



Erlang (programming language)

Erlang

	
Paradigm	multi-paradigm: concurrent, functional
Appeared in	1986
Designed by	Ericsson
Developer	Ericsson
Stable release	R14B02 (March 16, 2011)
Typing discipline	dynamic, strong
Major implementations	Erlang
Influenced by	Prolog
Influenced	Clojure, Scala
License	Modified MPL
Website	[1]
 Erlang Programming at Wikibooks	

Erlang is a general-purpose concurrent, garbage-collected programming language and runtime system. The sequential subset of Erlang is a functional language, with strict evaluation, single assignment, and dynamic typing. For concurrency it follows the Actor model. It was designed by Ericsson to support distributed, fault-tolerant, soft-real-time, non-stop applications. It supports hot swapping, thus code can be changed without stopping a system.^[2]

While threads are considered a complicated and error-prone topic in most languages, Erlang provides language-level features for creating and managing processes with the aim of simplifying concurrent programming. Though all concurrency is explicit in Erlang, processes communicate using message passing instead of shared variables, which removes the need for locks.

The first version was developed by Joe Armstrong in 1986.^[3] It was originally a proprietary language within Ericsson, but was released as open source in 1998.

History

The name "Erlang", attributed to Bjarne Däcker, has been understood either as a reference to Danish mathematician and engineer Agner Krarup Erlang, or alternatively, as an abbreviation of "Ericsson Language".^[3]^[4]

Erlang was designed with the aim of improving the development of telephony applications. The initial version of Erlang was implemented in Prolog.^[3]

In 1998, the Ericsson AXD301 switch was announced, containing over a million lines of Erlang, and reported to achieve a reliability of nine "9"s.^[5] Shortly thereafter, Erlang was banned within Ericsson Radio Systems for new products, citing a preference for non-proprietary languages. The ban caused Armstrong and others to leave Ericsson.^[6] The implementation was open sourced at the end of the year.^[3] The ban at Ericsson was eventually lifted, and Armstrong was re-hired by Ericsson in 2004.^[6]

In 2006, native symmetric multiprocessing support was added to the runtime system and virtual machine.^[3]

Philosophy

The philosophy used to develop Erlang fits equally well with the development of Erlang-based systems. Quoting Mike Williams, one of the three inventors of Erlang:

1. Find the right methods—Design by Prototyping.
2. It is not good enough to have ideas, you must also be able to implement them and know they work.
3. Make mistakes on a small scale, not in a production project.

Functional programming examples

A factorial algorithm implemented in Erlang:

```
-module (fact).      % This is the file 'fact.erl', the module and the
filename MUST match
-export ([fac/1]). % This exports the function 'fac' of arity 1 (1
parameter, no type, no name)

fac(0) -> 1; % If 0, then return 1, otherwise (note the semicolon ;
meaning 'else')
fac(N) when N > 0, is_integer(N) -> N * fac(N-1).
% Recursively determine, then return the result
% (note the period . meaning 'endif' or 'function end')
```

A sorting algorithm (similar to quicksort):

```
%% qsort:qsort(List)
%% Sort a list of items
-module (qsort).      % This is the file 'qsort.erl'
-export ([qsort/1]). % A function 'qsort' with 1 parameter is exported
(no type, no name)

qsort([]) -> []; % If the list [] is empty, return an empty list
(nothing to sort)
qsort([Pivot|Rest]) ->
    % Compose recursively a list with 'Front' for all elements that
should be before 'Pivot'
    % then 'Pivot' then 'Back' for all elements that should be after
```

```
'Pivot'
  qsort([Front || Front <- Rest, Front < Pivot])
  ++ [Pivot] ++
  qsort([Back || Back <- Rest, Back >= Pivot]).
```

The above example recursively invokes the function `qsort` until nothing remains to be sorted. The expression `[Front || Front <- Rest, Front < Pivot]` is a list comprehension, meaning “Construct a list of elements `Front` such that `Front` is a member of `Rest`, and `Front` is less than `Pivot`.” `++` is the list concatenation operator.

A comparison function can be used for more complicated structures for the sake of readability.

The following code would sort lists according to length:

```
% This is file 'listsort.erl' (the compiler is made this way)
-module(listsort).
% Export 'by_length' with 1 parameter (don't care of the type and name)
-export([by_length/1]).

by_length(Lists) -> % Use 'qsort/2' and provides an anonymous function
as a parameter
  qsort(Lists, fun(A,B) when is_list(A), is_list(B) -> length(A) < length(B) end).

qsort([], _) -> []; % If list is empty, return an empty list (ignore the
second parameter)
qsort([Pivot|Rest], Smaller) ->
  % Partition list with 'Smaller' elements in front of 'Pivot' and
not-'Smaller' elements
  % after 'Pivot' and sort the sublists.
  qsort([X || X <- Rest, Smaller(X,Pivot)], Smaller)
  ++ [Pivot] ++
  qsort([Y || Y <- Rest, not(Smaller(Y, Pivot))], Smaller).
```

Here again, a `Pivot` is taken from the first parameter given to `qsort()` and the rest of `Lists` is named `Rest`. Note that the expression

```
[X || X <- Rest, Smaller(X,Pivot)]
```

is no different in form from

```
[Front || Front <- Rest, Front < Pivot]
```

(in the previous example) except for the use of a comparison function in the last part, saying “Construct a list of elements `X` such that `X` is a member of `Rest`, and `Smaller` is true”, with `Smaller` being defined earlier as

```
fun(A,B) when is_list(A), is_list(B) -> length(A) < length(B) end
```

Note also that the anonymous function is named `Smaller` in the parameter list of the second definition of `qsort` so that it can be referenced by that name within that function. It is not named in the first definition of `qsort`, which deals with the base case of an empty list and thus has no need of this function, let alone a name for it.

Data structures

Erlang has eight primitive data types:

1. **Integers:** integers are written as sequences of decimal digits, for example, 12, 12375 and -23427 are integers. Integer arithmetic is exact and only limited by available memory on the machine.
2. **Atoms:** atoms are used within a program to denote distinguished values. They are written as strings of consecutive alphanumeric characters, the first character being a small letter. Atoms can obtain any character if they are enclosed within single quotes and an escape convention exists which allows any character to be used within an atom.
3. **Floats:** floating point numbers use the IEEE 754 64-bit representation. (Range: $\pm 10^{308}$.)
4. **References:** references are globally unique symbols whose only property is that they can be compared for equality. They are created by evaluating the Erlang primitive `make_ref()`.
5. **Binaries:** a binary is a sequence of bytes. Binaries provide a space-efficient way of storing binary data. Erlang primitives exist for composing and decomposing binaries and for efficient input/output of binaries.
6. **Pids:** Pid is short for Process Identifier—a Pid is created by the Erlang primitive `spawn(...)` Pids are references to Erlang processes.
7. **Ports:** ports are used to communicate with the external world. Ports are created with the built-in function (BIF) `open_port`. Messages can be sent to and received from ports, but these message must obey the so-called "port protocol."
8. **Funs :** Funs are function closures. Funs are created by expressions of the form: `fun(...) -> ... end`.

And two compound data types:

1. **Tuples :** tuples are containers for a fixed number of Erlang data types. The syntax `{D1,D2,...,Dn}` denotes a tuple whose arguments are D1, D2, ... Dn. The arguments can be primitive data types or compound data types. The elements of a tuple can be accessed in constant time.
2. **Lists :** lists are containers for a variable number of Erlang data types. The syntax `[Dh[D2]..[Dn][[]]`. The first element of a list can be accessed in constant time. The first element of a list is called the *head* of the list. The remainder of a list when its head has been removed is called the *tail* of the list.

Two forms of syntactic sugar are provided:

1. **Strings :** strings are written as doubly quoted lists of characters, this is syntactic sugar for a list of the integer ASCII codes for the characters in the string, thus for example, the string "cat" is shorthand for `[99,97,116]`.
2. **Records :** records provide a convenient way for associating a tag with each of the elements in a tuple. This allows us to refer to an element of a tuple by name and not by position. A pre-compiler takes the record definition and replaces it with the appropriate tuple reference.

Concurrency and distribution orientation

Erlang's main strength is support for concurrency. It has a small but powerful set of primitives to create processes and communicate among them. Processes are the primary means to structure an Erlang application. Erlang processes loosely follow the communicating sequential processes (CSP) model. They are neither operating system processes nor operating system threads, but lightweight processes somewhat similar to Java's original "green threads". Like operating system processes (and unlike green threads and operating system threads) they have no shared state between them. The estimated minimal overhead for each is 300 words,^[7] thus many of them can be created without degrading performance: a benchmark with 20 million processes has been successfully performed.^[8] Erlang has supported symmetric multiprocessing since release R11B of May 2006.

Inter-process communication works via a shared-nothing asynchronous message passing system: every process has a "mailbox", a queue of messages that have been sent by other processes and not yet consumed. A process uses the receive primitive to retrieve messages that match desired patterns. A message-handling routine tests messages in turn

against each pattern, until one of them matches. When the message is consumed and removed from the mailbox the process resumes execution. A message may comprise any Erlang structure, including primitives (integers, floats, characters, atoms), tuples, lists, and functions.

The code example below shows the built-in support for distributed processes:

```
% Create a process and invoke the function web:start_server(Port,
MaxConnections)
ServerProcess = spawn(web, start_server, [Port, MaxConnections]),

% Create a remote process and invoke the function
% web:start_server(Port, MaxConnections) on machine RemoteNode
RemoteProcess = spawn(RemoteNode, web, start_server, [Port,
MaxConnections]),

% Send a message to ServerProcess (asynchronously). The message
consists of a tuple
% with the atom "pause" and the number "10".
ServerProcess ! {pause, 10},

% Receive messages sent to this process
receive
    a_message -> do_something;
    {data, DataContent} -> handle(DataContent);
    {hello, Text} -> io:format("Got hello message: ~s", [Text]);
    {goodbye, Text} -> io:format("Got goodbye message: ~s", [Text])
end.
```

As the example shows, processes may be created on remote nodes, and communication with them is transparent in the sense that communication with remote processes works exactly as communication with local processes.

Concurrency supports the primary method of error-handling in Erlang. When a process crashes, it neatly exits and sends a message to the controlling process which can take action.^[9] ^[10] This way of error handling increases maintainability and reduces complexity of code.

Implementation

The Ericsson Erlang implementation loads virtual machine bytecode which is converted to threaded code at load time. It also includes a native code compiler on most platforms, developed by the High Performance Erlang Project (HiPE) at Uppsala University. Since October 2001 the HiPE system is fully integrated in Ericsson's Open Source Erlang/OTP system.^[11] It also supports interpreting, directly from source code via abstract syntax tree, via script as of R11B-5.

Hot code loading and modules

Code is loaded and managed as "module" units; the module is a compilation unit. The system can keep two versions of a module in memory at the same time, and processes can concurrently run code from each. The versions are referred to as the "new" and the "old" version. A process will not move into the new version until it makes an external call to its module.

An example of the mechanism of hot code loading:

```

%% A process whose only job is to keep a counter.
%% First version
-module(counter).
-export([start/0, codeswitch/1]).

start() -> loop(0).

loop(Sum) ->
    receive
        {increment, Count} ->
            loop(Sum+Count);
        {counter, Pid} ->
            Pid ! {counter, Sum},
            loop(Sum);
        code_switch ->
            ?MODULE:codeswitch(Sum)
            % Force the use of 'codeswitch/1' from the latest MODULE
version
    end.

codeswitch(Sum) -> loop(Sum).

```

For the second version, we add the possibility to reset the count to zero.

```

%% Second version
-module(counter).
-export([start/0, codeswitch/1]).

start() -> loop(0).

loop(Sum) ->
    receive
        {increment, Count} ->
            loop(Sum+Count);
        reset ->
            loop(0);
        {counter, Pid} ->
            Pid ! {counter, Sum},
            loop(Sum);
        code_switch ->
            ?MODULE:codeswitch(Sum)
    end.

codeswitch(Sum) -> loop(Sum).

```

Only when receiving a message consisting of the atom 'code_switch' will the loop execute an external call to codeswitch/1 (?MODULE is a preprocessor macro for the current module). If there is a new version of the "counter" module in memory, then its codeswitch/1 function will be called. The practice of having a specific entry-point into a new version allows the programmer to transform state to what is required in the newer version. In our example we

keep the state as an integer.

In practice, systems are built up using design principles from the Open Telecom Platform which leads to more code upgradable designs. Successful hot code loading is a tricky subject; code needs to be written to make use of Erlang's facilities.

Distribution

In 1998, Ericsson released Erlang as open source to ensure its independence from a single vendor and to increase awareness of the language. Erlang, together with libraries and the real-time distributed database Mnesia, forms the Open Telecom Platform (OTP) collection of libraries. Ericsson and a few other companies offer commercial support for Erlang.

Since the open source release, Erlang has been used by several firms worldwide, including Nortel and T-Mobile.^[12] Although Erlang was designed to fill a niche and has remained an obscure language for most of its existence, its popularity is growing due to demand for concurrent services.^{[13] [14]} Erlang has found some use in fielding MMORPG servers.^[15]

Erlang is available for many Unix-like operating systems, including Mac OS X, as well as Microsoft Windows.

Projects using Erlang

Projects using Erlang include:

- CouchDB, a document based database that uses MapReduce
- ejabberd, an Extensible Messaging and Presence Protocol (XMPP) instant messaging server
 - Facebook Chat system,^[16] based on ejabberd^[17]
- GitHub egitd,^[18] a replacement for stock git-daemon that ships with Git
- Issuu, an online digital publisher
- Membase, database management system optimized for storing data behind interactive web applications.
- RabbitMQ, an implementation of Advanced Message Queuing Protocol (AMQP)
- SimpleDB, a distributed database that is part of Amazon Web Services^[19]
- Twitterfall, a service to view trends and patterns from Twitter^{[20] [21]}
- Wings 3D, a 3D modeller
- Yaws web server
- Riak, a distributed database
- Goldman Sachs used Erlang for the high-frequency trading programs. Sergey Aleynikov, a former programmer at Goldman Sachs was accused of stealing their code.

Clones

Erlang has inspired clones of its concurrency facilities for other languages:

- Reia
- Scala

References

- [1] <http://www.erlang.org>
- [2] Joe Armstrong, Bjarne Däcker, Thomas Lindgren, Håkan Millroth. "Open-source Erlang - White Paper" (http://erlang.org/white_paper.html). . Retrieved 2008-01-23.
- [3] Joe Armstrong, "History of Erlang", in *HOPL III: Proceedings of the third ACM SIGPLAN conference on History of programming languages*, 2007, ISBN 978-1-59593-766-X
- [4] Erlang, the mathematician? (<http://www.erlang.org/pipermail/erlang-questions/1999-February/000098.html>)
- [5] "Concurrency Oriented Programming in Erlang" (<http://l12.ai.mit.edu/talks/armstrong.pdf>). November 2, 2002. .

- [6] "question about Erlang's future" (<http://erlang.org/pipermail/erlang-questions/2006-July/021336.html>). July 6, 2010. .
- [7] "Erlang Efficiency Guide - Processes" (http://www.erlang.org/doc/efficiency_guide/processes.html). .
- [8] Ulf Wiger (2005-11-14). "Stress-testing erlang" (<http://groups.google.com/group/comp.lang.functional/msg/33b7a62afb727a4f?dmode=source>). *comp.lang.functional.misc*. . Retrieved 2006-08-25.
- [9] Joe Armstrong. "Erlang robustness" (http://www.erlang.org/doc/getting_started/robustness.html). . Retrieved 2010-07-15.
- [10] "Erlang Supervision principles" (http://www.erlang.org/doc/design_principles/sup_princ.html). . Retrieved 2010-07-15.
- [11] "High Performance Erlang" (<http://www.it.uu.se/research/group/hipe/>). . Retrieved 2011-03-26.
- [12] "Who uses Erlang for product development?" (<http://www.erlang.org/faq/faq.html#AEN50>). *Frequently asked questions about Erlang*. . Retrieved 2007-07-16. "The largest user of Erlang is (surprise!) Ericsson. Ericsson use it to write software used in telecommunications systems. Many dozens projects have used it, a particularly large one is the extremely scalable AXD301 ATM switch. Other commercial users listed as part of the FAQ include: Nortel, Deutsche Flugsicherung (the German national air traffic control organisation), and T-Mobile."
- [13] "Programming Erlang" (http://www.ddj.com/linux-open-source/201001928?cid=RSSfeed_DDJ_OpenSource). . Retrieved 2008-12-13. "Virtually all language use shared state concurrency. This is very difficult and leads to terrible problems when you handle failure and scale up the system...Some pretty fast-moving startups in the financial world have latched onto Erlang; for example, the Swedish www.kreditor.se."
- [14] "Erlang, the next Java" (<http://www.cincomsmalltalk.com/userblogs/ralph/blogView?showComments=true&entry=3364027251>). . Retrieved 2008-10-08. "I do not believe that other languages can catch up with Erlang anytime soon. It will be easy for them to add language features to be like Erlang. It will take a long time for them to build such a high-quality VM and the mature libraries for concurrency and reliability. So, Erlang is poised for success. If you want to build a multicore application in the next few years, you should look at Erlang."
- [15] Clarke, Gavin (5 Feb 2011). "Battlestar Galactica vets needed for online roleplay" (http://www.theregister.co.uk/2011/02/05/battlestar_galactica_mmp/) (HTML). *Music and Media*. The Reg. . Retrieved 2011-02-08.
- [16] http://www.facebook.com/note.php?note_id=16787213919&id=9445547199&index=2
- [17] <http://developers.facebook.com/news.php?blog=1&story=110>
- [18] <http://github.com/blog/112-supercharged-git-daemon>
- [19] What You Need To Know About Amazon SimpleDB (<http://www.satine.org/archives/2007/12/13/amazon-simpledb/>)
- [20] <http://twitter.com/jalada/status/1206606823>
- [21] <http://twitter.com/jalada/statuses/1234217518>

Further reading

- Joe Armstrong (2003). *Making reliable distributed systems in the presence of software errors* (http://www.sics.se/~joe/thesis/armstrong_thesis_2003.pdf). Ph.D. Dissertation. The Royal Institute of Technology, Stockholm, Sweden.
- Armstrong, J. (2007). *A history of Erlang*. pp. 6–1. doi:10.1145/1238844.1238850.
- Early history of Erlang (<http://www.erlang.se/publications/bjarnelic.pdf>) by Bjarne Däcker
- "Mnesia - A distributed robust DBMS for telecommunications applications". *First International Workshop on Practical Aspects of Declarative Languages (PADL '99)*: 152–163. 1999.
- Armstrong, Joe; Viriding, Robert; Williams, Mike; Wikstrom, Claes (January 16, 1996). *Concurrent Programming in Erlang* (http://www.erlang.org/erlang_book_toc.html) (2nd ed.). Prentice Hall. pp. 358. ISBN 9780135083017.
- Armstrong, Joe (July 11, 2007). *Programming Erlang: Software for a Concurrent World* (<http://pragprog.com/titles/jaerlang/programming-erlang>) (1st ed.). Pragmatic Bookshelf. pp. 536. ISBN 9781934356005.
- Thompson, Simon J.; Cesarini, Francesco (June 19, 2009). *Erlang Programming: A Concurrent Approach to Software Development* (<http://www.erlangprogramming.org>) (1st ed.). Sebastopol, California: O'Reilly Media, Inc. pp. 496. ISBN 978059651818.
- Cant, Geoff (March 1, 2010). *Mastering Erlang: Writing Real World Applications* (<http://www.apress.com/book/view/9781430227694>) (1st ed.). Apress. pp. 350. ISBN 9781430227694.
- Logan, Martin; Merritt, Eric; Carlsson, Richard (May 28, 2010). *Erlang and OTP in Action* (<http://www.manning.com/logan>) (1st ed.). Greenwich, CT: Manning Publications. pp. 500. ISBN 9781933988788.
- Gerakines, Nick (August 2, 2010). *Erlang Web Applications: Problem - Design - Solutions* (<http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470743840.html>) (1st ed.). John Wiley & Sons. pp. 512. ISBN 9780470743843.

External links

- Official website (<http://http://www.erlang.org>)
 - Erlang (<http://www.dmoz.org/Computers/Programming/Languages/Erlang/>) at the Open Directory Project
 - trapexit.org (<http://trapexit.org/>), site with much Erlang/OTP information
 - Erlang: The Movie (<http://www.archive.org/details/ErlangTheMovie>)
 - Erlang for Skeptics (<http://www.erlangforsceptics.com/>), book for beginners, work in progress
 - Learn You Some Erlang (<http://www.learnyouosomeerlang.com/>), tutorial for beginners
 - erldocs.com (<http://erldocs.com/>), alternative topic documentation
 - Joe Armstrong on Erlang (<http://www.se-radio.net/2008/03/episode-89-joe-armstrong-on-erlang>), Software Engineering Radio Podcast
-

Article Sources and Contributors

Erlang (programming language) *Source:* <http://en.wikipedia.org/w/index.php?oldid=421016424> *Contributors:* :Ajvol:, 1234456556BK, 209.75.42.xxx, 4th-otaku, Adrianwn, Akavel, Allan McInnes, Amire80, AmritTuladhar, Andreas.hillqvist, Andyjsmith, AntiRush, Antonielli, Argv0, Arn7, Ashley Y, AutumnSnow, Barkjon, BenBaker, Bender235, Bosmon, Callidior, Camw, Carnildo, ChasRMartin, Conversion script, Cromain, Csl77, Cybercobra, Danakil, Darklich14, David Gerard, David Wahler, DavidDouthitt, Decltype, Defza, Demi, DennyAbraham, Dlambert, Dmitriid, Doradus, Drbreznjev, Drewnoakes, Dysprosia, Ebraminio, Ejabberd, Eliassen, Emurphy42, Erlang 1, Erlang.consulting, Etz Haim, Faisal.akeel, Fanf, FatalError, Fikusfail, Frecklefoot, Fudoreaper, Furrykef, Gaius Cornelius, Gf uip, Giardante, Gleber.p, Gpierre, Graham87, Greenrd, GregorB, Grimboy, Grshiplett, Gwern, H2g2bob, Halfacanuck, Hcs42, Hdante, Hsk0114, Ignatz, Intr, Iridescent, JLD, JLaTondre, James Hague, JamesBWatson, Jameshague, Janko.stefancic, Jasontrost, Jeltz, Jerryobject, JohnnyBjorn, Jonasfagundes, JonathanFeinberg, Jonnabuz, Justin W Smith, Justinsheehy, Karvendhan, Kevingc, Korpo, Krallja, Kricxjo, Kuszi, Levin, Lihaitao, Liujiang, Loopkid, Lproven, Luke.randall, M7, Mapfn, Mariehuynh, Markpeak, Martinl, Marudubshinki, Masharabinovich, Matt Crypto, Matthewdunsdon, Matthewokeefe, Maxim.kharchenko, Maxlapshin, Maxsharples, Memming, Miketomasello, Mistercupcake, Moltone13xCombo, Msbmsb, Mwarren us, NaturalBornKiller, Neustradamus, Niten, Noliver, Nyco, Ohnoitsjamie, Palfrey, Papppaffe, Pauli133, Pavel Vozenilek, Peepeedia, Peet74, Peter S., Pgan002, Piet Delpport, Potto007, Prof3ta, Qianguuol, Quadell, Ravidgemole, Renku, Richardhenwood, RickBeton, Riffic, Rklophaus, Rogper, Rohan Jayasekera, Ronewolf, Ronz, Ruud Koot, Ryzol, Sandos, Sanspeur, Sbierwagen, ScierGuy, Sedrik666, Stevenj, Stewartadcock, Stpeter, Svick, Tarka, Taw, The Anome, Tim Ivorson, Tobias Bergemann, Tobias382, TobinFricke, ToddDeLuca, Tonieisner, Traroth, Twn, Ultramandk, Uninverted, Untalker, Vectro, Vikreykja, VladimirKorablin, Vocaro, Vroo, Yakushima, Yaronf, Zetawoof, Zvar, 342 anonymous edits

Image Sources, Licenses and Contributors

File:Erlang logo.png *Source:* http://en.wikipedia.org/w/index.php?title=File:Erlang_logo.png *License:* Trademarked *Contributors:* Kyro, WikipediaMaster

File:Wikibooks-logo-en.svg *Source:* <http://en.wikipedia.org/w/index.php?title=File:Wikibooks-logo-en.svg> *License:* logo *Contributors:* User:Bastique, User:Ramac

License

Creative Commons Attribution-Share Alike 3.0 Unported
<http://creativecommons.org/licenses/by-sa/3.0/>