Erlang for OpenVMS on Integrity Servers

Brett Cameron (brett.cameron@hp.com), John Apps (john.apps@hp.com) December 2011

Disclaimer: the information in this document is the sole responsibility of the authors. The information contained herein is offered on a best-effort basis. Hewlett-Packard offers no support or warranty on any of the content in this document or the software described in it.

1. Introduction

Thank your for your interest in this port of Erlang to OpenVMS on HP Integrity Servers. The current release of Erlang for OpenVMS on Integrity is based on the Erlang R13A distribution.

Erlang (see http://www.erlang.org) is a general-purpose programming language and runtime environment that has built-in support for concurrency, distribution, and fault tolerance. The language was designed by Ericsson to support distributed, fault-tolerant, soft-real-time, non-stop applications, and it is used by Ericsson in several large telecommunications systems. The first version of Erlang was developed by Joe Armstrong in 1986. He has continued to evolve and develop the language. Erlang was originally a proprietary language within Ericsson, but it was released as Open Source in 1998, and the popularity of the language has increased significantly in recent years due to its ability to address problems that cannot be readily or satisfactorily addressed by other languages.

In addition to providing support for concurrency, distribution, and fault tolerance, Erlang supports hot swapping, allowing code to be changed without stopping a system. And while threads are considered a complicated and error-prone topic in most languages, Erlang provides language-level features for creating and managing processes (within the Erlang interpreter) with the aim of simplifying concurrent programming. Erlang implements a “shared-nothing” policy in which cooperating Erlang processes (as distinct from operating system processes) communicate using message passing instead of shared variables, which removes the need for any form of locking.

The capabilities of the core Erlang language are extended by OTP (Open Telecom Platform), which is a large and powerful collection of libraries for Erlang to do everything from compiling ASN.1 to providing a web server; and most Erlang applications make extensive use of facilities provided by OTP. Like Erlang, OTP is also Open Source.

Erlang is ideally suited to the implementation of distributed, highly-reliable, soft real-time concurrent systems. For example, Erlang provides a simple and powerful model for error containment and fault tolerance, and concurrency and message passing are a fundamental to the language with context switching between Erlang processes typically being several orders of magnitude cheaper than switching between threads in a C program. Writing applications made of components that execute on different machines is straightforward, and the Erlang runtime environment (a virtual machine similar to the Java virtual machine) means that code compiled on one platform can typically be run on any other platform that supports the Erlang virtual machine.

The OpenVMS on Integrity port of Erlang includes the core Erlang language distribution along with much of OTP.

While the port has been used to successfully run several large and complex applications on OpenVMS such as the Mnesia database (http://www.erlang.org/doc/apps/mnesia/index.html), the Yaws web server (http://yaws.hyber.org/) and the RabbitMQ AMQP implementation (http://www.rabbitmq.com), it must be emphasised that the port is still very much a “work in progress” and problems are likely to be found. The authors would appreciate hearing about any such problems and will endeavour to address them.

2. What’s new in this release?

This release includes support for HP SSL V1.4 (superseding support for HP SSL V1.3). The distribution now includes a shareable image that may be installed to reduce the memory overhead on systems that run multiple Erlang processes. The shareable image also simplifies linking of external drivers.

3. Requirements

The kit you are receiving has been compiled and built using the operating system and compiler versions listed below. While it is highly likely that you will have no problems installing and using the kit on systems running higher product versions of the products listed, we cannot say for sure that you will be so lucky if your system is running older versions. If you require a kit for a different configuration, we will do what we can to oblige. Note that the UnZip utility is required in order to unpack the ZIP kit.

- OpenVMS 8.3 or higher (IA64 only)
- HP TCP/IP Services V5.6 or higher (if you wish to perform network operations)

  It has not been verified whether the kit works with the MultiNet TCP/IP stack, but there is a good chance that it will.
- HP SSL V1.4 for OpenVMS is required if you intend to use any Erlang libraries that require SSL. Note that HP SSL V1.3 for OpenVMS is not supported by this release.
- UnZip 5.42 (or similar) for OpenVMS (required to unpack the ZIP kit and can be found on the OpenVMS Freeware CD)
- An ODS5-formatted disk with approximately 400,000 blocks of free disk space

  The kit must be installed on an ODS5-formatted disk, and applications must generally be developed on and run from an ODS5-formatted disk.

In addition to the above requirements, it is assumed that the reader has a good knowledge of OpenVMS and of software development in the OpenVMS environment.

4. Recommended reading

If you are new to Erlang, there are a number of good tutorials available to be found on the Internet (see for example http://www.erlang.org/download/getting_started-5.4.pdf), and there are several good books available on Erlang, including this excellent book written by Joe Armstrong: http://www.pragprog.com/titles/jaerlang/programming-erlang.

5. Installing and setup

The kit is currently provided as a ZIP file (created using Zip 2.3 for OpenVMS), which allows individual users to install the software under their own accounts, if so desired.

Unpacking and installing the contents of the ZIP file kit is very straightforward. After copying the supplied ZIP file (`ERLANG-R13A-VMS-I64-05.ZIP`) to a suitable location, unpack the contents of the file using the `unzip` command (see below). This will create the OpenVMS backup save set file[1] `ERLANG-R13A-VMS-I64-05.BCK` in the current directory, which may in turn be restored as illustrated below.

```
$ unzip ERLANG-R13A-VMS-I64-05.ZIP
$ backup/log ERLANG-R13A-VMS-I64-05.BCK/save [...]
```

[1] A backup save set is used to ensure that all files contained in the distribution are restored with the correct attributes.

After unpacking the save set as shown above you will find a `[.erl]` directory below your current directory that contains all the files restored by the `backup` command. Note that you must install the software onto an ODS5-formatted disk.

To complete the installation it is necessary to define the concealed logical name `erlang$root` to point to the top-level `[.erl]` directory and to modify the Erlang portmapper start-up script to specify the desired run-time options.

The following notes describe the necessary post-installation steps:

1. Set default to the `[.erl.bin]` directory and edit the Erlang start-up command procedure `erlang$startup.com` (see Figure 1). Modify the definition of `erlang$root` as required to point to the top level `[.erl]` directory and save your changes. There is no requirement for the `erlang$root` logical name to be defined on a system-wide basis.

   Note that an additional logical name, `erlang$top`, is defined that points to `erlang$root:[000000]`.

   Figure 1 `ERLANG$STARTUP.COM`

```
$ ! ERLANG$STARTUP.COM
$ !+
$ ! 18-Apr-2011
$ ! Startup file for Erlang 13A on OpenVMS Integrity
$ !-
$
$ set noverify
$ set noon
$
$ ! Define system logicals (change definition of erlang$root as appropriate)
$ !
$ define/sys/exec/tran=(conc,term) erlang$root $11$dka800:[user.biggles.erl.]
$ define/sys/exec erlang$top erlang$root:[000000]
$ define/sys/exec erlang$shr erlang$root:[bin]erlang$shr.exe
$
$ ! Install shareable image
$ !
$ install replace erlang$shr/shared/header/open/nowriteable
$
$ ! Start the port mapper daemon...
$ !
$ @erlang$root:[bin]epmd$startup.com
$
$ write sys$output "%ERLANG-I-DONE, Startup complete"
$
$ ! Users may wish to include the following commands in their login.com:
$ !
$ ! erl     :== $erlang$root:[bin]erlexec.exe
$ ! erlc    :== $erlang$root:[bin]erlc.exe
$ ! escript :== $erlang$root:[bin]escript.exe
$
$
$ exit
```

2. Optionally modify `erlang$root:[bin]epmd$run.com`. This command procedure is executed by the Erlang portmapper start-up procedure `epmd$startup.com`, which in turn is run by the main Erlang start-up procedure, `erlang$startup.com`. You may also wish to tailor the command line options for portmapper daemon.

   The portmapper daemon command line options are illustrated in Figure 2. This output may be obtained at any time by defining a foreign command for the portmapper (`erlang$root:[bin]epmd.exe`) and invoking this program with the "`-h`" option. By default no command line options are specified.

Figure 2 EPMD Portmapper daemon command line options

```
$ epmd -h
usage: epmd [-d|-debug] [DbgExtra...] [-port No] [-daemon]
            [-d|-debug] [-port No] [-names|-kill]

See the Erlang epmd manual page for info about the usage.
The -port and DbgExtra options are

    -port No
        Let epmd listen to another port than default 4369
    -d
    -debug
        Enable debugging. This will give a log to
        the standard error stream. It will shorten
        the number of saved used node names to 5.

        If you give more than one debug flag you may
        get more debugging information.

    -packet_timout Seconds
        Set the number of seconds a connection can be
        inactive before epmd times out and closes the
        connection (default 60).

    -delay_accept Seconds
        To simulate a busy server you can insert a
        delay between epmd gets notified about that
        a new connection is requested and when the
        connections gets accepted.

    -delay_write Seconds
        Also a simulation of a busy server. Inserts
        a delay before a reply is sent.
```

3. Set default to the `erlang$root:[bin]` directory and run the command procedure `erlang$startup.com` to define the necessary logical names and to start the portmapper.

Optionally include the command procedures `erlang$root:[bin]erlang$startup.com` and `erlang$root:[bin]erlang$shutdown.com` in the OpenVMS system start-up and shutdown procedures, respectively.

Generally speaking there are no special quota or privilege requirements required for users wishing to develop applications using Erlang, although it should be noted that some extensions may have special requirements (for example, networking extensions may require a high `BYTLM` quota). If you are performing a system-wide installation, be sure to ensure that users can read all of the files in the distribution and can run the executables that reside in the `erlang$root:[bin]` directory.

6. Getting started

To get started, you will want to define foreign commands for the Erlang interpreter (virtual machine) and the Erlang compiler as follows:

```
$ erl      :== $erlang$root:[bin]erlexec.exe
$ erlc     :== $erlang$root:[bin]erlc.exe
```

You should now be able to run the interpreter and start developing Erlang programs! For example, print the date and time:

```
$ erl
Eshell V5.7  (abort with ^Z)
1>  Current = calendar:local_time().
{{2010,6,30},{21,20,54}}
```

Appendix

Tools

Over time the authors will update this document with tools they have found useful in the course of their work. The authors welcome any suggestions and will include them in future versions of the document.

Zip and UnZip for OpenVMS may be obtained from the OpenVMS freeware CD at http://h71000.www7.hp.com/openvms/freeware/.