Wolframe Installation

Installation and Configuration Guide

Wolframe Installation: Installation and Configuration Guide

Publication date March 19, 2014 version 0.0.1

Table of Contents

Foreword

This guide describes how to install and set up the Wolframe application server.

Chapter 1. Installation from source

This section describes how to build the Wolframe application from the source code.

1.1. Source Releases

Tarballs with release source code are available from SourceForce in the directories

http://sourceforge.net/projects/wolframe/files/wolframe/ [http://openbuildservice.org]

respectively

http://sourceforge.net/projects/wolframe/files/wolfclient/ [http://openbuildservice.org] .

The `wolframe-0.0.1.tar.gz` contains the Wolframe server, the modules and 3rdParty software needed to build the server.

The `wolfclient-0.0.3.tar.gz` contains the Wolframe client implementing the graphical user interface.

1.2. Building on Unix systems

For building Wolframe on Unix systems you need at least the following software:

- A recent C/C++ compiler, the following ones are known to work:
  - gcc 4.1.x or newer, http://gcc.gnu.org
  - clang 3.4 or newer, http://clang.llvm.org
  - Intel Compiler ICC 14.0 or newer, http://software.intel.com/en-us/c-compilers
- GNU make 3.81 or newer (but preferably 3.82 or newer) from https://www.gnu.org/software/make/
- boost 1.48.0 or newer from http://www.boost.org

Wolframe is build and installed by simply calling:

```
make
make install
```

The makefiles understand the standard GNU targets like clean', 'distclean', 'test', 'install', 'uninstall', etc. Also the standard installation variables 'DESTDIR' and 'prefix' are understood. The whole list of options can be seen with:

```
make help
```

There is no configure. Porting to platforms and distributions is done in the makefiles. For most platforms we provide reasonable default values in `makefiles/gmake/platform.mk`.

Optional features are enabled by using 'WITH_XXX' variables when calling make, e. g. to enable SSL support you call make like this:

```
make WITH_SSL=1
```

Additional variables can be set when 3rdParty software is not in the standard location, for instance:

```
make BOOST_DIR=/usr/local/boost-1.55.0
```

You can check how your software will be build with:

```
make config
```

If you get a 'NOT SUPPLIED ON THIS PLATFORM' you have to provide the variables explicitly as mentioned above in the example with 'BOOST_DIR'.

1.2.1. GCC compiler

Compilation with GNU gcc is the default on all Unix platforms. It corresponds to the call:

```
make CC=gcc CXX=g++
```

Per default all reasonable warnings are enabled. To add your own flags you can set 'CFLAGS' or 'CXXFLAGS' respectively for instance to turn compiler warnings into fatal errors with:

```
make CFLAGS='-Werror' CXXFLAGS='-Werror'
```

or

```
make CFLAGS='-g -O0' CXXFLAGS='-g -O0'
```

to turn off optimization and to enable debug information.

Certain embedded 3rdParty software may choose to have it's own flags for compilation, you can't override those in the make invocation.

1.2.2. clang compiler

Compilation with clang is possible, only set the correct compiler variables:

```
make CC=clang CXX=clang++
```

Also here you can set 'CFLAGS' and 'CXXFLAGS' at will.

1.2.3. Intel compiler

Compilation with the Intel C compiler is done with:

```
source /opt/intel/bin/iccvars.csh intel64
make CC=icc CXX=icpc
```

(where '/opt/intel/bin/icc' is the location of the Intel compiler).

Also here you can set 'CFLAGS' and 'CXXFLAGS' at will.

When running the tests or any binaries you have to make sure that 'LD_LIBRARY_PATH' is set correctly (the example is for csh/tcsh, Intel 64-bit):

```
setenv LD_LIBRARY_PATH $PROD_DIR/lib/intel64
```

1.2.4. Platform-specific build instructions

FreeBSD

You need GNU make, BSD make doesn't work. You have to install the 'gmake' package.

Both FreeBSD 8 and FreeBSD 9 are supported.

NetBSD

You need GNU make, BSD make doesn't work. You have to install the 'gmake' package.

NetBSD 6 is supported, NetBSD 5 not.

Packages are installed with 'pkgin' into the directory `/usr/pkg`. Make sure `/usr/pkg/bin` is part of your PATH.

OpenIndiana 151a8

The official gcc is too old to build Boost. The Forte compiler is not free and has big problems to compile modern C++ code. So we must use the CSW/gcc/C++ toolchain for Wolframe.

Install the CSW toolchain (http://www.opencsw.org) and basic development tools:

```
pkgadd -d http://get.opencsw.org/now
pkgutil --install CSWgcc4core CSWgcc4g++ CSWgmake
```

You also need some system files:

```
pkg install pkg:/system/header
pkg install pkg:/developer/library/lint
pkg install system/library/math/header-math
```

Make sure `/opt/csw/bin` is part of your PATH.

Install packages with 'pkgutil --install'.

Solaris 10

We only build for SPARC and Solaris 10 currently.

You may have to install a 'SFWgtar' or 'CSWgtar' in order to unpack the sources. Make sure to rename them to 'gtar' to avoid collisions with the standard 'tar'!

The official gcc is too old to build Boost. The Forte compiler is not free and has big problems to compile modern C++ code. So we must use the CSW/gcc/C++ toolchain for Wolframe.

Install the CSW toolchain (http://www.opencsw.org) and basic development tools:

```
pkgadd -d http://get.opencsw.org/now
pkgutil --install CSWgcc4core CSWgcc4g++ CSWgmake
```

Make sure the build environment is always set as follows:

```
PATH=/opt/csw/bin:/usr/ccs/bin:/usr/bin:/bin:/opt/csw/sbin:/usr/sbin:/sbin
export PATH
```

Install official packages with 'pkgadd -d' and CSW packages with 'pkgutil --install'.

Building Wolframe is more complex as on other platforms, so we provide this working example invocation of make:

```
LD_RUN_PATH=/opt/csw/lib:/opt/csw/postgresql/lib \
OPENSSL_DIR=/opt/csw PGSQL_DIR=/opt/csw/postgresql \
LIBLT_DIR=/usr BOOST_DIR=/opt/csw/boost-1.48.0 \
 WITH_EXPECT=1 WITH_SSL=1 WITH_SYSTEM_SQLITE3=1 WITH_PGSQL=1 WITH_LUA=1 \
 WITH_LIBXML2=1 WITH_LIBXSLT=1 WITH_PAM=1 WITH_SASL=1 WITH_LOCAL_LIBHPDF=1 \
 WITH_ICU=1 ICU_DIR=/opt/csw/icu4c-49.1.2 \
 WITH_LOCAL_FREEIMAGE=1 \
 WITH_PYTHON=1 \
 gmake \
  CC=gcc CXX=g++ CFLAGS=-mcpu=v9 CXXFLAGS=-mcpu=v9
```

1.2.5. Boost

Boost (http://www.boost.org [http://www.boost.or]) is the only library which is absolutely required in order to build Wolframe.

Build your own version of Boost

The following Boost libraries are required for building Wolframe:

```
./bootstrap.sh --prefix=/usr/local/boost-1.55.0 \
 --with-libraries=thread,filesystem,system,program_options,date_time
./bjam install
```

If you want to build the ICU normalization module (WITH_ICU=1) you will have to build 'boost-locale' with ICU support and you have to enable the 'regex' and the 'locale' boost libraries too:

```
./bootstrap.sh --prefix=/usr/local/boost-1.55.0 \
 --with-libraries=thread,filesystem,system,program_options,date_time,regex,loc
./bjam install
```

The location of the Boost library can be set when building Wolframe as follows:

```
make BOOST_DIR=/usr/local/boost-1.48.0
```

RedHat, Fedora, CentOS, Scientific Linux and similar Linux distributions

RedHat/Centos/Scientific Linux 5 and similar Linux distributions

The official Boost packages are not recent enough. Build your own Boost version here.

If you want ICU support you will also need the 'libicu-devel' package.

RedHat 6

The official Boost packages are not recent enough. Build your own Boost version here.

If you want ICU support you will also need the 'libicu-devel' package.

We currently build the official packages without ICU support. The reason is that there is no 'libicu-devel' package available for RHEL6 on OBS (see http://permalink.gmane.org/gmane.linux.suse.opensuse.buildservice/17779).

Get a Redhat developer license to get the 'libicu-devel' package or build your own libicu library and build your own Boost library with boost-locale and ICU support.

Centos/Scientific Linux 6 and similar Linux distributions

The official Boost packages are not recent enough. Build your own Boost version here.

If you want ICU support you will also need the 'libicu-devel' package.

Fedora and similar Linux distributions

You need the 'boost-devel' package. This package contains also the boost-locale and ICU backend.

Debian, Ubuntu and similar Linux distributions

Debian 6

The official Boost packages are not recent enough. Build your own Boost version here.

Debian 7

You need the following packages: 'libboost-dev', 'libboost-program-options-dev', 'libboost-filesystem-dev', 'libboost-thread-dev', 'libboost-random-dev'.

If you want ICU support you will also need the 'libboost-locale-dev' package.

Ubuntu 10.04.1 LTS, Ubuntu 12.04

The official Boost packages are not recent enough. Build your own Boost version here.

Ubuntu 12.10

You need the following packages: 'libboost-dev', 'libboost-program-options-dev', 'libboost-filesystem-dev', 'libboost-thread-dev', 'libboost-random-dev'.

If you want ICU support you will also need the 'libboost-locale-dev' package.

Ubuntu 13.04

Currently libboost breaks in enable_if template, compile your own Boost version here.

Ubuntu 13.10

You need the following packages: 'libboost-dev', 'libboost-program-options-dev', 'libboost-filesystem-dev', 'libboost-thread-dev', 'libboost-random-dev'.

If you want ICU support you will also need the 'libboost-locale-dev' package.

openSUSE, SLES and similar Linux distributions

OpenSuSE 12.3, 13.1

You need the 'boost-devel' package.

SLES 11 SP1, SP2 and SP3

The official Boost packages are not recent enough. Build your own Boost version here.

ArchLinux

You need the 'boost' and 'boost-libs' packages. The official Boost packages contains support for boost-locale and the ICU backend.

Slackware

You need the 'boost' package. This package is part of the 'l' package series. The official Boost package contains support for boost-locale and the ICU backend.

FreeBSD

You need the 'boost-libs' package.

Some boost header files are broken, for patches see `packaging/patches/FreeBSD`. They can be applied to the ports directory before rebuilding Boost or directly to the installed header files in `/usr/local/include/boost`.

NetBSD

You need the 'boost-libs' package.

OpenIndiana 151a8

We don't use the CSW boost packages.

As long you don't need ICU support you can build Boost as follows:

First apply all patches found in `packaging/patches/Solaris/1.54.0`.

Then build boost with:

```
./bootstrap.sh --prefix=/opt/csw/boost-1.54.0 \
 --with-libraries=thread,filesystem,system,program_options,date_time
./b2 -a -d2 install
```

Note: The only tested version for now is version 1.54.0! Other versions of Boost may work or not..

Solaris 10

We don't use the CSW boost packages.

As long you don't need ICU support you can build Boost as follows:

Apply the following patch to `boost/cstdint.ppp` for boost (https://svn.boost.org/trac/boost/ticket/6158, see also `packaging/patches/Solaris/boost-1.48.0` for patch files):

```
...
namespace boost
{

#if defined(sun) || defined(__sun)
 typedef signed char int8_t;
#else
 using ::int8_t;
#endif
..
```

Patch the correct architecture (V8 is not really supported, but V8 is also very old) in `tools/build/v2/user-config.jam`:

```
using gcc : 4.6.3 : g++ : <compileflags>-mcpu=v9 ;
```

Then build boost with:

```
./bootstrap.sh --prefix=/opt/csw/boost-1.48.0 \
 --with-libraries=thread,filesystem,system,program_options,date_time
./b2 -a -d2 install
```

Note: The only tested version for now is version 1.48.0! Other versions of Boost may work or not:

Do not use boost 1.49.0, it has a missing function fchmodat, see http://lists.boost.org/boost-build/2012/02/25680.php causing building of libboost_filesystem to fail!

Boost 1.50.0 and 1.51.0 have never been tested with Wolframe, so don't use those!

Support for 1.52.0 and later is not guaranteed.

1.2.6. Secure Socket Layer (SSL)

The Wolframe protocol can be secured with SSL. You have to specify the following when building:

```
make WITH_SSL=1
```

Currently only OpenSSL (http://www.openssl.org) is supported. The location of the library can be overloaded with:

```
make WITH_SSL=1 OPENSSL_DIR=/usr/local/openssl-1.0.1.f
```

Use the most recent version of the OpenSSL library available for you platform.

RedHat, Fedora, CentOS, Scientific Linux and similar Linux distributions

You need the 'openssl-devel' package.

Debian, Ubuntu and similar Linux distributions

You need the 'libssl-dev' package.

openSUSE, SLES and similar Linux distributions

You need the 'openssl-devel' package.

ArchLinux

You need the 'openssl' package.

Slackware

You need the 'openssl' package. This package is part of the 'n' package series.

FreeBSD

FreeBSD contains all necessary SSL libraries per default, you don't have to install any special packages.

NetBSD

NetBSD contains all necessary SSL libraries per default, you don't have to install any special packages.

OpenIndiana 151a8

You need the 'CSWlibssl-dev' package.

Solaris 10

You need the 'libssl_dev' package.

1.2.7. SQLite database support

Wolframe can use an Sqlite3 database (http://sqlite.org) as backend for data storage and for authentication and autorization.

You enable the building of a loadable Sqlite3 database module with:

```
make WITH_SYSTEM_SQLITE3=1
```

If you don't have a recent Sqlite version on your system you can also build the module against the embedded version:

```
make WITH_LOCAL_SQLITE3=1
```

The location of the Sqlite library can be overloaded with:

```
make WITH_SYSTEM_SQLITE3=1 SQLITE3_DIR=/usr/local/sqlite-3.4.3
```

You can also override all compilation and linking flags of Sqlite separately:

```
make WITH_SYSTEM_SQLITE3=1 \
 SQLITE3_INCLUDE_DIR=/usr/local/sqlite-3.4.3/include \
 SQLITE3_LIB_DIR= /usr/local/sqlite-3.4.3/lib \
 SQLITE3_LIBS=-lsqlite3
```

When building with 'WITH_SYSTEM_SQLITE3' it is enough to install the correct development library.

RedHat/Centos/Scientific Linux 5 and similar Linux distributions

The official Sqlite package is too old, use the embedded version of Sqlite with 'WITH_SYSTEM_SQLITE3=1'.

RedHat/Centos/Scientific Linux 6, Fedora and similar Linux distributions

You need the 'sqlite-devel' package.

Debian, Ubuntu and similar Linux distributions

You need the 'libsqlite3-dev' package.

For running the Sqlite3 database tests you also need the 'sqlite3' package.

openSUSE, SLES and similar Linux distributions

You need the 'sqlite3-devel' package.

For running the Sqlite3 database tests you also need the 'sqlite3' package.

ArchLinux

You need the 'sqlite' package.

Slackware

You need the 'sqlite' package. This package is part of the 'ap' package series.

FreeBSD

You need the 'sqlite3' package.

NetBSD

You need the 'sqlite3' package.

OpenIndiana 151a8

You need the 'CSWlibsqlite3-0' and the 'CSWlibsqlite3-dev' packages.

Solaris 10

You need the 'CSWlibsqlite3-0' and the 'CSWlibsqlite3-dev' packages.

1.2.8. PostgreSQL database support

Wolframe can use a PostgreSQL database (http://postgresql.org) as backend for data storage and for authentication and autorization.

You enable the building of a loadable PostgreSQL database module with:

```
make WITH_PGSQL=1
```

The location of the PostgreSQL library can be overloaded with:

```
make WITH_PGSQL=1 PGSQL_DIR=/usr/local/postgresql-9.1.3
```

You can also override all compilation and linking flags of PostgreSQL separately:

```
make WITH_PGSQL=1 \
 PGSQL_INCLUDE_DIR=/usr/local/postgresql-9.1.3/include \
 PGSQL_LIB_DIR=/usr/local/postgresql-9.1.3/lib \
 PGSQL_LIBS=-lpq
```

RedHat, Fedora, CentOS, Scientific Linux and similar Linux distributions

You need the 'postgresql-devel' package.

For Centos/RHEL/SciLi 5 you can choose between the 'postgresql-devel'package (which is version 8.1) or the 'postgresql84-devel' package. The 8.4 version is recommended over 8.1.

For running the Postgresql tests you need a fully function 'postgresql-server' with a db user 'wolfusr' (password: 'wolfpwd') owning a database called 'wolframe'.

Setting up a test user in PostgreSQL is done with:

```
postgresql-setup initdb
systemctl start postgresql.service
systemctl enable postgresql.service
```

change the authentication method from 'ident' to 'md5' in `pg_hba.conf`.

Debian, Ubuntu and similar Linux distributions

You need the 'libpq-dev' package.

For running the PostgreSQL database tests you also need the 'postgresql-client' package.

openSUSE, SLES and similar Linux distributions

You need the 'postgresql-devel' package.

ArchLinux

You need the 'postgresql-libs' package.

If you want to test you also have to set up the PostgreSQL server which comes in the 'postgresql' package.

Slackware

A PostgreSQL package is not available on Slackware, build your own one with:

```
./configure --prefix=/usr/local/pgsql
make
make install
groupadd -g 990 postgres
useradd -g postgres -u 990 postgres
mkdir /usr/local/pgsql/var
chown -R postgres:users /usr/local/pgsql/var
su postgres
/usr/local/pgsql/bin/initdb -D /usr/local/pgsql/var
exit
cat > /etc/rc.d/rc.postgresql
#!/bin/sh

case "$1" in
 start)
  su -l postgres -s /bin/sh -c "/usr/local/pgsql/bin/pg_ctl  -D /usr/local/pgs
  ;;
 stop)
  kill `ps -efa | grep postmaster | grep -v grep | awk '{print $2}'`
  ;;
 *)
  echo $"Usage: $0 {start|stop}"
  exit 1
esac

exit 0
(ctrl-D)
chmod 0775 /etc/rc.d/rc.postgresql
usermod -d /usr/local/pgsql postgres
```

Compile Wolframe now with:

```
make WITH_PGSQL=1 \
 PGSQL_DIR=/usr/local/pgsql
```

Alternatively you can of course also build the 'postgresql' package with the help of SlackBuilds.

FreeBSD

You need the 'postgresql92-client' package.

For testing you also need the 'postgresql92-server' package.

NetBSD

You need the 'postgresql92-client' package.

For testing you also need the 'postgresql92-server' package.

OpenIndiana 151a8

You need the 'CSWpostgresql-dev' package.

For testing you also need the 'CSWpostgresql91-server' package.

Solaris 10

You need the 'CSWpostgresql-dev' package.

For testing you also need the 'CSWpostgresql91-server' package.

1.2.9. Oracle database support

Wolframe can use a Oracle database (http://www.oracle.com) as backend for data storage and for authentication and autorization.

Import note: Make sure you have all the licenses to develop with Oracle and to install an Oracle database! The Wolframe team doesn't take any responsability if licenses are violated!

You enable the building of a loadable Oracle database module with:

```
make WITH_ORACLE=1
```

The location of the Oracle instantclient library can be overloaded with:

```
make WITH_ORACLE=1 ORACLE_DIR=/opt/oracle/instantclient_11_2
```

You can also override all compilation and linking flags of Oracle separately:

```
make WITH_ORACLE=1 \
 ORACLE_INCLUDE_DIR=/usr/lib/oracle/11_2/client64 \
 ORACLE_LIB_DIR=/usr/lib/oracle/11_2/client64 \
 ORACLE_LIBS=-lclntsh
```

If you want to run the tests for Oracle you'll have to set up an Oracle database. Then install the 'wolframe' database and the 'wolfusr' database user. Sql example files can be found in `contrib/database/oracle`.

RedHat, Fedora, CentOS, Scientific Linux and similar Linux distributions

For building the Oracle database module you have to download the RPM packages `oracle-instantclient12.1-basic-12.1.0.1.0-1.i386.rpm` and `oracle-instantclient12.1-devel-12.1.0.1.0-1.i386.rpm`. You can of course also install the zipfiles and install those.

From the system repositories you'll need the 'libaio' package.

If you want to use the 'sqlplus' command line tool for manual testing you also have to install the package `oracle-instantclient12.1-sqlplus-12.1.0.1.0-1.i386.rpm`. If you want a history in sqlplus it's highly recommended that you install a command line history wrapper like for instance 'rlwrap'.

Debian, Ubuntu and similar Linux distributions

For building the Oracle database module you have to download the RPM packages `oracle-instantclient12.1-basic-12.1.0.1.0-1.i386.rpm` and `oracle-instantclient12.1-devel-12.1.0.1.0-1.i386.rpm`.

To install those RPM files you'll need the 'alien' tool. You can of course also install the zipfiles and install those.

From the system repositories you'll need the 'libaio1' package.

If you want to use the 'sqlplus' command line tool for manual testing you also have to install the package `oracle-instantclient12.1-sqlplus-12.1.0.1.0-1.i386.rpm`. If you want a history in sqlplus it's highly recommended that you install a command line history wrapper like for instance 'rlwrap'.

openSUSE, SLES and similar Linux distributions

For building the Oracle database module you have to download the RPM packages `oracle-instantclient12.1-basic-12.1.0.1.0-1.i386.rpm` and `oracle-instantclient12.1-devel-12.1.0.1.0-1.i386.rpm`. You can of course also install the zipfiles and install those.

From the system repositories you'll need the 'libaio1' package.

If you want to use the 'sqlplus' command line tool for manual testing you also have to install the package `oracle-instantclient12.1-sqlplus-12.1.0.1.0-1.i386.rpm`. If you want a history in sqlplus it's highly recommended that you install a command line history wrapper like for instance 'rlwrap'.

ArchLinux

You need the two packages 'oracle-instantclient-basic' and 'oracle-instantclient-sdk'.

Have a look at https://wiki.archlinux.org/index.php/Oracle_client on how to install the Oracle packages. Basically you have two options: either you use the 'oracle' pacman repository or you download the Oracle packages by hand and run the build scripts from AUR.

If you want to use the 'sqlplus' command line tool for manual testing you also have to install the package 'oracle-instantclient-sqlplus'. If you want a history in sqlplus it's highly recommended that you install a command line history wrapper like for instance 'rlwrap'.

Slackware

Simply download the zipfiles and install them to a directory, let's say '/opt/oracle/instantclient_12_1':

```
mkdir -p /opt/oracle
cd /opt/oracle
unzip instantclient-basic-linux.x64-12.1.0.1.0.zip
unzip instantclient-sdk-linux.x64-12.1.0.1.0.zip
```

Add the following line to `/etc/ld.so.conf` and reload the cached shared libraries:

```
echo "/opt/oracle/instantclient_12_1" >> /etc/ld.so.conf
ldconfig
```

Call 'make' with:

```
make WITH_ORACLE=1 ORACLE_DIR=/opt/oracle/instantclient_12_1
```

FreeBSD

There are no plans for an Oracle module on FreeBSD.

NetBSD

There are no plans for an Oracle module on NetBSD.

OpenIndiana 151a8

There are no plans for an Oracle module on OpenIndiana.

Solaris 10

For building the Oracle database module you need the two packages 'oracle-instantclient-basic', 'oracle-instantclient-sdk' (both 11.2 and 12.1 are ok, 12.1 needs a higher patchlevel of the SUNW C library though).

Unpack the ZIPs for instance to '/opt/oracle/instantclient_11_2' and build set ORACLE_DIR accordingly (together with WITH_ORACLE=1).

1.2.10. XML filtering support with libxml2 and libxslt

Wolframe can use libxml2 and libxslt (http://xmlsoft.org/) for filtering and the conversion of XML data.

You can build only filtering with libxml2. But if you enable libxslt filtering you also have to enable libxml2 filtering.

You enable the building of a loadable libxml2/libxslt filtering module with:

```
make WITH_LIBXML2=1 WITH_LIBXSLT=1
```

The location of those two libraries can be overloaded with:

```
make WITH_LIBXML2=1 WITH_LIBXSLT=1 \
 LIBXML2_DIR=/usr/local/libxml2-2.9.1 \
 LIBXSLT_DIR=/usr/local/libxslt-1.1.28
```

You can also override all compilation and linking flags of libxml2 and libxslt separately:

```
make WITH_LIBXML2=1 WITH_LIBXSLT=1 \
LIBXML2_INCLUDE_DIR=/usr/local/libxml2-2.9.1/include \
LIBXML2_LIB_DIR=/usr/local/libxml2-2.9.1/lib \
LIBXML2_LIBS=-lxml2 \
LIBXSLT_INCLUDE_DIR=/usr/local/libxslt-1.1.28/include \
LIBXSLT_LIB_DIR=/usr/local/libxslt-1.1.28/lib \
LIBXSLT_LIBS=-lxslt
```

RedHat/Centos/Scientific Linux 5 and similar Linux distributions

The official libxml2 and libxslt package is too old, compile your own versions. Make sure your own libxslt version uses the libxml2 version you compiled and not the system one!

If you don't need working iconv support for non-UTF8 character sets you may also try to use the provided packages 'libxml2-devel' and 'libxslt-devel' but we cannot recommend this.

RedHat/Centos/Scientific Linux 6, Fedora and similar Linux distributions

You need the 'libxml2-devel' and 'libxslt-devel' packages.

Debian, Ubuntu and similar Linux distributions

You need the 'libxml2-dev' and 'libxslt1-dev ' packages.

openSUSE, SLES and similar Linux distributions

You need the 'libxml2-devel' and 'libxslt-devel' packages.

ArchLinux

You need the 'libxml2' and 'libxslt' packages.

Slackware

You need the 'libxml2' and the 'libxslt' packages. Both packages are part of the 'l' package series.

FreeBSD

You need the 'libxml2' and 'libxslt' packages.

NetBSD

You need the 'libxml2' and 'libxslt' packages.

OpenIndiana 151a8

You need the 'CSWlibxml2-dev' and 'CSWlibxslt-dev' packages.

Solaris 10

Both standard packages 'SUNWlxml' and 'SUNWlxsl' are too old, we use the two CSW packages 'CSWlibxml2-dev' and 'CSWlibxslt-dev'.

1.2.11. XML filtering support with Textwolf

Wolframe can use Textwolf (http://textwolf.net) for filtering and the conversion of XML data.

The textwolf library is embedded in the subdirectory `3rdParty/textwolf`.

You enable the building of a loadable Textfolw filtering module with:

```
make WITH_TEXTWOLF=1
```

Note: If you plan to run tests when building the Wolframe you should enable Textwolf as many tests rely on it's presence.

1.2.12. JSON filtering support with cJSON

Wolframe can use cJSON (http://sourceforge.net/projects/cjson/) for filtering and the conversion of JSON data.

The cjson library is embedded in the subdirectory `3rdParty/libcjson`.

You enable the building of a loadable cJSON filtering module with:

```
make WITH_CJSON=1
```

1.2.13. Scripting support with Lua

Wolframe can be scripted with Lua (http://www.lua.org).

The Lua interpreter is embedded in the subdirectory `3rdParty/lua`.

You enable the building of a loadable Lua scripting module with:

```
make WITH_LUA=1
```

1.2.14. Scripting support with Python

Wolframe can be scripted with Python (https://www.python.org).

The module supports only version 3 of the Python interpreter, version 2 is not supported.

You enable the building of a loadable Python scripting module with:

```
make WITH_PYTHON=1
```

The location of the Python library can be overloaded with:

```
make WITH_PYTHON=1 \
 PYTHON_DIR=/usr/local/Python-3.3.5
```

You can also override all compilation and linking flags of the Python library separately:

```
make WITH_PYTHON=1 \
 PYTHON_CFLAGS=-I/usr/include/python3.3m -I/usr/include/python3.3m \
 PYTHON_LDFLAGS=-lpthread -Xlink -export-dynamic \
PYTHON_LIBS=-lpython3.3m
```

Normally you should not change those flags by hand and rely on the results of the 'python-config' script.

RedHat/Centos/Scientific Linux 5 and 6 and similar Linux distributions

There are no official Python packages for version 3 of Python. Build your own version of Python. Make sure the location of 'python3-config' is in your path.

Fedora and similar Linux distributions

You need the 'python3-devel' package.

Debian, Ubuntu and similar Linux distributions

You need the 'python3-dev' package.

openSUSE, SLES and similar Linux distributions

You need the 'python3-devel' package.

ArchLinux

You need the 'python' package.

Slackware

On Slackware you have to build your own version of Python with:

```
./configure --enable-shared
make
make install
```

Alternatively you can of course also build the 'python3' package with the help of SlackBuilds.

FreeBSD

Build the BSD ports package for 'python33'. There is no binary Python 3 package.

NetBSD

You need the 'python33' package.

OpenIndiana 151a8

We cannot use 'CSWpython31-dev' because it's build with the Forte compiler.

We build our own Python 3 with:

```
./configure --prefix=/opt/csw/python-3.3.2/ --enable-shared
gmake
gmake install
```

Solaris 10

We cannot use 'CSWpython31-dev' because it's build with the Forte compiler.

We build our own Python 3 with:

```
./configure --prefix=/opt/csw/python-3.3.2/ --enable-shared
gmake
gmake install
```

1.2.15. Printing support with libhpdf

Wolframe can print with libhpdf (http://libharu.org/, also called libharu).

You enable the building of a loadable libhpdf printing module with:

```
make WITH_SYSTEM_LIBHPDF=1
```

You can also link against the embedded version of libhpdf in '3rdParty/libhpdf' instead of the one of the Linux distribution:

```
make WITH_LOCAL_LIBHPDF=1
```

The location of the libhpdf library can be overloaded with:

```
make WITH_SYSTEM_LIBPHDF=1 \
 LIBHPDF_DIR=/usr/local/libharu-2.2.1
```

You can also override all compilation and linking flags of the libhpdf library separately:

```
make WITH_SYSTEM_LIBPHDF=1 \
 LIBHPDF_INCLUDE_DIR=/usr/local/libharu-2.2.1/include \
 LIBHPDF_LIB_DIR=/usr/local/libharu-2.2.1/lib \
 LIBHPDF_LIBS=-lhpdf
```

Though most Linux distributions have a 'libhpdf' package we recommend to use the embedded version 2.3.0RC2 as this version contains many patches.

RedHat/Centos/Scientific Linux, Fedora and similar Linux distributions

You need the 'zlib-devel' and 'libpng-devel' packages to build libhpdf.

On Fedora you can also try to use the 'libhpdf-devel' package.

Debian, Ubuntu and similar Linux distributions

You need the 'zlib1g-dev' and 'libpng12-dev' packages to build libhpdf.

You can also try to use the 'libhpdf-devel' package.

openSUSE, SLES and similar Linux distributions

You need the 'zlib-devel' and 'libpng12-devel' or 'libpng15-devel' packages to build libhpdf.

You can also try to use the 'libhpdf-devel' package.

ArchLinux

You need the 'zlib' and 'libpng' packages to build libhpdf.

You can also try to use the 'libharu' package.

Slackware

On Slackware you have to build your own version of libhpdf. You need the 'zlib' and 'libpng' packages.

Both packages are part of the 'l' package series.

FreeBSD

Build the embedded version of libhpdf with 'WITH_LOCAL_LIBHPDF=1'.

You need the 'png' package for this.

NetBSD

Build the embedded version of libhpdf with 'WITH_LOCAL_LIBHPDF=1'.

You need the 'png' and the 'zlib' packages for this.

OpenIndiana 151a8

Build the embedded version of libhpdf with 'WITH_LOCAL_LIBHPDF=1'.

You need the 'CSWlibz-dev' package for this.

Solaris 10

Build the embedded version of libhpdf with 'WITH_LOCAL_LIBHPDF=1'.

You need the 'CSWlibz-dev' package for this.

'SUNWzlib' is missing 64-bit support so don't use it!

1.2.16. Image processing with FreeImage

Wolframe can manipulate various image formats with the help of the FreeImage project (http://freeimage.sourceforge.net).

You enable the building of a loadable FreeImage processing module with:

```
make WITH_SYSTEM_FREEIMAGE=1
```

You can also link against the embedded version of FreeImage in '3rdParty/freeimage' instead of the one of the Linux distribution:

```
make WITH_LOCAL_FREEIMAGE=1
```

The location of the FreeImage package can be overloaded with:

```
make WITH_SYSTEM_FREEIMAGE=1 \
 FREEIMAGE_DIR=/usr/local/FreeImage-3.15.4
```

You can also override all compilation and linking flags of the FreeImage package separately:

```
make WITH_SYSTEM_FREEIMAGE=1 \
 FREEIMAGE_INCLUDE_DIR=/usr/local/FreeImage-3.15.4/include \
 FREEIMAGE_LIB_DIR=/usr/local/FreeImage-3.15.4/lib \
 FREEIMAGE_LIBS=-lfreeimage \
 FREEIMAGEPLUS_INCLUDE_DIR=/usr/local/FreeImage-3.15.4/include \
 FREEIMAGEPLUS_LIB_DIR=/usr/local/FreeImage-3.15.4/lib \
 FREEIMAGEPLUS_LIBS=-lfreeimageplus
```

Though there are FreeImage packages on most Linux distributions you may still want to use the locally embedded version.

RedHat/Centos/Scientific Linux and similar Linux distributions

There are FreeImage packages, but they are usually quite old. Better build you own version if FreeImage.

You need the 'zlib-devel' and 'libpng-devel' packages to build FreeImage.

Fedora and similar Linux distributions

You need the 'zlib-devel' and 'libpng-devel' packages to build FreeImage.

You may also try to use 'freeimage-devel' package.

Debian, Ubuntu and similar Linux distributions

You need the 'zlib1g-dev' and 'libpng12-dev' packages to build FreeImage.

You can also try to use the 'libfreeimage-devel' package.

openSUSE, SLES and similar Linux distributions

You need the 'zlib-devel' and 'libpng12-devel' or 'libpng15-devel' packages to build FreeImage.

You can also try to use the 'freeimage-devel' package.

ArchLinux

You need the 'freeimage' package.

Slackware

On Slackware you have to build your own version of FreeImage. You need the 'zlib' and 'libpng' packages. Both packages are part of the 'l' package series.

Alternatively you can of course also build the 'FreeImage' package with the help of SlackBuilds.

FreeBSD

Build the embedded version of FreeImage with 'WITH_LOCAL_FREEIMAGE=1'.

You need the 'png' package for this.

There is a FreeImage port but it doesn't build the libfreeimageplus library we need.

Note: FreeImage doesn't build on 32-bit currently because gcc doesn't support some 64-bit constants on FreeBSD.

NetBSD

Build the embedded version of FreeImage with 'WITH_LOCAL_FREEIMAGE=1'. You need the 'png' and the 'zlib' packages for this.

OpenIndiana 151a8

Build the embedded version of FreeImage with 'WITH_LOCAL_FREEIMAGE=1'.

You need the 'CSWlibz-dev' package for this.

Solaris 10

Build the embedded version of FreeImage with 'WITH_LOCAL_FREEIMAGE=1'.

You need the 'CSWlibz-dev' package for this.

'SUNWzlib' is missing 64-bit support so don't use it!

1.2.17. zlib and libpng

FreeImage and libhpdf need the zlib and libpng libraries.

The location of the zlib and libpng package can be overloaded with:

```
make \
 LIBZ_DIR=/usr/local/zlib-1.2.8 \
 LIBPNG_DIR=/usr/local/libpng-1.6.10
```

You can also override all compilation and linking flags of the zlib and libpng packages separately:

```
make \
 LIBZ_INCLUDE_DIR=/usr/local/zlib-1.2.8/include \
 LIBZ_LIB_DIR=/usr/local/zlib-1.2.8/lib \
 LIBZ_LIBS=-lz \
 LIBPNG_INCLUDE_DIR=/usr/local/libpng-1.6.10/include \
 LIBPNG_LIB_DIR=/usr/local/libpng-1.6.10/libs \
 LIBPNG_LIBS=-lpng
```

1.2.18. Support for ICU

Wolframe can use the International Components for Unicode (ICU, http://site.icu-project.org [http://site.icu-project.org/]) library for text normalization and conversion.

For this to work you need the ICU library itself (ICU4C, at least version 3.6) and the 'boost-locale' library has to have the ICU backend enabled. This is not the case in all Linux distributions.

Note: The Wolframe server doesn't depend directly on the ICU library, only the ICU normalization module does!

You enable the building of a loadable ICU normalization module with:

```
make WITH_ICU=1
```

The location of the ICU library can be overloaded with:

```
make WITH_ICU=1 \
 ICU_DIR=/usr/local/icu4c-52_1
```

You can also override all compilation and linking flags of the ICU library separately:

```
make WITH_ICU=1 \
 ICU_INCLUDE_DIR=/usr/local/icu4c-52_1/include \
 ICU_LIB_DIR=/usr/local/icu4c-52_1/lib \
 ICU_LIBS=-licuuc -licudata -licui18n
```

RedHat/Centos/Scientific Linux, Fedora and similar Linux distributions

Boost is too old, build your own Boost locale and ICU support.

Fedora and similar Linux distributions

You need the 'boost-devel' package. The official Boost packages have support for Boost locale and the ICU backend.

Debian, Ubuntu and similar Linux distributions

Debian 6

The official Boost packages are too old, build your own Boost locale and ICU support.

Debian 7

You need 'libboost-locale-dev' package.

Ubuntu 10.04.1 LTS, Ubuntu 12.04

The official Boost packages are not recent enough. Build your own Boost version with ICU support here.

Ubuntu 12.10

You need 'libboost-locale-dev' package.

Ubuntu 13.04

The official Boost packages have problems. Build your own Boost version with ICU support here.

Ubuntu 13.10

You need 'libboost-locale-dev' package.

openSUSE, SLES and similar Linux distributions

OpenSuSE 12.3, 13.1

You need the 'boost-devel' package. The official Boost packages have support for Boost locale and the ICU backend.

SLES 11 SP1, SP2 and SP3

The official Boost packages are not recent enough and lack ICU support. Build your own Boost with ICU support here.

ArchLinux

You need the 'boost-libs' package. The official Boost package have support for Boost locale and the ICU backend.

Slackware

The official Boost package contains support for boost-locale and the ICU backend. This package is part of the 'l' package series.

FreeBSD

The official Boost packages don't contain a boost-locale with ICU backend.

Build Boost in this case with the patched from `packaging/patches/FreeBSD` applied.

You also need the 'icu' package in this case.

NetBSD

Note: The Boost locale and ICU support is currently broken, see also https://github.com/Wolframe/Wolframe/issues/59.

OpenIndiana 151a8

You cannot use the 'SUNWicud/SUNWicu' and 'CSWlibicu_dev' packages as they are both linked with the Forte C++ compiler. You have to compile your own version compiled with the gcc compiler from CSW:

```
gtar zxf icu4c-51_2-src.tgz
```

apply the solaris XOPEN patch (`packaging/patches/Solaris/icu4c-1.51.2/icu_source_common_uposixdefs_h.patch`), then build ICU with:

```
cd source/icu
./runConfigureICU Solaris/GCC --prefix=/opt/csw/icu4c-51.2
gmake
gmake install
```

Then build Boost as follows:

First apply all patches found in `packaging/patches/Solaris/1.54.0`.

Then build boost with:

```
./bootstrap.sh --prefix=/opt/csw/boost-1.54.0 \
 --with-icu=/opt/csw/icu4c-51.2 \
 --with-libraries=thread,filesystem,system,program_options,date_time,regex,loca
./b2 -a -sICU_PATH=/opt/csw/icu4c-51.2 -d2 install
```

Note: The only tested version for now is version 1.54.0! Other versions of Boost may work or not..

Solaris 10

We don't use the CSW boost packages.

You cannot use the 'SUNWicud/SUNWicu' and 'CSWlibicu_dev' packages as they are both linked with the Forte C++ compiler. You have to compile your own version compiled with the gcc compiler from CSW:

```
gtar zxf icu4c-49_1_2-src.tgz
```

apply the solaris icu_source_configure patch (`packaging/patches/Solaris/icu4c-1.49.2/icu_source_configure.patch`), then build ICU with:

```
cd source/icu
./runConfigureICU Solaris/GCC --prefix=/opt/csw/icu4c-49.1.2
gmake
gmake install
```

Apply the following patch to `boost/cstdint.ppp` for boost (https://svn.boost.org/trac/boost/ticket/6158, see also `packaging/patches/Solaris/boost-1.48.0` for patch files):

```
...
namespace boost
{

#if defined(sun) || defined(__sun)
 typedef signed char int8_t;
#else
 using ::int8_t;
#endif
..
```

Patch the correct architecture (V8 is not really supported, but V8 is also very old) in `tools/build/v2/user-config.jam`:

```
using gcc : 4.6.3 : g++ : <compileflags>-mcpu=v9 ;
```

Apply all Boost compilation patches from 'packaging/patches/Solaris/boost-1.48.0' now.

Then build boost with:

```
./bootstrap.sh --prefix=/opt/csw/boost-1.48.0 \
 --with-libraries=thread,filesystem,system,program_options,date_time,regex,loca
 --with-icu=/opt/csw/icu4c-49.1.2
./b2 -a -sICU_PATH=/opt/csw/icu4c-49.1.2 -d2 install
```

Note: The only tested version for now is version 1.48.0! Other versions of Boost may work or not:

Do not use boost 1.49.0, it has a missing function fchmodat, see http://lists.boost.org/boost-build/2012/02/25680.php causing building of libboost_filesystem to fail!

Boost 1.50.0 and 1.51.0 have never been tested with Wolframe, so don't use those!

Support for 1.52.0 and later is not guaranteed.

1.2.19. Authentication support with PAM

Wolframe can authenticate users with PAM.

You enable the building of a loadable PAM authentication module with:

```
make WITH_PAM=1
```

The location of the PAM library can be overloaded with:

```
make WITH_PAM=1 \
 PAM_DIR=/usr/local/pam-1.1.8
```

You can also override all compilation and linking flags of the PAM library separately:

```
make WITH_PAM=1 \
 PAM_INCLUDE_DIR=/usr/local/pam-1.1.8/include \
 PAM_LIB_DIR=/usr/local/pam-1.1.8/lib \
 PAM_LIBS=-lpam
```

RedHat/Centos/Scientific Linux, Fedora and similar Linux distributions

You need the 'pam-devel' package.

Debian, Ubuntu and similar Linux distributions

You need the 'libpam0g-dev' package.

openSUSE, SLES and similar Linux distributions

You need the 'pam-devel' package.

ArchLinux

You need the 'pam' package.

Slackware

On Slackware there is no official PAM package. You have to build 'linux-pam' on your own.

FreeBSD

Note: We currently don't support PAM on FreeBSD.

NetBSD

Note: We currently don't support PAM on NetBSD.

OpenIndiana 151a8

PAM support is available out of the box just specify 'WITH_PAM=1'.

Solaris 10

PAM support is available out of the box just specify 'WITH_PAM=1'.

1.2.20. Authentication support with SASL

Wolframe can authenticate users with the Cyrus SASL library (http://cyrusimap.org/ [http://cyrusimap.org]).

Note: GNU SASL is currently not supported.

You enable the building of a loadable SASL authentication module with:

```
make WITH_SASL=1
```

The location of the Cyrus SASL library can be overloaded with:

```
make WITH_SASL=1 \
 SASL_DIR=/usr/local/cyrus-sasl-2.1.26
```

You can also override all compilation and linking flags of the Cyrus SASL library separately:

```
make WITH_SASL=1 \
 SASL_INCLUDE_DIR=/usr/local/cyrus-sasl-2.1.26/include \
 SASL_LIB_DIR=/usr/local/cyrus-sasl-2.1.26/lib \
 SASL_LIBS=-lsasl2
```

RedHat/Centos/Scientific Linux, Fedora and similar Linux distributions

You need the 'cyrus-sasl-devel' package.

Debian, Ubuntu and similar Linux distributions

You need the 'libsasl2-dev' package.

For running the SASL tests you also need the 'sasl2-bin' package.

openSUSE, SLES and similar Linux distributions

You need the 'cyrus-sasl-devel' package.

ArchLinux

You need the 'libsasl' package.

Slackware

You need the 'cyrus-sasl' package. This package is part of the 'n' package series.

FreeBSD

You need the 'cyrus-sasl' package.

NetBSD

You need the 'cyrus-sasl' package.

OpenIndiana 151a8

You need the 'CSWsasl' and 'CSWsasl-dev' packages.

Solaris 10

You need the 'CSWsasl' and 'CSWsasl-dev' packages.

1.2.21. Testing Wolframe

Wolframe has tests written in Google gtest (https://code.google.com/p/googletest/).

Tests are run with:

```
make test
```

Some tests run for a long time (regression and stress tests). They are not run per default when calling 'make test', but you have to call:

```
make longtest
```

Sometimes you only want to build the test programs but not to run them (for instance when cross-compiling). Then you can set the 'RUN_TESTS' variable as follows:

```
make test RUN_TESTS=0
```

1.2.22. Testing with Expect

Some more complex tests are written with Expect (http://expect.sourceforge.net/).

You enable testing with Expect with:

```
make WITH_EXPECT=1
```

The location of the Expect interpreter can be overloaded with:

```
make WITH_EXPECT=1 \
 EXPECT=/usr/local/bin/expect
```

RedHat/Centos/Scientific Linux, Fedora and similar Linux distributions

You need the 'expect' and the 'telnet' packages.

Debian, Ubuntu and similar Linux distributions

You need the 'expect' and the 'telnet' packages.

openSUSE, SLES and similar Linux distributions

You need the 'expect' and the 'telnet' packages.

ArchLinux

You need the 'expect' and 'inetutils' packages.

Slackware

You need the 'expect' and 'telnet' packages. Those packages are part of the 'tcl' respectively the 'n' package series.

FreeBSD

You need the 'expect' package.

NetBSD

You need the 'tcl-expect' package.

OpenIndiana 151a8

You need the 'CSWexpect' package.

Solaris 10

You need the 'CSWexpect' package.

1.2.23. Building the documentation

The documentation including the man pages is written using DocBook (http://www.docbook.org).

Developer documentation is generated with Doxygen (http://www.doxygen.org).

All documentation is built in the 'docs' subdirectory:

```
cd docs
make doc
```

Note: The various tools are not able to produce the same results on all platforms. Your experience in the quality of the generated artifacts may vary. Generally, the newer the tools, the better.

The validity of the XML of the documenation can be checked with:

```
cd docs
make check
```

RedHat/Centos/Scientific Linux and similar Linux distributions

You need the 'libxslt', 'doxygen' and 'docbook-style-xsl' packages (from EPEL).

When generating PDFs you need the 'fop' package.

When rebuilding the SVG images of the documentation you also need 'dia'.

Fedora and similar Linux distributions

You need the 'libxslt', 'doxygen' and 'docbook-style-xsl' packages.

When generating PDFs you need the 'fop' package.

When rebuilding the SVG images of the documentation you also need 'dia'.

Debian, Ubuntu and similar Linux distributions

You need the 'xsltproc', 'doxygen' and 'docbook-xsl' packages.

When generating PDFs you need the 'fop' package.

When rebuilding the SVG images of the documentation you also need 'dia'.

For checking the validity of various XML files you need 'libxml2-utils' (for xmllint).

openSUSE, SLES and similar Linux distributions

You need the 'libxslt', 'doxygen' and 'docbook-style-xsl' packages.

When generating PDFs you need the 'fop' package.

When rebuilding the SVG images of the documentation you also need 'dia'.

ArchLinux

You need the 'xsltproc', 'doxygen' and 'docbook-xsl' packages.

When generating PDFs you need the 'fop' package. Newest versions run only with the SVN version of 'java-xmlgraphics-commons'. Install the package 'java-xmlgraphics-commons-svn' from the AUR.

When rebuilding the SVG images of the documentation you also need 'dia'.

Slackware

You need the 'libxslt' and 'doxygen' packages. Those packages part of the 'l', 'd' package series. DocBook you have to install on your own.

When generating PDFs you have to install 'fop' on your own.

When rebuilding the SVG images of the documentation you also need 'dia' which you will have to build on your own. Alternatively you can of course also build the 'dia' package with the help of SlackBuilds.

FreeBSD

We never tried to build the documentation on FreeBSD so far.

NetBSD

We never tried to build the documentation on NetBSD so far.

1.2.24. Installation

The makefiles provide a 'install' and an 'uninstall' target to install and uninstall the software.

The 'DESTDIR' and 'prefix' parameters are useful for packagers to reroute the destination of the installation.

For instance:

```
make DESTDIR=/var/tmp prefix=/usr/local/wolframe-0.0.1 install
```

installs the software in:

```
/var/tmp/usr/local/wolframe-0.0.1/sbin/wolframed
/var/tmp/usr/local/wolframe-0.0.1/etc/wolframe/wolframe.conf
...
```

The 'DEFAULT_MODULE_LOAD_DIR' parameter can be used by packagers to set the load directory for loadable modules. For instance a Redhat SPEC file will contain a line like:

```
make DEFAULT_MODULE_LOAD_DIR=%{_libdir}/wolframe/modules
```

1.2.25. Manual dependency generation

Usually dependencies are automatically recomputed and stored in files with extension '.d'.

On some platforms and with some older versions of GNU make you can run into problems, especially if you build the software in parallel. For this case you can force the computation of depencies in a special make step as follows:

```
make depend
make -j 4
```

1.2.26. Creating source tarballs

Wolframe supports the standard targets 'dist', 'dist-Z', 'dist-gz' and 'dist-bz2' to create a tarball containg all the necessary sources.

Chapter 2. Installation via binary packages

This section describes how to install the Wolframe application via packages on various operating systems.

2.1. Linux distributions

Linux distributions are currently built on the Open Build Server (http://openbuildservice.org) and on a bunch of virtual machines.

The resulting packages and the repository metadata is hosted on Sourceforge (http://sourceforge.net).

The packages are always build with the default system compiler, which is currently GNU gcc.

Packages for proprietary software (like the Oracle database module) have to be built manually, they can not be distributed as binary packages due to license problems.

2.1.1. RedHat, Fedora, CentOS, Scientific Linux and similar Linux distributions

Available packages

- wolframe-0.0.1.rpm: contains the Wolframe core server with minimal 3rd party software requirements
- wolframe-sqlite3-0.0.1.rpm: the database module for Sqlite3 databases
- wolframe-postgresql-0.0.1.rpm: the database module for PostgreSQL databases
- wolframe-libxml2-0.0.1.rpm: filtering module for XML and XSLT (using libxml2/libxslt)
- wolframe-textwolf-0.0.1.rpm: filtering module for XML (using textwolf)
- wolframe-cjson-0.0.1.rpm: filtering module for JSON (using cJSON)
- wolframe-pam-0.0.1.rpm: authentication module for PAM
- wolframe-sasl-0.0.1.rpm: authentication module for SASL
- wolframe-python-0.0.1.rpm: language bindings for Python
- wolframe-lua-0.0.1.rpm: language bindings for Lua
- wolframe-libhpdf-0.0.1.rpm: printing module using libhpdf
- wolframe-freeimage-0.0.1.rpm: image manipuation module using FreeImage
- wolframe-libclient-0.0.1.rpm: C/C++ client library
- wolframe-client-0.0.1.rpm: command line tool
- wolfclient-0.0.3.rpm: Wolframe graphical frontend

Prerequisites

Install binary packages manually

Installing the packages via repositories is usually the prefered way.

Install from repository

First install the repository file for the corresponding distribution (as example we choose Centos 6):

```
cd /etc/yum.repos.d
wget http://sourceforge.net/projects/wolframe/files/repositories/CentOS_CentOS
```

You can list all available Wolframe packages with:

```
yum search wolframe
```

You install the main Wolframe package with:

```
yum install wolframe
```

You have to accept the signing key:

```
Retrieving key from http://sourceforge.net/projects/wolframe/files/repositorie
Importing GPG key 0x9D404026:
Userid: "home:wolframe_user OBS Project <home:wolframe_user@build.opensuse.org
From  : http://sourceforge.net/projects/wolframe/files/repositories/CentOS_Cen
Is this ok [y/N]: y
```

You can start the service with:

```
service wolframed start
```

respectively

```
systemctl start wolframed
```

on newer Fedora systems.

2.1.2. Debian, Ubuntu and similar Linux distributions

Add a new repository file `/etc/apt/sources.list.d/wolframe.list` which contains:

```
deb http://sourceforge.net/projects/wolframe/files/repositories/xUbuntu_13.10/
```

(as example we choose Ubuntu 13.10).

Download the signing key:

```
wget http://sourceforge.net/projects/wolframe/files/repositories/xUbuntu_13.10/
```

Verify that the key then add it with:

```
apt-key add - < Release.key
```

Update the repository with:

```
apt-get update
```

You can list all available Wolframe packages with:

```
apt-cache search wolframe
```

You install the main Wolframe package with:

```
apt-get install wolframe
```

To start the Wolframe service you have to edit the file `/etc/default/wolframe` and enable the wolframe daemon there:

```
RUN=yes
```

You can start the service now with:

```
service wolframed start
```

2.1.3. openSUSE, SLES and similar Linux distributions

First we add the Wolframe repository for the corresponding distribution (as example we choose OpenSUSE 13.1):

```
zypper addrepo http://sourceforge.net/projects/wolframe/files/repositories/ope
```

You may get the following error:

```
/var/adm/mount/AP_0xmiyYP3/projects/wolframe/files/repositories/openSUSE_13.1/w
Is it a .repo file? See http://en.opensuse.org/Standards/RepoInfo for details.
```

Try to download the repo file by hand and install it by hand:

```
wget http://sourceforge.net/projects/wolframe/files/repositories/openSUSE_13.1
zypper addrepo wolframe.repo
```

Now refresh your repositories with:

```
zypper refresh
```

If you get the following message

```
File 'repomd.xml' from repository 'Wolframe Project (openSUSE_13.1)' is unsign
```

the signing key could not be downloaded from SourceForge. Accept it in this case anyway.

If you get the following message

```
File './repodata/ea7cb8d9a0caa2c3d8977919be124accdf55c6b8952ddee72f1b48f4decb0

Abort, retry, ignore? [a/r/i/? shows all options] (a): u^H
Invalid answer ''. [a/r/i/? shows all options] (a): ?

a - Skip retrieval of the file and abort current operation.
r - Try to retrieve the file again.
i - Skip retrieval of the file and try to continue with the operation without
u - Change current base URI and try retrieving the file again.

[a/r/i/? shows all options] (a): u
```

the Sourceforge redirect didn't work and you have to force the baseURL to be a SourceForge mirror like:

```
New URI: http://freefr.dl.sourceforge.net/project/wolframe/repositories/openSU
```

You can list all available Wolframe packages with:

```
zypper se wolframe
zypper se wolfclient
```

You install the main Wolframe package with:

```
zypper install wolframe
```

You can start the service with:

```
service wolframed start
```

respectively

```
systemctl start wolframed
```

on newer openSUSE systems.

2.1.4. ArchLinux

First add the following section to `/etc/pacman.conf`:

```
[wolframe]
SigLevel = Optional DatabaseRequired
Server = http://sourceforge.net/projects/wolframe/files/repositories/Arch_Extra
```

Fetch and verify the sigining key, import and locally sign the key:

```
wget http://download.opensuse.org/repositories/home:/wolframe_user/xUbuntu_13.1
pacman-key --add Release.key
pacman-key --lsign 9D404026
```

Alternatively you can also disable the verification of the signature of the database by removing 'DatabaseRequired' from the 'SigLevel' option.

Update the repository data with:

```
pacman -Syy
```

You can list all available Wolframe packages with:

```
pacman -Sl wolframe
```

You install the main Wolframe package with:

```
pacman -S wolframe
```

You can start the service with:

```
systemctl start wolframed
```

2.1.5. Slackware

Download the package file (we choose 64-bit Slackware 14 for example):

```
wget http://sourceforge.net/projects/wolframe/files/wolframe-binaries/0.0.1/Sla
```

You install the Wolframe package with:

```
installpkg wolframe-0.0.1-x86_64.tgz
```

The Slackware packages contain the whole server and the whole client respectively.

You can start the service with:

```
/etc/rc.d/rc.wolframed start
```

2.2. Other Unix systems

2.2.1. FreeBSD

Download the package file (we choose 64-bit FreeBSD 9 for example):

```
wget http://sourceforge.net/projects/wolframe/files/wolframe-binaries/0.0.1/Fre
```

You install the Wolframe package with:

```
pkg_add wolframe-0.0.1-x86_64.tgz
```

The FreeBSD packages contain the whole server and the whole client respectively.

You can start the service with:

```
/usr/local/etc/rc.d/wolframed onestart
```

To start the Wolframe service at system boot time you have to edit the file `/etc/rc.conf` and enable the wolframe daemon there with:

```
wolframed_enable="YES"
```

You can start the service now with:

```
service wolframed start
```

2.2.2. NetBSD

Download the package file (we choose 64-bit NetBSD 6 for example):

```
wget http://sourceforge.net/projects/wolframe/files/wolframe-binaries/0.0.1/Net
```

You install the Wolframe package with:

```
pkg_add wolframe-0.0.1-x86_64.tgz
```

The NetBSD packages contain the whole server and the whole client respectively.

You can start the service with:

```
/usr/pkg/share/examples/rc.d/wolframed onestart
```

To start the Wolframe service at system boot time you have to edit the file `/etc/rc.conf` and enable the wolframe daemon there with:

```
wolframed=YES
```

Copy the example startup script to the final place:

```
cp /usr/pkg/share/examples/rc.d/wolframed /etc/rc.d/
```

You can start the service now with:

```
/etc/rc.d/wolframed
```