א Programming Language

Volume 0 - Installation guide

Revision 0.8.1

Contents

Preface

This manual is part of the *Aleph Programming Language Series*, a multi volume set, describing the programming environment of the Aleph system.

- *Volume 0 - Aleph Installation Guide* is the distribution installation manual.
- *Volume 1 - Aleph Programmer Guide* is the first volume of this set. It is both an introduction and an advanced guide for the the developer.
- *Volume 2 - Aleph Library Reference* is the second volume of this set. It is a complete description of the Aleph standard library.

The Aleph programming language

Aleph is a multi-threaded functional programming language with dynamic symbol bindings that support the object oriented paradigm. **Aleph** features a state of the art runtime engine that supports both 32 and 64 bits platforms. **Aleph** comes with a rich set of libraries that are designed to be platform independent. **Aleph** is a free software. A flexible license has been designed for both individuals and corporations. Everybody is encouraged to use, distribute and/or modify the aleph engine for any purpose.

Features

The **Aleph** engine is written in C++ and provides runtime compatibility with it. Such compatibility includes the ability to instantiate C++ classes, use virtual methods and raise or catch exceptions. A comprehensive API has been designed to ease the integration of foreign libraries.

- Builtin objects
  More than 50 reserved keywords and predicates. Various containers like list, vector, hash table, bitset, and graphs.
- Functional programming
  Support for *lambda expression* with explicit closure. Symbol scope limitation with *gamma expression*. Form like notation with an easy block declaration.
- Object oriented
  Single inheritance object mechanism with dynamic symbol resolution. Native class derivation and method override. Static class data member and methods.
- Multi-threaded engine
  True multi-threaded engine with automatic object protection mechanism against concurrent access. Read and write locking system and thread activation via condition objects.
- Original regular expression
  Builtin regular expression engine with group matching, exact or partial match and substitution.

Aleph is a core language and libraries. The libraries are a specific set of classes and functions which are structured per application domains. **Aleph** is delivered with a set of standard libraries.

- **aleph-sys**
  The `aleph-sys` library is the system calls library. Standard classes and functions are provided to interact with the running machine.
- **aleph-sio**
  The `aleph-sio` library is the standard input/output All input/output operations are performed with this library.
- **aleph-net**
  The `aleph-net` library is the networking library. The library is based on the standard *Internet Protocol* and provides various classes to manipulates IP address, client or server sockets.
- **aleph-www**
  The `aleph-www` library is the World Wide Web library. The library provides various classes that ease the development of web applications or CGI scripts.

Aleph provides *extensions.* An extension is a library or an application which is not installed by default. The user selects during the installation process which extension is needed. For example, the static version of the interpreter is an extension.

Aleph engine

Aleph is an interpreted language. When used interactively, commands are entered on the command line and executed when a complete and valid syntactic object has been constructed. Alternatively, the interpreter can execute a source file. **Aleph** does not have a garbage collector. **Aleph** operates with a lazy, scope based, object destruction mechanism. Each time an object is no longer visible, it is destroyed automatically. At this time, the **Aleph** interpreter is unable to reclaim memory with circular structures. This is a well known problem when using a reference count mechanism. In the future, the **Aleph** engine will provide some mechanisms to resolve this problem.

Flexible Distribution

Aleph is a free software. A flexible license model encourages individuals or corporations to use, copy, modify and/or distribute this software. **Aleph** is designed by software professionals. Quality is one the driving force of the development effort. This is reflected in this distribution by the extensive documentation. A large test suite is used to assess the quality of the distribution. Right now, the engine has been successfully tested on most Linux platforms, Free BSD and Solaris.

License

Aleph is a free software. It can be used, modified and redistributed by anybody for personal or commercial use. The only restriction is altering the copyright notice associated with the material. Individual or corporation are permitted to use, include or modify the Aleph engine. All material developed with the Aleph language belongs to their respective copyright holder.

This program is free software. You can redistribute it and/or modify it provided that this copyright notice is kept intact. This program is distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. In no event shall the copyright holder be liable for any direct, indirect, incidental or special damages arising in any way out of the use of this software.

Chapter 1

Release Notes

Welcome to **Aleph**. This manual contains the release notes and the installation guide. The latest distribution is available at *www.aleph-lang.org* or by FTP at *ftp.aleph-lang.org*.

1.1 Contents

This is the release **0.8.1** of the **Aleph Programming Language**. This release contains the aleph core language, its associated libraries and the documentation. Examples and utilities are also included in this distribution. The **Aleph** core language is released as a source file. In order to use **Aleph** , you will have to compile the distribution and install it. More information about this operation is given in chapter 2.

1.1.1 Core language

The **Aleph** core language comes with an interpreter and a set of predefined keywords, functions and base objects. For a short introduction on the language itself, please refer to the preface of this document or the tutorial in the volume 1 of this documentation set.

1.1.2 Libraries

Aleph is released with several libraries. A library is a specialized collection of classes and functions. A library is a shared object which is loaded on demand. The complete library description is given in the *Volume 2 - Library Reference Manual*. Some libraries have a also a manual of their own.

- **aleph-sio**
  The standard input/output library. This library contains the classes and functions related to input/output operations. Files and directory classes are part of this library.
- **aleph-sys**
  The system calls library. This library contains various system calls and classes.
- **aleph-net**
  The network library. This library contains all classes which can be used to build server or client applications. The library is largely based on the TCP/IP socket model.

- **aleph-www**
  The *web* library. This library contains the functions and classes that can be used to develop web applications or CGI scripts.

1.1.3 Documentation

The Aleph documentation is organized in various volumes. Each volume is distributed as a Postscript or PDF file. The complete documentation set is also available by FTP or via the Web site.

- **Volume 0: Installation guide**
  This volume is the distribution installation guide. It also contains useful notes for the package maintainer as well as some extra notes about the distribution.
- **Volume 1: Programmer's guide**
  This volume is a tutorial and a reference manual of the language. The first chapter contains a quick tour of the language features.
- **Volume 2: Library reference**
  This volume is the library reference manual. It contains a complete descriptions of the classes and functions.

1.2 New features

This section describes quickly the latest changes that have been made in this release.

1.2.1 Core system

- **Librarian axl**
  A new program called **axl** *Aleph Cross Librarian* is a utility that combines several files into one. The result is called a *Librarian file*. Such file can be used later within the **Aleph** interpreter.
- **Compiler axc**
  A new program called **axc** *Aleph Cross Compiler* is a utility that at this time performs syntax analysis and generates a serialized binary version of the original program. The resulting file is portable across platforms.

1.2.2 Core language

- **Relatif number**
  The `Relatif` object has been added to provided support for infinite precision integer arithmetic. The relatif number *(aka big-num)* supports all standard integer methods.
- **Librarian object**
  The **Librarian** object has been added as a standard object within the **Aleph** interpreter. The librarian object allows the creation or extraction of files in a programmable way, as the **axl** program do.
- **Resolver object**
  The **Resolver** object has been added as a standard object within the **Aleph** interpreter. The object resolves a name according to a search path that can be build out of directories or librarians.

- **Interpreter member data**
  The `program-name` data member has been added. The data member evaluates to a string returning the program name.

- **Closure arguments**
  The `args` reserved keyword has been added to support extended closure arguments. When arguments are passed to a closure, they are first bound to the formal arguments. The rest is bound to the closure `args` list.

- **Functional programming**
  The `Lexical`, `Qualified`, `Symbol` and `Closure` objects have been enhanced to support the functional programming paradigm. A new chapter in the documentation has been added specifically to cover this subject.

1.2.3 Library aleph-sys

- **User name**
  The **get-user-name** function has been added. The function returns the current user name running the interpreter.

- **Time object**
  The **Time** object has been enhanced to report the date and time according to standard formats.

1.2.4 Library aleph-sio

- **I/O multiplexing**
  The **Selector** object has been added to provide support for I/O multiplexing. The selector reacts to **Input** and **Output** stream objects.

- **InputString class**
  A new input stream object called **InputString** has been added to the standard i/o library. Such object gives the developer the opportunity to use a string as an input stream.

- **OutputString class**
  A new output stream object called **OutputString** has been added to the standard i/o library. Such object gives the developer the opportunity to write a string with the **Output** methods.

- **InputMapped class**
  A new input stream class called **InputMapped** has been added to the standard i/o library. The class behaves like an **InputFile** except that the file is mapped into memory and accept an offset and a size. Such object permits to treat a particular block in a file like an input file.

- **OutputFile class**
  A new constructor has been added to the `OutputFile` file. The new constructor takes several flags which controls the truncate and append mode.

1.2.5 Library aleph-net

- **Mail delivery**
  The **Mail** object has been added to provide support for mail delivery. The class follows the RFC822 for address specification.

1.2.6 Library aleph-www

- **Cookie**
  The **Cookie** object has been added to provide support for http cookies. Once a cookie has been created, it can be added into the HTML page with the `add-cookie` method.

Chapter 2

Installation Procedures

This chapter describes the installation procedures for the **Aleph** programming language distribution. **Aleph** is distributed as a source code only. This chapter explains how to set and compile this distribution.

2.1 Getting Aleph

Aleph is available either by FTP or from the *Aleph Web* site. The standard address is **aleph-lang.org**. **Aleph** has been built with the following platforms.

Platform	**Description**
Linux	Linux x86, alpha, m68k, Sparc, PowerPC, Arm and Mips
FreeBsd	FreeBsd x86 4.x (reported to work with 5.0)
Solaris	Solaris 2.5.1, 2.7 and 2.8

Table 2.1: Aleph supported platforms

Additionnaly, Aleph is part of the **Debian** distribution and the **FreeBsd** port collection.

2.2 Installation

Aleph is written in C++. You will need a modern compiler with support for exception and rtti. Aleph has been successfully built with the GNU `gcc 2` or `gcc 3`. No other compiler have been tested yet. You will need also *GNU Make*. With *Free BSD* the command is called `gmake`. Note that the *Makefile* hierarchy is designed to operate safely with the `-j` *GNU Make* option.

2.2.1 Unpacking the distribution

The distribution is available as a compressed tar file. Note that the documentation is distributed in a separate `tar` file. The following command unpacks the distribution.

```
zsh> gzip -d aleph-src-[version].tar.gz
zsh> tar  xf aleph-src-[version].tar
```

The `version` field is the current aleph distribution version.

2.2.2 Quick command reference

Here are the commands you should execute. If you are not sure of what is done, read the section below. If you are using a platform with `gcc` and want the distribution to be installed in the `/usr/local` directory, you can simply enter the `make world` command. If you want more tuning for the configuration, please continue to read this section. Below is a quick command reference to better control the build process.

```
zsh> ./cnf/bin/aleph-setup -o --prefix=/usr/local/aleph
zsh> make [-j]
zsh> make test
zsh> make install
zsh> make clean
```

or with FreeBSD

```
zsh> ./cnf/bin/aleph-setup -o --prefix=/usr/local/aleph
zsh> gmake [-j]
zsh> gmake test
zsh> gmake install
zsh> gmake clean
```

2.2.3 Configuration

The utility `aleph-setup` can be invoked to setup a particular configuration. You should have your compiler on your search path. Normally, the command below is enough.

```
zsh> ./cnf/bin/aleph-setup -o --prefix=/usr/local/aleph
```

This command checks that your platform can be detected and configure the platform. The `-o` option configure the compilation in optimized mode. Use the `-g` options to compile in debug mode. The `--prefix` option specify the installation directory. Note that the compilation process is done in the distribution tree. The `-v` option is the verbose option. Other options are available for fine tuning.

```
zsh> ./cnf/bin/aleph-setup -h
usage: aleph-setup [options]
       -h                 print this help message
       -v                 be verbose
       -g                 set debug mode
       -o                 set optimized mode
       -s                 compile and link statically
       --help             print this help message
       --prefix           set target directory to install
       --compiler         set default compiler
       --proctype         set processor architecture
       --soname           use soname during linking
       --static           compile and link statically
```

The `compiler` option can be used to force a particular compiler configuration file. The `proctype` option can be used to force a particular processor architecture. More on this one later. The `-s` or `--static` option can be used to build a *static* executable. Normally, you should not use this option since it restrict the use of extension modules. The `soname` option controls whether or not you want the dynamic libraries built with a version number and their `soname` changed accordingly. This options is detected automatically for a particular platform and should be used only by package maintainer.

2.2.4 Compiling the distribution

The compilation process is straightforward. Just enter the command:

```
zsh> make [-j]
```

This will build the complete distribution locally. If you have an error, at this stage, it is best to report it at `bugs@aleph-lang.org`.

2.2.5 Testing the distribution

The distribution contains all test suites. The test suites are compiled and executed with the following command.

```
zsh> make [-j] test
```

2.2.6 Installing the distribution

Once the system has been built and tested, it can be installed. By default, the installation goes into the `/usr/local` directory. This can be overwritten with the `--prefix` option during the configuration process.

```
zsh> make install
```

There are several variables that controls the behavior of the `install` rule.

- **PREFIX**
  This variable defines the root destination directory. Its default is `/usr/local`. This variable is set during the configuration stage, but can be overwritten.
- **BINDIR**
  This variable defines the binary destination directory. Its default is `prefix/bin`.
- **LIBDIR**
  This variable defines the library destination directory. Its default is `prefix/lib`.
- **HDRDIR**
  This variable defines the include destination directory. Its default is `prefix/include/aleph`.
- **MANDIR**
  This variable defines the man page destination directory. Its default is `prefix/man`.
- **ETCDIR**
  This variable defines the extra destination directory. Its default is `prefix/etc/aleph`.

Each variable can be overwritten during the installation process. This is done with the `install` rule.

```
make install BINDIR=/usr/bin LIBDIR=/usr/lib
```

2.2.7 Cleaning the distribution

The distribution can be cleaned with the `clean` rule.

```
zsh> make clean
```

This rule does not clean the configuration. For a complete cleaning the `distclean` rule is more appropriate.

```
zsh> make distclean
```

2.3 Running Aleph

The command **aleph** invokes the interpreter. You must have the `LD_LIBRARY_PATH` environment variable properly configured with the directory containing the aleph shared libraries. If you have installed all libraries in a standard UNIX location like `/usr/local/lib`, you should not have to set the variable.

2.3.1 Running some examples

The directory `exp` contains various examples which can be run. Each example is labeled according to their use in the volume 1 of the documentation set. Example `0101.als` prints the message *hello world*. Example `0501.als` prints various information about the system configuration, the interpreter revision, etc.

```
zsh> aleph 0101.als
hello world

zsh> aleph 0501.als
major version number   : 0
minor version number   : 8
patch version number   : 1
interpreter version    : 0.8.1
operating system name  : linux
operating system type  : unix
aleph official url     : http://www.aleph-lang.org
```

2.4 Special targets

The distribution can be configured to operate on a specific machine target. For example, a typical Linux-PC box will be compiled with the default compiler target, which is the 386 processor. You can force the compilation to be optimized for a particular processor. This is done with the `--proctype` option of the `aleph-setup` utility. Currently the distribution supports the `586` and `686` for the Intel architecture. The `ultra` architecture is valid for the Sparc architecture.

```
zsh> cnf/bin/aleph-setup -o --prefix=/usr/local --proctype=586
```

This command will configure the distribution to be compiled specifically for the `Pentium` architecture.

```
zsh> cnf/bin/aleph-setup -o --prefix=/usr/local --proctype=ultra
```

This command will configure the distribution to be compiled specifically for the `UltraSparc` architecture.

2.5 Extensions

Extensions are specific libraries or executables which are not build automatically during the build process. The user is responsible to decide which extension is needed for the system All extensions are located under the `src/ext` directory. Simply going into the appropriate directory and running the `make` command will build the extension.

2.5.1 Aleph static interpreter (asi)

The `asi` extension creates a static interpreter with all libraries automatically included in the final executable. The extension is simply build with the following command. Note that this extension overwrite the previous executable in the `bld/bin` directory. The following command build the `asi` extension.

```
zsh> make -C src/ext/asi
```

2.6 Documentation

The documentation is available as a tar file in both *postscript* or *pdf* files. To unpack the documentation use the following commands.

```
zsh> gzip -d aleph-doc-[version].tar.gz
zsh> tar  xf aleph-doc-[version].tar
```

The documentation is delivered as a set of manuals in Postscript or PDF format. It is your responsibility to install the documentation in a specific location of your choice.

2.6.1 Extra files

Aleph comes with some extra files. The most important is the `aleph-mode` for Emacs. The original source file written in `Emacs Lisp` is available in the `etc` directory of the distribution. You should install this file according to your current Emacs installation.

2.6.2 Rebuilding the documentation

If you have LaTeXand/or `pdflatex` available, the documentation can be rebuild with the following rules.

- `doc`
  This rule rebuilds the full documentation set and place it into the `bld` directory. Both Postscript and PDF manuals are generated.

- `epsman`
  The rule rebuilds only the Postscript manuals and place them in the `bld` directory.

- `pdfman`
  The rule rebuilds only the PDF manuals and place them in the `bld` directory.

Once the documentation is rebuild, it can be installed in a location of your choice.

Chapter 3

Package Maintainer Notes

This chapter contains additional notes for the package maintainer. They are also useful for anybody who is in charge of integrating the **Aleph** distribution in a build process. The chapter describes the distribution tree with more details.

3.1 The Distribution Tree

The distribution tree is composed of various directories. Each of them has a `Makefile` which can be called locally or from the top level.

- **`cnf`**
  This directory contains the configuration distribution and various utilities. Normally you should not touch it, unless you are using a compiler different than `gcc`.
- **`src`**
  This directory contains the complete source tree. The source code is written in C++. Normally this directory is left untouched. If there are good reasons to modify it, please let us know.
- **`tex`**
  This directory contains the complete documentation *latex* source tree. It should be left untouched.
- **`etc`**
  This directory contains various files associated with the distribution. You might want to copy some of them.
- **`exp`**
  This directory contains various examples. They are included for illustration purpose.

3.2 Configuration and setup

The configuration process involves the use of the `cnf/bin/aleph-setup` utility. This utility is used to configure the distribution. Package maintainers are encouraged to use it with specific options.

3.2.1 Platform detection

The utility `cnf/bin/aleph-guess` is used during the configuration process to detect a supported platform. This utility is a shell script and can be run in stand-alone mode. Various options can be used to tune the type of information requested.

```
zsh> ./cnf/bin/aleph-guess -h
usage: aleph-guess [options]
       -h                 print this help message
       -n                 print the platform name
       -v                 print the platform version
       -M                 print the platform major number
       -m                 print the platform minor number
       -p                 print the processor name
       -t                 print the processor type
```

Without option, the utility prints a platform and processor description string.

```
zsh> ./cnf/bin/aleph-guess
linux-2.2-ia-generic
```

3.2.2 Platform defaults

The directory `cnf/def` contains a platform specific default file. The file determines what is the default compiler and linking mode. This file is used by the `aleph-setup` utility. For example, the `aleph-linux.def` file contains:

```
compiler:       gcc
lkmode:         soname
```

Such options instructs the configuration utility, that the default compiler is `gcc` and the linking mode should use the *soname* option. These default values can be overwritten with the equivalent option of the `aleph-setup` utility. Note that the compiler version is automatically detected by the system. The utility `aleph-vcomp` will return the appropriate compiler version running on your system.

3.2.3 C++ source file convention

Aleph has two types of C++ file. The first type has the extension `.cxx` and the second type has the extension `.cpp`. The `.cxx` (and the associated `.hxx`) files are only used to indicate a system dependency. These files are found only in the `src/plt/lib` directory. The `.cxx` extension indicates that the file might use system specific include files. The `.cpp` (and the associated `.hpp`) files are the normal C++ source files. The `.cpp` extension is used to indicate that these files will not use a system specific file. By default this rule is enforced in the *compiler file* by specifying some compiler flags which do not authorize such access.

3.2.4 Configuration files

The configurations files are located in the `./cnf/mak` directory. Normally they should be left untouched. The most important one is the `aleph-rule.mak` file that defines most of the compilation and linking rules. Additionnaly, during the setup operation, the `aleph-setup` utility creates two files in the `./bld/cnf` directory. The `./bld` is the build directory. The file `aleph-plat.mak` is the platform configuration file and the `aleph-comp` is a link to the appropriate compiler configuration file.

3.3 Compilation

Normally, the compilation process is immediate. Just invoking the command `make` will do the job. However, some package maintainer have the desire to overwrite some flags. Some options are provided to help you in this task.

- **`EXTCPPFLAGS`**
  This flag can be used to add some compilation flags for all `.cpp` files.
- **`EXTCXXFLAGS`**
  This flag can be used to add some compilation flags for all `.cxx` files.
- **`EXTCCDEFINE`**
  This flag can be used to add some compilation definitions for all source files.
- **`EXTINCLUDES`**
  This flag can be used to add some compilation paths for the *cxx* files.

.

For example, it is common to have some maintainer to compile with both the debug and optimize flags. This can be done with the following command (assuming an optimized configuration):

```
make EXTCPPFLAGS=-g EXTCXXFLAGS=-g
```

3.3.1 Compilation results

All include files, compiled libraries and executables are placed in the `bld` directory. This directory contains the `bld/bin` for binaries, `bld/lib` for libraries and `bld/hdr` for the header files.

3.4 Building the package

The package can be built by accessing the `bld` directory or by invoking the `install` rule. The second method is not recommended for package construction, since it might trigger some file installation without your control.

3.4.1 Special files

The `etc` directory contains some special files that might be used for the package construction. Here is a sample list of them.

- **`license.txt`**
  This file is the license file.
- **`what-short.txt`**
  This file is a short description of what is **Aleph**.
- **`what-long.txt`**
  This file is a longer description of what is **Aleph**.
- **`aleph.man`**
  This file is the **Aleph** man page.
- **`aleph-mode.el`**
  This file is the **Aleph** Emacs mode.

3.5 Building the documentation

The documentation is available as *Postscript* or *PDF* files. The documentation can be rebuild assuming *latex* and *pdflatex* are available on the system. The `doc` rule rebuilds all documentation files and place them in the `bld` directory. Note that the documentation is also available as a precompiled *tar* file.

3.6 Other make rules

The top level `Makefile` contains other rules that might be useful.

- `debug`
  This rule invokes the default configuration in debug mode.
- `optimized`
  This rule invokes the default configuration in optimized mode.
- `build`
  This rule invokes the default configuration in debug mode and compile the whole distribution. The default install directory is `/usr/local`.
- `world`
  This rule invokes the default configuration in optimized mode and compile the whole distribution. The default install directory is `/usr/local`.
- `test`
  This rule runs all test suites.
- `doc`
  This rule builds the documentation
- `distri`
  This rule builds the distribution
- `install`
  This rule install the distribution
- `clean`
  This rule cleans the distribution but keep the configuration.
- `distclean`
  This rule cleans the distribution including the configuration.