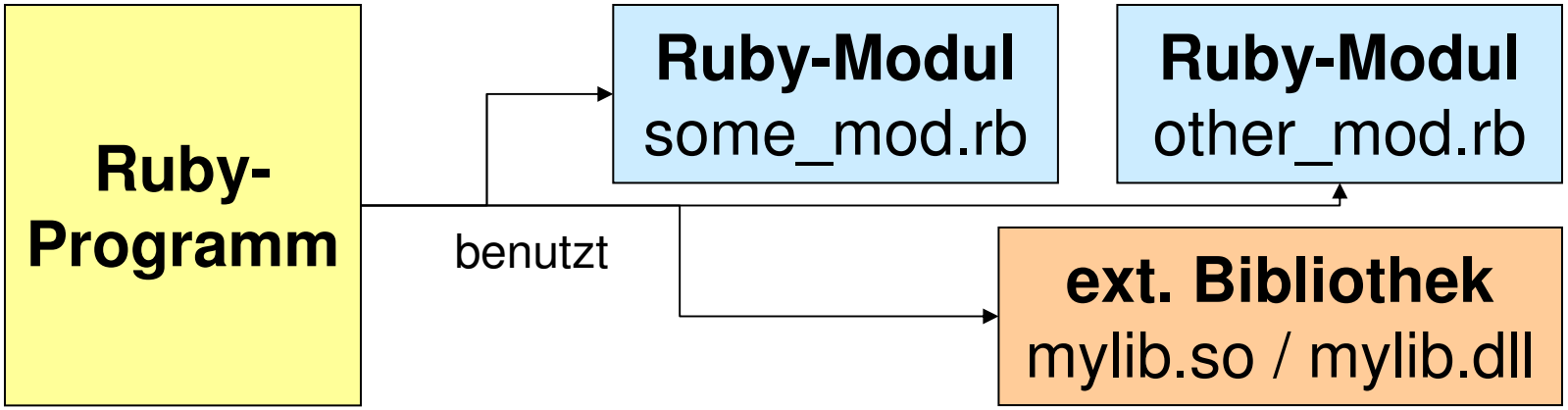
Script-Funktionen in C einbinden

C-Bibliotheken in der Skriptsprache nutzen: SWIG

Windows Extensions: Win32API, Win32OLE

Ruby erweitern - warum ?

- Benutzung vorhandener Schnittstellen
  - Existierende Bibliotheken einfach in Ruby mitbenutzen
  - Objektorientierte Kapselung älterer Komponenten
  - Hardwarenahe Programmierung
- Effizienz
  - Laufzeit- oder speicherplatzkritische Abschnitte in einer schlanken Compilersprache implementieren,
  - Integrationskomfort von Ruby beibehalten.



- Merke: Was Assembler für C, ist C für Ruby!

Ruby-Scripts von C aus starten

Ruby-Skripte von C aus verwenden

- Grundlage:
  - Ruby ist in C geschrieben, daher vorhanden:

```
ruby.h, C-Bibliotheken von Ruby-Funktionen
```

  - Embedded Ruby API:

```
void  ruby_init()
```

Immer als erstes aufzurufen

```
void  ruby_options(int argc, char **argv)
```

Kommandozeilenparameter an Ruby senden

```
void  ruby_script(char *name)
```

Name des Scripts setzen

```
void  rb_load_file(char *file)
```

Datei in den Interpreter laden

```
void ruby_run()
```

Script starten

Beispiel

```
#include "ruby.h"
main(int argc, char **argv) {
  /* ... unser Code ...*/
  /* Gelegentlich erforderlich:    */
#if defined(NT)
  NtInitialize(&argc, &argv);
#endif
  ruby_init();
  ruby_script("embedded");
  rb_load_file("start.rb");
  while (1) {
        if (need_to_do_ruby) {
              ruby_run();
        }
  /* Hier unser Anwendungscode ... */
  }
}
```

Ruby-Funktionen in C einbinden

Ruby-Funktionen in C einbinden

- Warum möglich?
  - Ruby selbst ist in C implementiert
  - Die Ruby-Bibliothek ist offen gelegt. Ihre Funktionen und Datenstrukturen können von beliebigen C-Programmen verwendet werden.
  - "ruby.h" und die Ruby-Laufzeitumgebung sind verfügbar
- Warum sinnvoll?
  - Effizienz: Stärken von Ruby in C nutzbar
  - Verständnis: Details des Übergangs C / Ruby
- Generelle Bemerkungen:
  - Die Verwendung von Ruby-Objekten in C erfordert ein gewisses Verständnis für den Aufbau von Ruby selbst.
  - Wichtig ist insbesondere das Umwandeln von Datentypen.
  - C-*Extensions* anderer Skriptsprachen sind komplizierter!

Ruby-Funktionen in C einbinden

- C-Implementierung einer Ruby-Klasse:

```
class Ministore
  def initialize
    @arr=Array.new
  end
  def add(anObj)
    @arr.push(anObj)
  end
  def retrieve
    @arr.pop
  end
end
```

Ruby-Funktionen in C einbinden

```
#include "ruby.h"
static VALUE t_init(VALUE self)
{  // Funktion zu "initialize"
  VALUE arr;
  arr = rb_ary_new();
  rb_iv_set(self, "@arr", arr);
  return self;
}

static VALUE t_add(VALUE self,
   VALUE anObj)
{  // Funktion zu "add"
  VALUE arr;
  arr = rb_iv_get(self, "@arr");
  rb_ary_push(arr, anObj);
  return arr;
}
//static VALUE t_retrieve(...){}
// gemeinsam an der Tafel
```

```
// Klasse ist globale Konstante:
VALUE cMinistore;

// Registrierung der Methoden:
void Init_Test() {
  cMinistore = rb_define_class(
               "Ministore",
               rb_cObject);
  rb_define_method(
               cMinistore,
               "initialize",
                t_init, 0);
  rb_define_method(
               cMinistore,
               "add",
               t_add, 1);
  // Fall "retrieve":
  // gemeinsam an der Tafel
}
```

Ruby-Funktionen in C einbinden

- Das Beispiel zeigte:
  - Erzeugen einer neuen Klasse
  - Erzeugen & Registrieren von Methoden, Verbindung mit C-Funktionen zur eigentlichen Arbeit
  - Erzeugen von Ruby-Variablen bzw. -Attributen, Verbindung von Attributen mit Klassen, Abrufen & Verändern von Werten.
- Zugriff auf Variablen (kleine Auswahl):

```
// Liefert "instance variable" zu name
VALUE rb_iv_get(VALUE obj, char *name)
// Setzt/ändert Wert der "instance variable" zu name
VALUE rb_iv_set(VALUE obj, char *name, VALUE value)
// Analog für globale Variablen bzw. Klassenattribute:
VALUE rb_gv_get/set, VALUE rb_cv_get/set
// Erzeugen von Objekten eingebauter Klassen:
VALUE rb_ary_new(), VALUE rb_ary_new2(long length), ...
VALUE rb_hash_new(), VALUE rb_str_new2(const char *src)
```

Ruby-Funktionen in C einbinden

- Der generische Datentyp VALUE:
  - Dahinter verbirgt sich eine Datenstruktur mit Verwaltungsinformation.
  - Bei der Umwandlung in C-Datentypen ist "*typecasting*" erforderlich.
  - Eine Reihe eingebauter Makros und Funktionen hilft dabei.
  - *Low-level* Beispiele:

```
VALUE str, arr;
RSTRING(str)->len    // Länge des Ruby-Strings
RSTRING(str)->ptr    // Zeiger zum Speicherbereich (!)
RARRAY(arr)->capa    // Kapazität des Ruby-Arrays
```

- Empfehlung:
  - Ausweichen auf *high-level* Makros (s.u.), Ruby-Interna vermeiden!
- Direkte Werte (*immediate values*):
  - Objekte der Klassen Fixnum und Symbol sowie die Objekte true, false und nil werden direkt in VALUE gespeichert; kein Zeiger auf Speicher

Ruby-Funktionen in C einbinden

- Umwandlung zwischen C-Typen und Ruby-Objekten...
  - ...mittels einer Reihe vordefinierter Makros und Funktionen. Beispiele:

```
INT2NUM(int) // Liefert ein Fixnum-bzw. Bignum-Objekt
INT2FIX(int)         // Fixnum (schneller als INT2NUM)
CHR2FIX(char)        // Fixnum
rb_str_new2(char *) // String
rb_float_new(double)// Float
```

```
int NUM2INT(Numeric)// incl. Typenüberprüfung
int FIX2INT(Fixnum) // schneller
unsigned int NUM2UINT(Numeric)           // analog
// usw.
char NUM2CHR(Numeric or String)
char * STR2CSTR(String)                  // ohne Länge
char * rb_str2cstr(String, int *length) // mit Länge
double NUM2DBL(Numeric)
```

Ruby-Funktionen in C einbinden

- Daten zwischen C und Ruby "teilen"?
  - Vorsicht - Ruby-Interna können sich ändern. Versuchen Sie möglichst nicht, die von Ruby genutzten internen Speicherbereiche von C aus (etwa per Pointer) zu verändern oder zu benutzen!
  - Besser: Gezielte Umwandlung incl. Kopieren
  - Bsp.: String in Ruby erlaubt NULL-Zeichen, char * in C nicht!
- Speicherallokation in C vs. Rubys *Garbage Collector*
  - Hier stoßen zwei verschiedene Konzepte aufeinander!
  - Anpassung erfordert Handarbeit, Bsp.: Kein free() auf Ruby-Objekte!
  - Vorsicht: Rubys GC kann von C aus angelegte Ruby-Objekte jederzeit löschen, wenn sie nicht ordentlich in Ruby "registriert" sind.
  - Rubys GC kann umgekehrt auch veranlasst werden, dynamisch erzeugte C-Strukturen bei Bedarf wieder freizugeben.
- Wir werden hier das Thema GC nicht weiter vertiefen.
  - Bei Bedarf: Pickaxe-Buch 2, Kap. 21 sowie Ruby Dev. Guide Kap. 10.

C-Bibliotheken in Ruby nutzen

C-Bibliotheken in Ruby nutzen

- Frage / Diskussion:
  - Was wäre alles notwendig, um C-Funktionen einer gegebenen Bibliothek (oder auch einer eigenen C-Objektdatei) von Ruby aus zu benutzen?
  - Beispiel:

```
double get_current_stock_price(
                        const char *ticker_symbol)
```

- Antworten sammeln (Tafel), z.B.:
  - Top-Level Methode eines Ruby-Moduls (etwa: "Stockprice")

```
Stockprice::get_current_stock_price(aString) --> aFloat
```

  - Umwandlungen

```
Ruby-String --> C-String    // Argument
double --> Float-Objekt     // Rückgabewert
```

  - Generierung eines Ruby-Moduls; evtl. Initialisierungen
  - "Wrapper", der die Ruby-Methode auf die Bibliotheksfunktion abbildet.

C-Bibliotheken in Ruby nutzen

- Die gute Nachricht:
  - **All dies lässt sich weitgehend automatisieren!**
- Bewährtes Hilfsmittel:
  - **SWIG** (***S****implified* ***W****rapper and* ***I****nterface* ***G****enerator*) !
  - Ein *Open Source-Tool*, seit V 1.3 auch mit Ruby-Modus

C-Bibliotheken in Ruby nutzen

- Vorgehen (einfacher Fall):

1. SWIG mitteilen, was zu verbinden ist (in Datei "stockprice.i"):

```
%module Stockprice
%{
#include "stockprice.h"
%}
extern double get_current_stock_price(const char *);
```

2. Mit SWIG eine Wrapper-Datei generieren:

```
$ swig -ruby stockprice.i  # stockprice_wrap.c erzeugt!
```

3. Makefile generieren, mit Unterstützung von Ruby:

```
$ ruby -r mkmf -e"create_makefile('Stockprice')"
# Erzeugt "Makefile" (mit Bezug auf "alle" C-Dateien)
```

4. *Shared Object*-Datei erzeugen:

```
$ make          # Generiert shared object "Stockprice.so"
```

5. Ruby-Anwendung starten

```
# Siehe nächste Seite
```

C-Bibliotheken in Ruby nutzen

- Neues Modul in Ruby-Anwendung nutzen (Datei stock.rb):

```
#!/usr/bin/env ruby
require "Stockprice"          # Ruby findet die *.so-Datei!

def prices_of( symbols )      # Beispiel für eine Nutzung
  prices = {}
  symbols.each do |sym|       # C-Funktion anwenden:
    prices[sym]=Stockprice::get_current_stock_price(sym)
  end
  prices
end
```

```
puts "Current NYSE values" # Anwenden:
prices_of( %w( F G GE LU MRK MOT TWX )).each do |sym,v|
   puts "%4s\t%6.2f USD" % [sym, v]
end
```

- Bemerkungen:
  - Ganz natürliche Verwendung, wie Ruby-Modul!
  - Vergleiche auch die Funktionen des Moduls "Math"

Die Situation in anderen Skriptsprachen

- Bei Perl und Python gibt es viele grundsätzliche Gemeinsamkeiten
- SWIG unterstützt auch Perl und Python

Python

- Das Vorgehen ist ähnlich, siehe z.B.

  `http://www.penzilla.net/tutorials/python/swig/`
- Generierung eines passenden Makefiles?

Perl

- Eigene Erfahrung (bis 2003): Möglich, aber auch mit SWIG eine umständliche Angelegenheit. Einbindung von SAP's RFC: ☹

Groovy

- Hier steht die Einbindung von Java-Paketen im Vordergrund
- Diese lassen sich sogar „nativ" ansprechen
- C/C++: Bibliotheken in Java via JNI einbinden, „nativ" von Groovy verwenden. Tipp: SWIG generiert auch Wrapper-Code für JNI !

Ruby und Windows

Beispiele für Ruby-*Extensions*:

Win32API und Win32OLE

Win32API

- *Extension*-Modul speziell für MS-Windows
- ***Low-level*** Zugriff auf alle Funktionen aus dem Win32 API.
- Hinweise:
  - Viele dieser Funktionen benötigen oder liefern einen Zeiger auf einen Speicherbereich.
  - Ruby verwaltet Speicherblöcke über Objekte der Klasse String.
  - Geeignetes Umcodieren von bzw. in brauchbare Darstellungen bleibt dem Anwender überlassen - z.B. mittels "pack" / "unpack".
- 1 Klassenmethode:

```
Win32API.new( dllname, procname, importArray, export )
```

`dllname`	`z.B. "user32", "kernel32"`
`procname`	`Name der aufzurufenden Funktion`
`importArray`	`Array von Strings mit Argumenttypen`
`export`	`String mit Typ des Rückgabewerts`

Win32API

- Typencodes:
  - (Klein- wie auch Großbuchstaben sind zulässig)

```
"n", "i"    Zahlen,
"p"         Zeiger auf (in Strings gespeicherte)
            Daten,
"v"         Void-Typ (nur Fall "export").
```

- 1 normale Methode:

```
call / Call ( [args]* ) ==> anObject
```

  - Die Anzahl und Art der Argumente wird von "`importArray`", die Objektklasse wird von "`export`" in `new()` bestimmt.

Win32API

- Beispiel:

```
require 'Win32API'

getCursorPos =
   Win32API.new("user32", "GetCursorPos", ['P'], 'V')

lpPoint = " " * 8    # Platz für zwei 'long'-Plätze
getCursorPos.Call( lpPoint )

x, y = lpPoint.unpack("LL")        # Decodieren
print "x: ", x, "\ty: ", y, "\n"  # Ausgeben
```

- Demos:
  - point.rb
  - win32api01.rb (erweitert, mit Schleife)

Win32API

- Beispiel (Forts.):

```
# Debug-Nachricht senden:
ods = Win32API.new("kernel32", "OutputDebugString",
                   ['P'], 'V')
ods.Call( "Hello, World\n" )

# Bildschirmschoner aktivieren:
GetDesktopWindow =
   Win32API.new("user32", "GetDesktopWindow", [], 'L')
GetActiveWindow =
   Win32API("user32", "GetActiveWindow", [], 'L')
SendMessage =
   WinAPI32("user32", "SendMessage", ['L'] * 4, 'L')
SendMessage.Call(GetDesktopWindow.Call, 274, 0xf140, 0)
```

Win32API

- Eigenes Beispiel: Einfache Töne

```
PlayNote =
   Win32API.new("kernel32", "Beep", ['I', 'I'], 'V')
PlayNote.call( 440, 500 )   # Kammerton A, 500 ms lang
```

- Eigenes Beispiel: MessageBox

```
GetDesktopWindow =
   Win32API.new("user32", "GetDesktopWindow", [], 'L')
MBox = Win32API.new("user32", "MessageBox",
                    ['L', 'P', 'P', 'L'], 'L')
rc = MBox.call( GetDesktopWindow.call,
                    "Meine Nachricht", "Boxtitel", 1 )
```

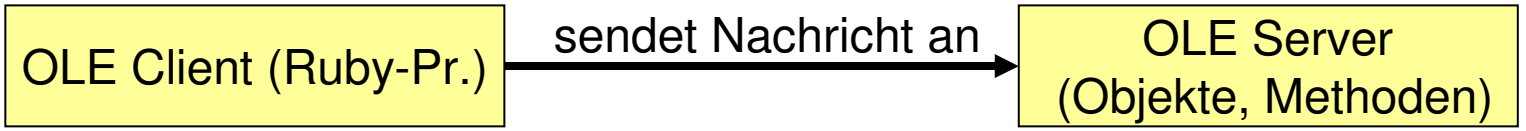
- Hal's Beispiel: *Unbuffered character input*
  - Problem:

    Eingabe eines Passwords am Bildschirm verdecken,
    indirektes Echo von "*" statt der gedrückten Tasten.
  - Demo dazu:

    pw_prompt.rb

Win32OLE

- ***High-level*** Zugriff auf Win 32 OLE *Automation Server*
  - Ruby ersetzt hier z.B Visual Basic (VB) oder den *Windows Scripting Host*.
  - Ruby tritt an die Stelle eines OLE *clients,* d.h. OLE Server wie MS Excel, Word, PowerPoint etc. können oo. "ferngesteuert" werden:



- *Mapping* der OO-Modelle: (vgl. Code-Beispiele unten)
  - Methoden eines OLE-Servers werden über gleich lautende Ruby-Methoden aktiviert,
  - Parameter der OLE-Objekte werden durch Abfragen & Setzen entsprechender Ruby-Attribute (genauer: Getter/Setter) kontrolliert.
  - Zugriff auf Eigenschaften erfolgt per Hash-Notation.

Win32OLE

- Hinweise:
  - OLE-Objekte und Eigenschaften verrät die Online-Hilfe von VBA.
  - Tipp:
    - VBA-Makro mit OLE-Server wie z.B. Excel aufzeichnen,
    - Einzelheiten ggf. in VB-Online-Dokumentation nachschlagen,
    - dann mit Ruby nachbauen / automatisieren.
  - Ruby-Methoden werden wie üblich klein geschrieben, auch wenn die entsprechenden Windows-Objekte groß geschrieben werden.

Win32OLE

- Einfaches Beispiel:
  - IE starten, eingestellte *homepage* anzeigen lassen

```
require 'win32ole'

ie = WIN32OLE.new( 'InternetExplorer.Application' )
ie.visible = true
ie.gohome
```

  - **Demo mit irb!**

- **Konvention** für benannte Argumente

VB-Def.:	**Song(artist, title, length):**
VB-Aufruf:	**Song title := 'Get it on';**
Ruby-Aufruf, simpel:	**Song(nil, 'Get it on', nil)**
Ruby-Aufruf, smart:	**Song('title' => 'Get it on')**

  - Vorteile: Frei von spezieller Reihenfolge, selbst-dokumentierend.

Win32OLE

- **OLE Demo mit Excel**
  - Vgl. Demo-Datei "09/win32ole01.rb und das Pickaxe-Buch, Kap. "Ruby and MS Windows", S. 168f.

- **Optimierungshinweise**

```
# Vorsicht - ineffizient:
workbook.Worksheets(1).Range("A1").value = 1
workbook.Worksheets(1).Range("A2").value = 2
workbook.Worksheets(1).Range("A3").value = 4
workbook.Worksheets(1).Range("A4").value = 8
```

```
# Besser so:
worksheet = workbook.Worksheets(1)
worksheet.Range("A1").value = 1
# usw.
worksheet.Range("A4").value = 8
```