libALF Compilation Guide

This document will guide you through the compilation and installation of libALF on Linux and Windows.

libALF works on Linux and Windows on both 32- and 64-bit architectures. However, as libALF has no prerequisites, it is most likely that it also runs on various additional operating systems. If you want to compile libALF for another operating system, e.g. MacOS X, the guidelines for compiling libALF for Linux may be a good reference.

This document is organized in six sections: The first two sections describe how to obtain libALF (if you not already have) and what prerequisites need to be fulfilled. Sections 3 to 5 show how to compile and use libALF, jALF and dispatcher. The sixth section gives help on troubleshooting.

1 libALF Package Information

The library is freely available under the open source LGPL v3 license at the libALF website http://libalf.informatik.rwth-aachen.de. Download the libALF package and extract it to a folder of your choice. The package contains the following components:

- The libALF C++ library.
- jALF (libALF's Java interface).
- The dispatcher (a network-based libALF server).

This guide will demonstrate how to employ and use libALF in user applications through *examples* available at libALF's website. We recommend that you download and extract the example sources to a folder of your choice.

2 Prerequisites

The libALF library itself does not have any prerequisites, but some components have. To use the additional components, please ensure that the following requirements are satisfied.

- For compiling and using jALF you need a Java Development Kit (JDK) Version 6.0 or later installed. Moreover, we recommend using the *Ant* build tool downloadable from http://ant.apache.org/.
- The dispatcher requires a POSIX-compliant operating system. While there is no problem under Linux, the dispatcher will not compile under Windows.

Linux. For compiling the C++ sources in Linux, you require a C++ compiler (this document assumes that you use the *GNU C++ compiler*) and the *make utility*, which is used to automate the build process. Both tools should be installed by default on every Linux machine.

Windows. To compile the C++ sources on Windows, we recommend using the *Minimalist GNU for Windows (MinGW)* compiler and *MSYS*, a Unix-style shell for Windows. Both can be obtained from http://www.mingw.org/.

Please follow the instructions on the website to set up MinGW and MSYS properly. In particular, make sure that you install the MSYS make package (if not done automatically). Using MSYS gives you the advantage of following all instructions described in this document no matter whether you use Linux or Windows. However, please be careful with folder names that contain blanks; you may have to enclose them in quotes and replace every blank with a backslash followed by a blank (or, in the best case, you avoid them completely).

3 The libALF C++ Library

The section will describe how to compile and install the library as well as how to run applications that use the library.

3.1 Compiling libALF

You have the choice to compile the libALF library either as a *static* or as a *shared* library. If you do not know the difference or if you just want to use the library, you should compile a shared library as described below and follow the respective instructions for running your application.

Compiling a Shared Library

Compiling libALF is easy: simply change into the libalf/src folder and invoke the make utility by typing

```
make
```

The make utility automatically detects which operating system you are running and compiles the library accordingly. After the compilation you should find the binary file libalf.so (on Linux) or libalf.dll (on Windows) inside the libalf/src folder. However, if you experience problems, you can explicitly tell the make utility for which system you want to compile libALF by typing make libalf.so (under Linux) or make libalf.dll (under Windows).

Compiling a Static Library

You can compile a static library using the command (inside libalf/src)

```
make libalf.a
```

on both Linux and Windows.

However, make sure that you delete any shared library in the folder before you link your application with libalf as some operating systems (e.g. Linux) always prefer shared libraries if present.

3.2 Installing libALF

Installing libALF means to copy to the compiled shared library and libALF's headers to a location where your operating system finds them.

Linux. To install libALF in Linux, first compile the library and type `make install`. You can uninstall libALF by using the command `make uninstall`. Please note that you need root privileges for both actions.

Windows. On Windows, you have to manually copy the compiled shared binary files to your windows/system directory. Unfortunately, there is no common place to put header files in. Thus, you have to specify the header's location every time you compile an application that uses libALF (see the section below).

3.3 Compiling Applications That Use libALF

When compiling an application that uses libALF, the compiler needs to find libALF's headers and the compiled library. Please note that you do not have to provide this information if you have libalf installed on your system.

Otherwise, you have to use the GNU C++ compiler's -I parameter to specify libALF's header locations (typically libalf/include) and the -L parameter to specify the location for the compiled library (which is libalf/src). You also have to link the application to libALF using -lalf.

We will consider the online-example to explain the compilation.

Compiling applications that links to shared library. To compile the online example that uses the shared library, type the following command.

```
g++ -I path_to_headers -L path_to_library online.cpp -lalf
```

Compiling applications that links to static library. If you want to link libalf statically into your application, you can do so by adding `-static` as additional parameter just before linking to libalf like below.

```
g++ -I path_to_headers -L path_to_library online.cpp -static -lalf
```

In both cases, it is also a good idea to specify the name of the output file using the `-o` parameter, e.g. `-o online`.

Additional Parameter for Windows. Please note that on Windows the Winsock2 library has to be linked additionally to every program using `libALF`. You can do this by adding `-lws2_32`. Again it is crucial that you add this parameter after all input files.

3.4 Running Applications That Use `libALF`

An application *statically* linked to libalf can be executed as usual. However, if you run a program that uses `libALF` as a *shared library*, you need to specify where your operating system can find the library (again, you do not need to provide this information if you have installed `libALF` on your system).

Linux. On Linux, use the `LD_LIBRARY_PATH` variable to point to the location of the shared library. For instance, you can run the above compiled online example with the command

```
LD_LIBRARY_PATH=path_to_library ./online
```

Windows. Unfortunately, on Windows there is no direct way of telling the system where to find shared libraries. Instead, you have to add their locations to Windows' `PATH` variable or copy the library into the folder your application is executed from. Then, execute your application as usual.

For further details please refer to the examples' Readme and Makefile.

4 The `jALF` Java Library

`jALF` is the Java interface to `libALF`. It lets you access `libALF` via the dispatcher or via Java's *Native Inter-face JNI*. The latter way requires that you compile a second `C++` library (some kind of wrapper), that obeys Java's naming convention and performs some basic conversions of internal data structures. However, if you want to use `jALF` only in connection with the dispatcher, it is enough to compile and use the Java sources.

In the following we assume that you are familiar with basics of compiling and running Java programs.

4.1 Compiling jALF's Java sources

In order to compile jALF's Java sources, change to the libalf/jalf folder and type

```
ant
```

This invokes the Ant build utility and produces the file jalf.jar containing all compiled class files inside the libalf/jalf folder. If you do not wish to use jALF via JNI, you can skip compiling jALF's C++ sources.

Note that you can generate jALF's *JavaDoc* also using Ant with the command ant doc. Thereafter, the JavaDoc can be found inside the libalf/jalf/java/doc folder.

4.2 Compiling jALF's C++ Sources

The jALF C++ library needs to be a shared library. However, you have the option to link libALF either dynamically or statically to jALF. The latter option is often preferred and enabled by default. Please remember to delete any shared library in libalf/src before you compile jALF's C++ sources (libALF is recompiled for you). You may use the command make -C libalf/src clean for this.

Compiling jALF's C++ sources is also automated by means of the make utility. However, as additional information the Java compiler requires the location of Java's JNI header files, which are contained in every JDK. Their location is passed on to the make utility using the JAVA_INCLUDE variable. Thus, to compile jALF's C++ sources, go to libalf/jalf/src and execute

```
JAVA_INCLUDE=path_to_jdk/include make
```

Again, the make utility should detect your operating system automatically, but you can also use the commands make libjalf.so (for Linux) and make jalf.dll (for Windows) to explicitly compile jALF for your desired operating system. After a successful compilation, the compiled binary is located in libalf/jalf/src.

Dynamic Linking. As mentioned, libALF is linked statically by default. If you want link libALF *dynamically*, you can use the commands make libjalf.so-dynamic (for Linux) and make jalf.dll-dynamic (for Windows).

4.3 Compiling Java applications that use jALF

Fortunately, the Java compiler does not need to know anything about the C++ libraries to compile your application and only needs access to jALF's Java class files. You specify this information by adding the jalf.jar file to Java's classpath. Our Java online example, for instance, can be compiled using the following command (first change into the folder containing the example sources):

```
javac -classpath "path_to_jalf/jalf.jar" Online.java
```

4.4 Running Java applications that use jALF

Besides the location of the jalf.jar, running a Java application that uses jALF requires telling Java where it can find the compiled jALF and libALF C++ libraries. (If you have installed libALF to your system or if you linked jALF statically to jALF, you do not need to bother about the latter.)

The place where Java looks for C++ libraries is controlled by Java's interval library path variable. This variable can only be changed at the start of the Java VM. You do so by setting the variable named java.library.path to the location of the jALF library (i.e, the jALF C++ binary which is typically libalf/jalf/src) using the -D parameter.

Linux. To run the online example on Linux, one has to execute the following command (inside the folder containing the compiled online example):

```
java -classpath "path_to_jalf/jalf.jar:."
     -Djava.library.path=path_to_jalf_library Online
```

If necessary, specify the location of the shared libALF library as described in Section 3.

Windows. Please recall that Linux and Windows use different ways of separating folders. While you must use a colon on Linux, you must use a semicolon on Windows; everything else is the same as before.

```
java -classpath "path_to_jalf/jalf.jar;."
     -Djava.library.path=path_to_jalf_library Online
```

For further details please refer to the examples' Readme.

5 Compiling and Using the Dispatcher

Please recall that the dispatcher only compiles and runs on a POSIX-compliant operating system such as Linux, but not on Windows.

5.1 Compiling the Dispatcher

To compile the dispatcher, first compile a shared libALF library as described Section 3.

Dynamic Linking. By default, the dispatcher is dynamically linked to libALF. To compile an executable linked statically, change into the folder libalf/dispatcher and execute

```
make
```

This creates the executable dispatcher in the same directory.

Static Linking. To link the dispatcher statically to `libALF`, use the following command

```
make dispatcher-static
```

Again, remember to remove any compiled shared library in `libalf/src` first.

5.2 Running the Dispatcher

The dispatcher is executed like any other executable on your system. However, remember to specify the location of the `libALF` shared library if necessary.

6 Troubleshooting

When experiencing troubles, the first thing you should try is to execute `make clean` in the `libalf` and `libalf/jalf` folders as well as `ant clean` in the `libalf/jalf` folder. This deletes all compiled files and solves most compiler and linker problems. However, if this does not work for you, you may find a solution for your problem in the list below:

- There are no known problems.