

# Erlang/OTP Mini Introduction

Tedi Heriyanto\*

August 17, 2005. Last update : November 10, 2005

## 1 Introduction to Erlang[2]

Erlang is a general-purpose programming language and runtime environment. Erlang has built-in support for concurrency, distribution and fault tolerance. OTP (Open Telephony Platform) is a large collection of libraries for Erlang to do everything from compiling ASN.1 to providing a WWW server.

Erlang is suitable for the following applications :

- Telecommunication systems : control a switch.
- Servers for Internet applications : HTTP server, mail transfer agent.
- Telecommunication agents : handling mobility in a mobile network.
- Database applications which require soft real time behavior.

Erlang was developed by Ericsson Computer Science Laboratory in 1980s. The people involved at the beginning were Joe Armstrong, Robert Virding, and Mike Williams.

## 2 Erlang/OTP Installation

### 2.1 Prerequisites

Below are some tools that need to be installed on your system before you can build Erlang :

- GNU make
- GNU C compiler

---

\*This document was written to celebrate Indonesia's 60th Anniversary on August 17, 2005. Last updated on Indonesia's Heroic Day. I dedicated this document to my beloved country, Indonesia. I am currently working in IT security field. I maintain my personal website at <http://tedi.heriyanto.net>. You can contact me at [tedi\\_heriyanto@yahoo.com](mailto:tedi_heriyanto@yahoo.com)

- Perl 5

The followings are some optional softwares that need to be installed :

- OpenSSL : needed for building the Erlang/OTP applications 'ssl' and 'crypto'. At least version 0.9.7 of OpenSSL is required.
- Sun Java jdk-1.2.2 or higher : needed for building the Erlang/OTP application 'jinterface' and parts of 'ic' and 'orber'. I use Sun JDK 1.5.0
- X Windows : needed to build the Erlang/OTP application 'gs' on Unix/Linux.
- Flex : needed to build the flex scanner for the megaco application on Unix/Linux

## 2.2 Installation Process

Here are the steps I took to install Erlang on my Linux system :

- Download Erlang source code from <http://www.erlang.org/download.html>

The latest Erlang version when I write this article is R10B-8. The source code is named `otp_src_R10B-8.tar.gz` (its size is around 10MB).

- After that extract the source code tarball :

```
$ tar xvzfp otp_src_R10B-8.tar.gz
...
otp_src_R10B-8/lib/xmerl/src/xmerl_xpath.erl
otp_src_R10B-8/lib/xmerl/src/xmerl_xpath_parse.yrl
otp_src_R10B-8/lib/xmerl/src/xmerl_xpath_pred.erl
otp_src_R10B-8/lib/xmerl/src/xmerl_xpath_scan.erl
otp_src_R10B-8/lib/xmerl/src/xmerl_xs.erl
otp_src_R10B-8/lib/xmerl/vsn.mk
otp_src_R10B-8/lib/xmerl/xmerl.pub
```

- Change to Erlang directory :

```
cd otp_src_R10B-8/
```

- Configure Erlang build :

```
./configure
checking host system type... i686-pc-linux-gnu
checking for GNU make... yes (make)
checking for a BSD compatible install... /usr/bin/install -c
checking whether ln -s works... yes
checking for ranlib... ranlib
creating ./config.status
```

```

creating Makefile
...
creating i686-pc-linux-gnu/config.h
creating include/internal/i686-pc-linux-gnu/ethread_header_config.h
*****
***** APPLICATIONS DISABLED *****
*****
jinterface      : No Java compiler found
odbc             : No odbc library found
*****

```

If you want to try the jinterface, make sure you've already installed J2SDK in your system. Then set the environment variable JAVA\_HOME to your Java SDK directory. In my system it is located in /usr/java/jdk1.5.0. And don't forget to add that entries to your .bash\_profile file (if you use Bash Shell). Here are two entries I add to my .bash\_profile file :

```

...
JAVA_HOME=/usr/java/jdk1.5.0
export JAVA_HOME
...

```

I made the shell read its environment variable again :

```
$ . .bash_profile
```

After that I run the configure script again :

```

./configure
...
include/internal/i686-pc-linux-gnu/ethread_header_config.h is unchanged
*****
***** APPLICATIONS DISABLED *****
*****
odbc : No odbc library found
*****

```

- Build Erlang (emulator, libraries) :

```

$ make
...
make[2]: Leaving directory '/home/tedi/software/erlang/otp_src_R10B-6/erts/start_scripts'
make[1]: Leaving directory '/home/tedi/software/erlang/otp_src_R10B-6/erts

```

The compilation process will take some time, it depends on the speed of your computer. In my system it took about 15 minutes 30 seconds. :(

After the compilation success, we will have a working Erlang/OTP system. We can check it by starting Erlang shell :



Figure 1: Erlang Toolbar

```
$ bin/erl
Erlang (BEAM) emulator version 5.4.10 [source] [hipe]
Eshell V5.4.10 (abort with ^G)
1>
```

Start the GS-based toolbar from the Erlang shell :

```
1> toolbar:start().
```

It then pop up a toolbar window :

To exit, enter the command `halt()` or `init:stop()` :

```
2> halt().
```

- (Optional) Next we can install Erlang/OTP to our system :

```
$ su
password :
# make install
...
ln -s /usr/local/lib/erlang/bin/erl      /usr/local/bin/erl
ln -s /usr/local/lib/erlang/bin/erlc    /usr/local/bin/erlc
ln -s /usr/local/lib/erlang/bin/ecc     /usr/local/bin/ecc
ln -s /usr/local/lib/erlang/bin/erlink  /usr/local/bin/erlink
ln -s /usr/local/lib/erlang/bin/ear     /usr/local/bin/ear
ln -s /usr/local/lib/erlang/bin/escript /usr/local/bin/escript
```

This will install Erlang/OTP to `/usr/local` directory or whatever directory you chose during the configure step.

### 3 A Taste of Erlang

It's just like a norm in computer language tutorial that the first program we wrote usually about displaying a "Hello World" sentence. So here is the program to display "Hello World" in Erlang/OTP :

```
-module(hello).
-export([hello_world/0]).
hello_world() -> io:fwrite("Hello World\n").
```

### 3.1 Compile and Run in Erlang Shell

To compile the program above, save it to the file named “`hello.erl`” and compile it in Erlang shell :

```
$ erl
Erlang (BEAM) emulator version 5.4.10 [source] [hipe]
Eshell V5.4.10 (abort with ^G)
1> c(hello).
{ok,hello}
2>
```

In the output above, we compile the file using command “`c`” and we give the filename as the parameter. The compilation process result then displayed by the shell `{ok,hello}`. After the compilation succeed, we can run the program :

```
2> hello:hello_world().
Hello World
ok
3>
```

To run the program we call the program module and the function we want. If succeed, the shell will displayed “`ok`” and give us a shell prompt.

### 3.2 Compile and Run Without Using Erlang Shell

Sometimes we don’t want to start the shell to compile and run Erlang program. Here is how to do the compilation and run without using Erlang shell :

```
$ erl -compile hello
$ erl -noshell -s hello hello_world -s init stop
Hello World
```

In `otp_src_R10B-8`, we can use `erlc` to compile Erlang program :

```
$ erlc hello.erl
```

In the next section, we will discuss the program above.

### 3.3 Hello World Program

Every program in Erlang composed of modules. In this tutorial we only use one module. The module must be saved in a file named `module.erl`. So if we create a module name “`hello`”, we must save that module in file “`hello.erl`”.

I copied the file “`hello.erl`” to file “`hello1.erl`”, then I tried to compile that in Erlang shell :

```
$ erl
Erlang (BEAM) emulator version 5.4.10 [source] [hipe]
Eshell V5.4.10 (abort with ^G)
1> c(hello1).
** Module name 'hello' does not match file name 'hello1' **
{error,badfile}
2>
```

From the output above, we know what the error is and how to fix that. :D  
Here is the program above with numbered lines :

```
1 -module(hello).
2 -export([hello_world/0]).
3
4 hello_world() -> io:fwrite("Hello World\n").
```

The first line, define the module name.

Functions in Erlang cannot be accessed from outside of the module, in order to make them callable from the outside, we need to export them. In the second line, we export `hello_world()` function. The “0” after the “/” symbol denotes that this function doesn’t need parameters.

The fourth line, define the `hello_world()` function. The `io:fwrite()` function does two things : it prints out “Hello World” and it returns the value “ok”.

## 4 Closing

In this article, I introduced you to Erlang, a general purpose language for distributed, concurrent, large soft realtime applications. This language has many interesting features, such as you can change application code without restarting the application.

Because I am very new to Erlang, I may only be able to touch the surface of Erlang language. But I hope this article will be useful in your journey to learn Erlang.

Until then....see you in other Erlang articles from me.

## References

- [1] Erlang/OTP R10B Documentation, 2005 Ericsson AB.
- [2] Erlang Frequently Asked Questions, [www.erlang.org/faq/](http://www.erlang.org/faq/), 2005.