Rebol

Rebol (/ˈrɛbəl/ *REB-əl*; historically **REBOL**) is a cross-platform[6] data exchange language and a multi-paradigm dynamic programming language designed by Carl Sassenrath for network communications and distributed computing. It introduces the concept of dialecting: small, optimized, domain-specific languages for code and data,[6][7] which is also the most notable property of the language according to its designer Carl Sassenrath:

> Although it can be used for programming, writing functions, and performing processes, its greatest strength is the ability to easily create domain-specific languages or dialects
>
> — Carl Sassenrath[8]

Douglas Crockford, known for his involvement in the development of JavaScript, has described Rebol as "a more modern language, but with some very similar ideas to Lisp, in that it's all built upon a representation of data which is then executable as programs" and as one of JSON's influences.[5]

Originally, the language and its official implementation were proprietary and closed source, developed by REBOL Technologies. Following discussion with Lawrence Rosen,[9] the Rebol version 3 interpreter was released under the Apache 2.0 license on December 12, 2012.[10] Older versions are only available in binary form, and no source release for them is planned.

Rebol has been used to program Internet applications (both client- and server-side), database applications, utilities, and multimedia applications.[6]

Contents

Rebol

Paradigm	language oriented programming, data exchange, functional, prototype-based, imperative
Designed by	Carl Sassenrath
Developer	REBOL Technologies
First appeared	1997
Stable release	2.7.8 / January 2011
Preview release	2.101.0 / December 2012
Typing discipline	dynamic, strong
OS	cross-platform
License	2.7.8 is Freely redistributable software,[1] 2.101.0 has Apache 2.0 license[2]
Filename extensions	.r, .reb[3]
Website	www.rebol.com (http://www.rebol.com)
Influenced by	
Self, Forth, Lisp, Logo[4]	
Influenced	
JSON,[5] Red	
Rebol Programming at	

Etymology

Rebol was initially an acronym for **R**elative **E**xpression **B**ased **O**bject **L**anguage written in all caps.[6][8] To align with modern trends in language naming represented, e.g. by the change replacing historical name *LISP* by *Lisp*, programmers ceased the practice of writing *REBOL* in all caps. Sassenrath eventually put the naming question to the community debate on his blog.[11] In subsequent writing, Sassenrath adopted the convention of writing the language name as *Rebol*.[12]

History

First released in 1997, Rebol was designed over a 20-year period by Carl Sassenrath, the architect and primary developer of AmigaOS, based on his study of denotational semantics and using concepts from the programming languages Lisp, Forth, Logo, and Self.

1. REBOL Technologies was founded in 1998.
2. *REBOL 2*, the interpreter, which became the core of extended interpreter editions, was first released in 1999.
    1. *REBOL/Command*, which added strong encryption and ODBC access, was released in September 2000.
    2. *REBOL/View* was released in April 2001, adding graphical abilities on the core language.
    3. *REBOL/IOS*, an extensible collaboration environment built with REBOL was released in August 2001.
    4. *REBOL/SDK*, providing a choice of kernels to bind against, as well as a preprocessor, was released in December 2002.
3. *Rebol 3* [R3], the newest version of the interpreter, had alpha versions released by REBOL Technologies since January 2008. Since its release as an Apache 2 project in December 2012, it is being developed by the Rebol community.[13] The last commit was in March 2014, so it's safe to assume that development has stalled.

Design

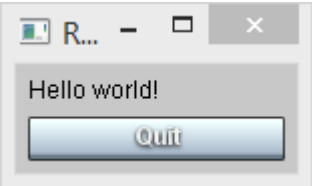
Ease of use

One of the Rebol design principles is "to do simple things in simple ways".[6] In the following example the *Visual interface dialect* is used to describe a simple Hello world program with a graphical user interface:

```
view layout [text "Hello world!" button "Quit" [quit]]
```

This is how a similar example looks in R3-GUI:

```
view [text "Hello world!" button "Quit" on-action [quit]]
```



Dialects

Rebol domain-specific languages, called *dialects*, are micro-languages optimized for a specific purpose. Dialects can be used to define business rules, graphical user interfaces or sequences of screens during the installation of a program. Users can define their own dialects, reusing any existing Rebol word and giving it a specific meaning in that dialect.[6] Dialects are interpreted by functions processing Rebol blocks (or parsing strings) in a specific way.

An example of Rebol's dialecting abilities can be seen with the word `return`. In the *data exchange dialect* `return` is just a word not having any specific meaning. In the *do dialect*, `return` is a global variable referring to a native function passing back a function result value.[4] In the *visual interface dialect (VID)*, `return` is a keyword causing the layout engine to simulate a carriage return, moving the "rendering pen" down to the beginning of the next line.[7]

A Rebol interpreter with graphical abilities must understand and interpret many dialects. The table below lists the most important ones in order of significance.

Dialect name	Interpreted by	Purpose
Data exchange dialect	`load` function	represents data and metadata; common platform for Rebol dialects
Do dialect	`do` function	programming
Parse dialect	`parse` function	pattern matching
Function specification dialect	`make` function	function definition; functional programming
Object specification dialect	`make` function	object definition/inheritance; prototype-based programming
Visual interface dialect (VID) or RebGUI	`layout` function or `display` function	specifies graphical user interface
Draw dialect	`view` function	defines graphical elements (lines, polygons, etc.)
Script specification dialect	`do` function	script definition
Security policy dialect	`secure` function	specifies security policy

Syntax

Rebol syntax is free-form, not requiring specific positioning. However, indentation is often used to better convey the structure of the text to human readers.

Syntactic properties of different dialects may differ. The common platform for all Rebol dialects is the *data exchange dialect*; other dialects are usually derived from it. In addition to being the common platform for all dialects, the *data exchange dialect* is directly used to represent data and metadata, populate data structures, send data over Internet, and save them in data storage.

In contrast to programming languages like C, the *data exchange dialect* does not consist of declarations, statements, expressions or keywords. A valid *data exchange dialect* text stream is a tree data structure consisting of blocks (the root block is implicit, subblocks are delimited by square brackets), parens (delimited by round brackets), strings (delimited by double quotes or curly brackets suitable for multi-line strings; caret notation is used for unprintable characters), URLs, e-mail addresses, files, paths or other composite values. Unlike ALGOL blocks, Rebol blocks are composite values similar to quoted s-expressions in Lisp. The fact that code is written in the form of Rebol blocks makes the language homoiconic.[4]

Blocks as well as parens may contain other composite values (a block may contain subblocks, parens, strings, ...) or scalar values like words, set-words (words suffixed by the colon), get-words (words prefixed by the colon), lit-words (words prefixed by the apostrophe), numbers, money, characters, etc., separated by whitespace. Note that special characters are allowed in words, so `a+b` is a word unlike `a + b`, which is a sequence of three words separated by spaces.

Comments may appear following the semicolon until the end of the line. Multi-line comments or comments not ignored by the lexical parser can be written using "ordinary" datatypes like multi-line strings.[4]

Semantics

Blocks containing domain-specific language can be submitted as arguments to specific *evaluator* functions.[6]

do

The most frequently used evaluator is the `do` function. It is used by default to interpret the text input to the interpreter console.

The *do dialect* interpreted by the `do` function, is an expression-oriented sublanguage of the *data exchange dialect*. The main semantic unit of the language is the expression. In contrast to imperative programming languages descending from ALGOL, the *do dialect* has neither keywords, nor statements.

Words are used as case-insensitive variables. Like in all dynamically typed languages, variables don't have an associated type, type is associated with values. The result, i.e. the evaluation of a word is returned, when a word is encountered by the `do` function. The set-word form of a word can be used for assignment. While not having statements, assignment, together with functions with side-effects can be used for imperative programming.[4]

Subblocks of the root block evaluate to themselves. This property is used to handle data blocks, for structured programming by submitting blocks as arguments to control functions like `if`, `either`, `loop`, etc., and for dialecting, when a block is passed to a specific interpreter function.[6]

A specific problem worth noting is that composite values, assigned to variables, are not copied. To make a copy, the value must be passed to the `copy` function.[4]

The `do` function normally follows a prefix style of evaluation, where a function processes the arguments that follow it. However, infix evaluation using infix operators exists too. Infix evaluation takes precedence over the prefix evaluation. For example,

```
abs -2 + 3
```

returns 1, since the infix addition takes precedence over the computation of the absolute value. When evaluating infix expressions, the order of evaluation is left to right, no operator takes precedence over another. For example,

```
2 + 3 * 4
```

returns 20, while an evaluation giving precedence to multiplication would yield 14. All operators have prefix versions. `Do` usually evaluates arguments before passing them to a function. So, the below expression:

```
print read http://en.wikipedia.org/wiki/Rebol
```

first reads the Wikipedia Rebol page and then passes the result to the `print` function. Parentheses can be used to change the order of evaluation. Using prefix notation, the usage of parentheses in expressions can be avoided.[4]

The simple precedence rules are both an advantage:

- No need to "consult" precedence tables when writing expressions
- No need to rewrite precedence tables when a new operator is defined
- Expressions can be easily transliterated from infix to prefix notation and vice versa

as well as a disadvantage:

- Users accustomed to more conventional precedence rules may easily make a mistake[6]

parse

The `parse` function is preferably used to specify, validate, transform and interpret dialects. It does so by matching *parse expressions* at run time.[6]

Parse expressions are written in the *parse dialect*, which, like the *do dialect*, is an expression-oriented sublanguage of the *data exchange dialect*. Unlike the *do dialect*, the *parse dialect* uses keywords representing operators and the most important nonterminals, infix parsing operators don't have prefix equivalents and use precedence rules (*sequence* has higher precedence than *choice*).[6]

Actions can be included to be taken during the parsing process as well and the `parse` function can be used to process blocks or strings. At the *string parsing* level `parse` must handle the "low level" parsing, taking into account characters and delimiters. *Block parsing* is higher level, handling the scanning at the level of Rebol values.[6]

The parse dialect belongs to the family of grammars represented by the top-down parsing language or the parsing expression grammar (PEG). The main similarity is the presence of the *sequence* and *choice* operators all the family members have. Parse dialect syntax and the similarities between the parse dialect and the PEG are illustrated by this transliteration of a PEG example that parses an arithmetic expression:

```
Digit: charset [#"0" - #"9"]
Value: [some Digit | "(" Expr ")"]
Product: [Value any [["*"| "/"] Value]]
```

```
Sum: [Product any [["+"| "-"] Product]]
Expr: Sum
parse/all "12+13" Expr
```

Implementations

The official Rebol 2.7.8 implementation is available in several editions (*/Core*, */View*, */Command*, */SDK* and */IOS*). Both */Core* and */View* editions are freely redistributable software.[1]

The runtime environment is stored in a single executable file. *Rebol/Core* 2.7.8, the console edition, is about 300 KB and *Rebol/View* 2.7.8, the graphical user interface edition, is about 650 KB in size.

Rebol/View provides platform-independent graphics and sound access, and comes with its own windowing toolkit and extensible set of styles (GUI widgets). Extended editions, such as *Rebol/Command* 2.7.8 or *Rebol/SDK* 2.7.8 require a paid license; they add features like ODBC data access, and the option to create standalone executable files.

Legacy

- Rebol was named by Douglas Crockford as one of the inspirations of JavaScript Object Notation.[5]
- Rebol inspired the open-source Orca project, which is an interpreted Rebol-like language.[14]
- Boron (http://urlan.sourceforge.net/boron/) is an interpreted, homoiconic language inspired by and similar to Rebol, which is meant for embedding domain specific languages. It is implemented as a C library licensed under the terms of the LGPLv3.
- The Red programming language was directly inspired by Rebol, yet the implementation choices of Red were geared specifically to overcoming its perceived limitations.[15]

See also

- Domain-specific language
- Language-oriented programming

References

1. REBOL Technologies. *The REBOL/View and REBOL/Core 2.7.8 license (http://www.rebol.com/license.html)*
2. R3 source (https://github.com/rebol/r3) at GitHub
3. "Carl's REBOL Blog - Let's switch to .reb suffix" (http://www.rebol.com/article/0540.html). Rebol.com. August 18, 2013. Retrieved January 23, 2014.
4. Goldman, E., Blanton, J. (2000). *REBOL: The Official Guide.* McGraw-Hill Osborne Media. ISBN 0-07-212279-X.
5. Crockford, Douglas. *The JSON Saga (https://www.youtube.com/watch?v=-C-JoyNuQJs)*, jsonsaga.ppt (http://crockford.com/codecamp/jsonsaga.ppt) Archived (https://web.archive.org/web/20121004043939/http://crockford.com/codecamp/jsonsaga.ppt#) October 4, 2012, at the Wayback Machine
6. Roberts, Ralph (2000). *REBOL for Dummies.* Hungry Minds. ISBN 0-7645-0745-1.
7. Auverlot, Olivier (2001). *Rebol Programmation.* Eyrolles. ISBN 2-212-11017-0.

8. Sassenrath, Carl (July 1, 2000). "Inside the REBOL scripting language" (http://www.ddj.com/184404172). Dr. Dobb's Journal.
9. "REBOL to become open source" (http://www.rebol.com/article/0511.html). Rebol.com. September 25, 2012. Retrieved January 23, 2014.
10. Sassenrath, Carl (December 12, 2012). "Comments on: R3 Source Code Released!" (http://www.rebol.com/cgi-bin/blog.r?view=0519). Retrieved August 14, 2014. "*You probably thought the source release would never happen? Am I right? Well, it's there now in github at github.com/rebol/rebol.*"
11. "Calling REBOL Rebol?" (http://www.rebol.com/cgi-bin/blog.r?view=0525). December 14, 2012. Retrieved December 2, 2013.
12. Sassenrath, Carl. "Cross-compiling Rebol for your favorite embedded board" (http://www.rebol.com/article/0526.html). Retrieved September 16, 2016.
13. "Source code for the Rebol interpreter" (https://github.com/rebol/rebol). *rebol/rebol GitHub*. Retrieved March 14, 2017.
14. The rebol-orca project (http://freecode.com/projects/rebol-orca/) at Freecode
15. The Red project (https://github.com/red/red/blob/master/README.md#red-programming-language) at GitHub

Further reading

- Mikes, Nora (November 20, 1999). "A REBOL incursion: It's not a scripting language, not a programming language -- and not a new Amiga, either. Or is it?" (https://web.archive.org/web/20050324001942/http://linuxworld.com/linuxworld/lw-1999-10/lw-10-rebol.html). *LinuxWorld*. Archived from the original (http://www.linuxworld.com/linuxworld/lw-1999-10/lw-10-rebol.html) on March 24, 2005.
- Story, Derrick (August 27, 1999). "Rebol Might Be the Language for the Rest of Us" (https://web.archive.org/web/19991012182529/http://webreview.com/pub/1999/08/27/feature/index.html). *Web Review*. Archived from the original (http://webreview.com/pub/1999/08/27/feature/index.html) on October 12, 1999.

External links

- REBOL (https://curlie.org/Computers/Programming/Languages/REBOL) at Curlie
- A REBOL tutorial (http://re-bol.com/rebol.html)
- Rebol 3 Tutorial (http://learnrebol.com/rebol3_book.html)
