Introduction to Red

From system programming to scripting
Introducing myself

• Nenad Rakocevic, 40, french

• Programming since 25 years:
  – in C/C++, *Basic, ASM, REBOL, web client-side languages,…
  – was a happy Amiga user and registered BeOS developer

• Founder of two software companies in Paris:
  – Softinnov
  – ElasticSoft

• Author of several libraries for REBOL:
  – MySQL, PostgresQL, LDAP native drivers
  – UniServe: asynchronous, event-driven network engine
  – Cheyenne Web Server: full-featured web application server
  – CureCode: very fast web-based bug tracker (Mantis-like)
Why make yet another language?

To build an efficient new tool.

To have an open source implementation of REBOL language.

It is a very exciting challenge.
What is REBOL?

- Invented by Carl Sassenrath (AmigaOS kernel father)
- Available since 1998
- Abandoned since a year by its author
- Closed source (part of standard library has been opened)

- Interpreted
- Multi-paradigm (imperative, functional, OO, declarative)
- Strong meta-programming abilities
Red quick tour: Genealogy

- LISP
  - Forth
  - Logo
- REBOL
- Lua
- Scala
- Red
Red quick tour: Natural range of application

Abstraction level

- DSL
- Scripting
- Applications
- OS
- Drivers
- Hardware

Languages:
- C
- Pascal
- Java
- C++
- Ruby
- Javascript
- Python
- REBOL
- Red
- Red/System

Hardware

Drivers

OS

Applications

Scripting

DSL
Red quick tour: Language stack

Red → Red/System → Native code

Red compiles to Red/System
Red/System compiles to Native code
Red quick tour: Red vs Red/System (1/6)

Memory management

Red

Automatic

Compacting garbage collector

Red/System

Manual

Allocate / Free
Red quick tour: Red vs Red/System (2/6)

<table>
<thead>
<tr>
<th>I/O</th>
<th>High level abstractions</th>
<th>Very limited native I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Red/System</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Red quick tour: **Red vs Red/System (3/6)**

<table>
<thead>
<tr>
<th>Type system</th>
<th>Red</th>
<th>Red/System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rich, more than 50 types</td>
<td>Poor, 6 primitives types</td>
<td></td>
</tr>
<tr>
<td>Sophisticated type inference</td>
<td>Limited type inference</td>
<td></td>
</tr>
</tbody>
</table>
Red quick tour: **Red vs Red/System** (4/6)

### Boxed values

- **Red**
  - Values
    - Header: 32 bits
    - Payload: 96 bits
    - 123

- **Red/System**
  - Values
    - Header: 32 bits
    - Payload: 123

### Primitives values

- **Red**
  - 123

- **Red/System**
  - 123
**Red quick tour: Red vs Red/System** (5/6)

<table>
<thead>
<tr>
<th>Performances</th>
<th>Red</th>
<th>Red/System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reasonably fast, C performances by a factor of 10-20</td>
<td></td>
<td>Very fast, C performances by a factor of 1-3</td>
</tr>
</tbody>
</table>
Red quick tour: Red vs Red/System (6/6)

Red

Completion

Red/System

No compiler yet

Beta state

Runtime partially done
Simplicity

An IDE should not be necessary to write code.
Compactness

*Being highly expressive maximizes productivity.*
Speed

*If too slow, it cannot be general-purpose enough.*
Be **Green**

Have a Small Footprint

*Because resources are not limitless.*
Red quick tour: Goals (5/7)

Ubiquity

Spread everywhere.
Portability
Write once run everywhere

That’s the least expected from a programming language.
Flexibility

*Best Good fit for any task!*
Red quick tour: Some features…

- Same syntax as REBOL, most of its semantic rules
- Strong DSL / dialecting support
- Red/System dialect inlined in Red

- (JIT) compiled instead of interpreted
- Statically typed + type inference
- Embeddable: distributed as a shared library with host languages bindings

- Concurrency support
  - Task parallelism: using "actor" abstraction
  - Data parallelism: using parallel collections
Red quick tour: Types tree

Rich type system: up to 50 first-class datatypes

- value!
  - none!
  - logic!
  - symbol!
    - word!
    - lit-word!
    - set-word!
  - ...!

- context!
  - object!
  - function!
  - module!
    - port!
    - integer!
    - date!
    - time!
    - char!
    - ...!

- scalar!
  - error!
    - ...!

- series!
  - block!
    - path!
    - hash!
    - list!
    - map!
    - ...!

- binary!

- string!
  - file!
  - url!
  - email!
  - tag!
  - issue!
  - ...!
Red quick tour: Target platforms

**Embedded**
- Android devices
- iOS devices
- Arduino boards (AVR 8/32-bit)

**Desktop**
- Windows
- Linux
- MacOS X
- Syllable
- FreeBSD
- Haiku

VM

.NET
AVM2 (Flash)
Javascript

JVM
Red quick tour: bootstrapping

1. Red and Red/System compilers written in REBOL

2. Red/System compiler rewritten in Red

3. Red compiler rewritten in Red

4. Red JIT-compiler written in Red
Red quick tour: Project

• BSD license

• Source code hosted on Github since March 2011
  – version 0.2.1, 3 committers, 537 public commits
  – 175 tickets (164 closed)
  – 8614 unit tests (framework built by Peter WA Wood)
  – 260 KiB of sources for Red/System
  – 3800 LOC for Red/System compiler
  – 2200 LOC for Red/System linker
Red quick tour: Planning

• **Sept. October 2011:**
  – **beta** alpha of Red (no JIT)
  – **alpha** beta of ARM support
  – alpha of the IDE

• **Dec. January 2012:**
  – V1.0 beta of Red (no JIT)
  – beta of the IDE

• **Q1 2012:**
  – beta of Red JIT-compiler
  – V1.0 of Red
  – v1.0 of IDE
Red online channels

- Home: [http://red-lang.org](http://red-lang.org)
- Twitter: #red_lang
- IRC channel: #red-lang on freenode
- Mailing list hosted on Google Groups
Red/System: features (1/2)

- Purely imperative, C-level language, with a Red syntax
- Statically compiled (naïve compilation for now)
- Limited type system:
  - Logic!, byte!, integer!, struct!, pointer!, c-string!
  - Simple type inference
  - Type casting supported
- Compiler directives: #define, #include, #import, #syscall, #if, #either, #switch, ...
- Low-level CPU support (interruptions, I/O, stack, privileged mode)
- Inlined ASM support
Red/System: features (2/2)

• Linker
  – Link-time shared libraries binding
  – Output types: Exe, Shared library, Static library
  – Formats: PE, ELF, mach-o, Intel-hex
  – Link third-party static libraries

• Targets: IA-32, ARM, JVM, AVM2, x64, CLR

• Red/System as an inlined dialect in Red
Red/System: Hello world!

Red/System [
    title: "Hello World demo"
]

print "hello world!"
Red/System: variables and expressions

a: 1
b: a + 2 * 4
c: a < b
d: "hello"

if a < b [print "b is greater"]
either a < b [print "b"] [print "a"]

print either a < b ["b"] ["a"]

print [a " is less than " b "," c "," d]
1 is less than 12, true, hello
print-wide [a "is less than" b c d]
1 is less than 12 true hello
Red/System: functions

nothing: function [][]

print-zero: function [n [integer!]][
    print either zero? n ["zero"]["not zero"]
]

abs: function [n [integer!]] return: [integer!][
    either positive? n [n][negate n]
]

uppercase: function [s [c-string!] /local offset][
    offset: #"a" - #"A"
    if any [#"a" <= b/1 b/1 <= #"z"][
        s/1: s/1 + offset
    ]
]
Red/System: shared libraries

```red
#import [
   "libc.so.6" cdecl [
      allocate: "malloc" [
         size [integer!]
         return: [byte-ptr!]
      ]
      free: "free" [
         block [byte-ptr!]
      ]
      quit: "exit" [
         status [integer!]
      ]
      printf: "printf" [[variadic]]
   ]
]
printf ["%i %s" 123 "hello"]
123 hello
```
Red/System: CPU low-level features

timer-handler: func [[interrupt]][...]

#interruptions [  
  0000h :reset  
  0004h :timer-handler  
]

a: read-io 0376h  
write-io 0376h 1

a: get-modes privileged  
set-modes privileged false

set-modes interrupt true  
set-modes interrupt-mask FF000000h
### Red/System: keywords

<table>
<thead>
<tr>
<th>Operator</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%</code></td>
<td>all</td>
</tr>
<tr>
<td><code>*</code></td>
<td>and</td>
</tr>
<tr>
<td><code>+</code></td>
<td>any</td>
</tr>
<tr>
<td><code>-</code></td>
<td>as</td>
</tr>
<tr>
<td><code>-**</code></td>
<td>assert</td>
</tr>
<tr>
<td><code>//</code></td>
<td>comment</td>
</tr>
<tr>
<td><code>///</code></td>
<td>declare</td>
</tr>
<tr>
<td><code>&lt;</code></td>
<td>either</td>
</tr>
<tr>
<td><code>&lt;=</code></td>
<td>exit</td>
</tr>
<tr>
<td><code>&gt;</code></td>
<td>false</td>
</tr>
<tr>
<td><code>&gt;&gt;</code></td>
<td>func</td>
</tr>
<tr>
<td><code>&gt;&gt;&gt;</code></td>
<td>function</td>
</tr>
<tr>
<td><code>alias</code></td>
<td>if</td>
</tr>
<tr>
<td><code>or</code></td>
<td>not</td>
</tr>
<tr>
<td><code>pop</code></td>
<td>push</td>
</tr>
<tr>
<td><code>return</code></td>
<td>size?</td>
</tr>
<tr>
<td><code>true</code></td>
<td>size?</td>
</tr>
<tr>
<td><code>until</code></td>
<td>while</td>
</tr>
<tr>
<td><code>xor</code></td>
<td>while</td>
</tr>
<tr>
<td><code>case</code></td>
<td>switch</td>
</tr>
<tr>
<td><code>repeat</code></td>
<td>loop</td>
</tr>
<tr>
<td><code>get-modes</code></td>
<td>read-io</td>
</tr>
<tr>
<td><code>set-modes</code></td>
<td>write-io</td>
</tr>
</tbody>
</table>
Red/System: library bindings

- C library binding
- cURL binding
- ZeroMQ binding
- SDL binding
- GTK binding

All written by Kaj de Vos.

Let see a few demos written with these bindings…
Arduino Uno

Microcontroller ATmega328
Flash Memory **32 KB** of which 0.5 KB used by bootloader
SRAM **2 KB**
EEPROM 1 KB (ATmega328)
Clock Speed 16 MHz