ROSE Installation Guide:

May 21, 2018

0.1 ROSE Installation

See http://rosecompiler.org/ROSE_HTML_Reference/installation.html.

0.1.1 Software/Hardware Requirements and Options

See http://rosecompiler.org/ROSE_HTML_Reference/installation.html

Required Hardware/Operating System

See http://rosecompiler.org/ROSE_HTML_Reference/installation_hardware.html

Software Requirements

See http://rosecompiler.org/ROSE_HTML_Reference/installation_prerequisites.html

0.1.2 Building BOOST

See http://rosecompiler.org/ROSE_HTML_Reference/installing_boost.html

0.1.3 Using Insure++

It is possible to configure ROSE to use insure++ for static and dynamic analysis of the ROSE source code as part of development of ROSE or ROSE based tools. This is a feature that is experimental, and currently not working well (because our older version of insure++ (version 7.1.6) is failing to compile the Linux header files used in ROSE). The configure option to turn on the use of insure++ is: *–enable-insure*. This is all that is required if `insure` is in your path (at LLNL run: `source /usr/apps/insure++/default/setup.csh` for csh and `source /usr/apps/insure++/default/setup.sh` for bash). *Don't use Insure++ until you have some experience with ROSE development, it makes everything more complex.*.

0.1.4 Building ROSE From a Distribution (ROSE-0.9.10.27.tar.gz)

ROSE is no longer distributed as a tarball. Use the Git repositories instead.

See http://rosecompiler.org/ROSE_HTML_Reference/installation.html

0.1.5 Building ROSE from a Development Version (from SVN or GIT)

See http://rosecompiler.org/ROSE_HTML_Reference/installation.html

0.1.6 TroubleShooting the ROSE Installation

There are a number of famous ways to screw up your installation of ROSE.

1. Message: `configure: error: Could not link against boost_filesystem-gcc41-mt`
   This message from running the `configure` command in ROSE (an initial step in building ROSE) indicates that your `LD_LIBRARY_PATH` (environment variable) is not set to to the location of the boost install tree. The ROSE configure scripts (autoconf) will test the linking to specific boost libraries and this is the first dynamic link library that it tests and so it will fail when many other tests on boost succeed because your `LD_LIBRARY_PATH` is finally required and is not properly set.

2. Message: `Making all in libltdl`
   `make[2]: *** No rule to make target 'all'. Stop.`
   Run `glibtoolize --force` to rebuild the libtool support in ROSE for your machine at the top level of the source tree. If that does not work then give up on the libtool that came with the apple dev tools and just build your own libtool in your home directory.

3. **Don't build ROSE in the source tree, it is not tested often, but it should work.**
   Save yourself some trouble and build a separate compile tree. This will also allow you to build a number of different versions of ROSE with different options.

4. Message: **configure: error: Unable to find path to JVM library**
   This message from running the `configure` command in ROSE (an initial step in building ROSE) indicates either that your `LD_LIBRARY_PATH` (environment variable) is not set to to the location of the *libjvm.so* or that your machines java is not one that we support (e.g. non-Sun Java). If you don't require Java (e.g. don't need Fortran support) then consider skipping the java support by using *–without-java* on the configure command line. Alternatively, your `LD_LIBRARY_PATH` should contain the path to the file *libjvm.so*. The likely path is specified in the lines just before the message. The full message will appear as:

   ```
   checking for Java... /usr/lib/jvm/java-1.5.0-ibm.x86_64/bin/../bin/java
   checking for Java JVM include and link options... JavaJREDir  = /usr/lib/jvm/java-1.5.0-ibm-1.5.0.8.x86_64
   JavaHomeDir = /usr/lib/jvm/java-1.5.0-ibm-1.5.0.8.x86_64
   JavaJVMDir  = /usr/lib/jvm/java-1.5.0-ibm-1.5.0.8.x86_64/jre/bin/classic
   configure: error: Unable to find path to JVM library
   ```

5. Previously installed version of Boost library.
   Some machines have a default version of Boost already installed (for example in `/usr/include/boost`). This always the wrong version since the OS installation of Boost lags by several years. ROSE now attempts to detect this and use the

   -isystem g++ option to have the explicitly specified version of boost from the configure command-line be search before the system include directories. This works well where a machine has a previously installed version of Boost, but it will fail when used with SWIG (so don't use `--with-javaport` where a previous system installation of Boost is detected). The ROSE configure scripts will detect the presence of a previously installed version of Boost and issue a warning message to not use `--with-javaport`. Also if no previously installed version of Boost is detected the configuration will report this as well and make clear that it will use the Boost include directory with a `-I` option.

6. libtoolize not available (or old version)
   The problem is that ROSE is calling `libtoolize` or `glibtoolize` and it seems that you don't have it on your machine (called by the build script). You will need it, it is a requirement. The `build` script will run this to build you the required libtool support. Since this happens upsteam of `configure` we don't have a test for it. The clue is the output:

   ```
   ls: cannot access libltdl/*: No such file or directory
   libtoolize: cannot list files in '/usr/share/libtool/libltdl'
   ```

   If you build libtool on your machine and add the installed libtool `bin` directory to you path, then it should work. I often use `libtool-2.2.4.tar` when I have this problem on a new platform.

   Report from use:

   ```
   Reason for the problem: I am not building libtools from source, instead using
   the packages from the Linux distribution repository. On my distribution
   (Ubuntu 8.04, X86_64), the libltdl3 (and libltdl3-dev)  does not come with
   the libtool package. After installing the libtools, I still need to install
   both the libltdl3 and libltdl3-dev package. That is the issues of unable to
   find libltdl folders.
   ```

7. ROSE fails to compile after **svn update**:
   We have seen this problem and had it reported and we don't understand it. It does however disappear after a fresh checkout from SVN into an empty directory. If you figure this out please let us know. Where this has happened to us, we were using svn version 1.4.6, where as our svn repository is more commonly (within development) had work checked in using svn version 1.5.1; since a lot changed from svn version 1.4 to version 1.5, this may be the issue.

   ```
   make[2]: Entering directory '<Your ROSE compile tree path>/src/frontend/SageIII'
     COMPILE preproc.lo
   /home/dquinlan/ROSE/svn-rose/src/frontend/SageIII/preproc.lex: In function 'ROSEAttributesList* getPreprocessorDirectives(std::string)'
   /home/dquinlan/ROSE/svn-rose/src/frontend/SageIII/preproc.lex:961: error: conversion from 'std::_Rb_tree_iterator<std::pair<const std::
   /home/dquinlan/ROSE/svn-rose/src/frontend/SageIII/preproc.lex:963: error: no match for 'operator!=' in 'iItr != (&mapFilenameToAttribut
   /home/dquinlan/local/gcc/3.4.3/bin/../lib/gcc/i686-pc-linux-gnu/3.4.3/../../../../include/c++/3.4.3/bits/stl_tree.h:213: note: candidat
   /home/dquinlan/ROSE/svn-rose/src/ROSETTA/Grammar/Node.code:50: note:      bool operator!=(const rose_rva_t&, const rose_rva_t&)
   /home/dquinlan/ROSE/svn-rose/src/ROSETTA/Grammar/Support.code:3984: note: bool operator!=(const Sg_File_Info&, const Sg_File_Info&)
   make[2]: *** [preproc.lo] Error 1
   make[2]: Leaving directory '<Your ROSE compile tree path>/src/frontend/SageIII'
   make[1]: *** [all-recursive] Error 1
   make[1]: Leaving directory '<Your ROSE compile tree path>/src/frontend/SageIII'
   make: *** [all] Error 2
   ```

0.1.7 ROSE Configure Options

For GNU autotools configuration options and documentation, run "configure –help".
Output of `configure --help` is detailed in Figures 0.1.7 (Part 1) and 0.1.7 (Part 2):

0.1.8 Running *GNU Make* in Parallel

See http://rosecompiler.org/ROSE_HTML_Reference/installation.html

0.1.9 Installing ROSE

See http://rosecompiler.org/ROSE_HTML_Reference/installation.html

0.1.10 Getting Help

See http://rosecompiler.org/ROSE_HTML_Reference/index.html

0.1.11 ROSE and the NMI Compile Farm

The NSF Middleware Initiative (NMI) has provides us with time on their system to support the robustness of ROSE across multiple platforms. ROSE is not tested on a wide range of platforms (see table 0.1.11). The prerequisites used for each platform (machine and operating system) are generated in the table from the input test descriptions located in the directory `ROSE/scripts/nmiBuildAndTestFarm/build_configs`.

For More information about NMI, see `http://nmi.cs.wisc.edu/`. To see the details of the ROSE nightly tests click on the link: *Run Results* and select the project, *rose compiler*, from the pull down menu.

NMI OS and machine (platform) Prerequisites for ROSE:

NMI OS and machine (platform) Configure Options for ROSE:

0.1.12 Installation Details for Specific Platforms

See http://rosecompiler.org/ROSE_HTML_Reference/installation.html

Mac OS X v10.6, Snow Leopard The non-standard STL hashmap implementation in Mac OS X v10.6 is broken and so ROSE can fail when using this OS. We are working on a fix to use more standard features of STL.

Fedora 11

Some trivia with regard to libtool and building the SVN version of Rose: Fedora (11), at least, has a package, libtool-ltdl-devel that is needed (and separate from the libtool package) to make autoconfiguration work right. The symptom is that the post-configure build enters libltdl, and finds no Makefile and fails immediately. Before running Rose's "./build", make sure the libtool-ltdl-devel RPM is installed.

Intel C++ Compiler

The Intel compiler can run out of space compiling some of the larger files in ROSE. Although not previously seen by anyone on the ROSE team, one user has reported that the Intel compiler option *-override-limits* was required. As used on the following configure line: */nfs/casc/aleamr/yana-local/rose/build/rose-sourcetree/configure CXX=icpc CC=icc CXXFLAGS=-override-limits –prefix=‘pwd‘ –with-boost=¡path to boost¿* More information is on this option and when to use it is at: *http://software.intel.com/en-us/articles/internal-threshold-was-exceeded.* The problem that this appears to fix is that on some machines the ROSE file AST_FILE_IO.C will fail with the error *memory limit error*, this flag to the Intel compiler will fix this (the mcpcom process goes above 2.5g memory for this file).

0.1.13 Installing ROSE under Windows

Under Windows ROSE uses CMake. This is a project that is currently under development. As of November 2010 we are able to compile and link the src directory. We are also able to run example programs that link against librose and execute the frontend and backend. *However, this is an internal capability and not available externally yet since we don't distribute the Windows generated EDG binaries that would be required. Also the current support for Windows is still incomplete, ROSE does not yet pass its internal tests under Windows.*

Setup

Under Windows we use the following tools for compilation and development:

- Microsoft Visual Studio (9.0) : We currently use the Debug mode
- CMake 2.8.0 : One only needs to specify the source and build directory. All configuration options should be chosen correctly during makefile generation.
- Bison 2.4.1 : Used by ROSE
- Flex 2.5.4 : Used by ROSE
- Git 1.6.5 : We utilize the git shell for debugging and command line operations
- Boost 1.37 : Boost is required by ROSE

In order to have Hudson test ROSE under Windows, we have additional tools set up:

- pegeant.exe : Allows to avoid entering ssh keys when connecting to git repository
- Hudson client : The client can be started through the web page. All that is required is Java to be installed.

Currently under tux270-0 Flex, cmake, boost and Bison are installed under c:/ROSE. In addition, a ROSE test branch for ROSE-tps is installed in order to test ROSE under Windows locally without Hudson. The directory c:/tools contains pegeant.exe and other useful tools like plink.exe and puttygen.exe. See further notes below. Finally required ssh keys are located in c:/putty_keys.

Debugging in VS 8

ROSE compiles now in release mode and debug mode. Debug mode was a challenge before because adding type-information (RTTI) caused the ROSE dll to be too large.

To use VS with debugging information right click on the ROSE.dll project and chose PROPERTIES/C-C++ INFO:

- GENERAL: turn on Debugging Information Format (/Z7) - do not use DB
- Language: Enable Runtime type information
- Linker: Turn off incremental linking (slightly slower but takes too much memory)

To test whether ROSE (src) compiles, links and a test program can run, modify the qualifiedName project in VS:

- General and Language options as above
- Debugging: enter the following under arguments "–help"

This should allow you to run "qualifiedName –help" in debug mode and all options of rose are print out.

Hudson specific

Setting up SSH keys on Windows Server (logged in as hudson-rose):

- Generate the keys:
  - Generate the keys, run "plink -agent tux269" and enter your own login and then password. to login to tux269 and then exit.
- Copy the keys onto tux270-0:
  - Double Click on "My_Computer" icon
  - Goto C:, tools, pageant.exe
  - Double click on pageant.exe
  - Should display a computer with a hat on it in the bottom right corner of the screen.
  - Right click on icon (of computer with a hat on it)
  - click on "Add Key" (this will cause a window to put up called "Select Private Key File"
  - Goto "My Computer TUX270-0", "Local Disk(C:)", "putty_keys"
  - Click on file "id_rsa.ppk" (this will load the private key for use by ssh).
  - Now try to run the git command: git clone ssh://hudson-rose@tux269/usr/casc/overture/ROSE/git/ROSE.git c:/ROSE/hudson/workspace/test-windows/label/windows-server

Now in order to run a job from Hudson under Windows we need to start the client on the Windows side. Go to the hudson webpage and chose "Manage Hudson"/"Manage Nodes"/"tux270" and the start the JNLP agent. On the Windows side a client job is active now and whenever the Windows job on Hudson runs the client is activated.

Currently Hudson is configured in a way that when a00-ROSE-from-scratch passes, a downstream is started and a01-ROSE-WINDOWS is started. The a01-ROSE-WINDOWS job will activate the client on tux270-0 and run the Windows test. In addition rose-hdsn-win-1 has been configured as a second Windows test node for Hudson.

Extending ROSE with the Windows SDK

Some functions used in Linux are not available under Windows - but they are available through the Windows SDK. For instance, realpath() in linux is available as PathCanonicalize() under Windows using the Windows SDK. You need to include : #include "Shlwapi.h" and the following library into the project : shlwapi.lib (part of SDK). More information can be found at: http://msdn.microsoft.com/en-us/windows/bb980924.aspx http://en.wikipedia.org/wiki/Microsoft_Windows_SDK

0.1.14 Options for Static vs. Dynamic Linking of Executables

ROSE will by default build dynamically linked executables. ROSE supports both static and dynamic linking. Some developers want the space savings of dynamic linking (also might link faster) and some want the simplicity of static linked executables (since they can be easily moved in a single step). The configure options to support linking options are:

- dynamic linking (default): –**enable-shared**

- dynamic and static linking (builds the static libraries, but executables are by default linked dynamically): –**enable-static**
- static linking only: –**enable-static** –**disable-shared**

0.1.15 Options to Control the Size of ROSE Executables

ROSE by default turns on the compiler's generation of debugging information (symbol tables and dwarf2 debug information). This is done to support development which will nearly always be easier with the symbol tables generated with the debugging options (typically "-g"). However, the volume of code in ROSE will cause a lot of debugging information to be generated (around 150Meg, just for the debugging information in a statically linked executable).

To turn off the generation of debug information in the executables use the configure options: `--with-CXX_DEBUG=no --with-C_DEBUG=no` Any other values will use those explicit value to turn on debugging information in the compilation of ROSE.

The sizes of executables built using dynamic linking are always trivially small, the significant different matters when executables are built using static linking (see section 0.1.14). For a statically linked executable (ROSE-based tool) the size with symbol tables (debugging information) can be 150-200 Meg per executable; this value is for the g++ compiler, other compilers will vary in the sizes of the executables they generate. Turning off the debug information generated by g++ will reduce the size of the same executable to be about 40 Meg. Executables can be stripped of symbol table information further using the `strip` utility (Linux). Executables using ROSE that are stripped will be about 25% smaller (about 30 Meg).

Further work in ROSE could likely reduce the size of ROSE based executables, but it is the judgment of the ROSE team that current work work is sufficient to generate small enough executables. If users have need for smaller executables for ROSE based tools, please contact the ROSE team directly.

`configure --help` Option Output (Part 1)

```
'configure' configures ROSE 0.9.10.27 to adapt to many kinds of systems.

Usage: /export/tmp.rose-mgr/jenkins/edg4x/workspace/release-ROSE-docs/configure [OPTION]... [VAR=VALUE]...

To assign environment variables (e.g., CC, CFLAGS...), specify them as
VAR=VALUE.  See below for descriptions of some of the useful variables.

Defaults for the options are specified in brackets.

Configuration:
  -h, --help              display this help and exit
      --help=short        display options specific to this package
      --help=recursive    display the short help of all the included packages
  -V, --version           display version information and exit
  -q, --quiet, --silent   do not print 'checking ...' messages
      --cache-file=FILE   cache test results in FILE [disabled]
  -C, --config-cache      alias for '--cache-file=config.cache'
  -n, --no-create         do not create output files
      --srcdir=DIR        find the sources in DIR [configure dir or '..']

Installation directories:
  --prefix=PREFIX         install architecture-independent files in PREFIX
                          [/usr/local]
  --exec-prefix=EPREFIX   install architecture-dependent files in EPREFIX
                          [PREFIX]

By default, 'make install' will install all the files in
'/usr/local/bin', '/usr/local/lib' etc.  You can specify
an installation prefix other than '/usr/local' using '--prefix',
for instance '--prefix=$HOME'.

For better control, use the options below.

Fine tuning of the installation directories:
  --bindir=DIR            user executables [EPREFIX/bin]
  --sbindir=DIR           system admin executables [EPREFIX/sbin]
  --libexecdir=DIR        program executables [EPREFIX/libexec]
  --sysconfdir=DIR        read-only single-machine data [PREFIX/etc]
  --sharedstatedir=DIR    modifiable architecture-independent data [PREFIX/com]
  --localstatedir=DIR     modifiable single-machine data [PREFIX/var]
  --libdir=DIR            object code libraries [EPREFIX/lib]
  --includedir=DIR        C header files [PREFIX/include]
  --oldincludedir=DIR     C header files for non-gcc [/usr/include]
  --datarootdir=DIR       read-only arch.-independent data root [PREFIX/share]
  --datadir=DIR           read-only architecture-independent data [DATAROOTDIR]
  --infodir=DIR           info documentation [DATAROOTDIR/info]
  --localedir=DIR         locale-dependent data [DATAROOTDIR/locale]
  --mandir=DIR            man documentation [DATAROOTDIR/man]
  --docdir=DIR            documentation root [DATAROOTDIR/doc/rose]
  --htmldir=DIR           html documentation [DOCDIR]
  --dvidir=DIR            dvi documentation [DOCDIR]
  --pdfdir=DIR            pdf documentation [DOCDIR]
  --psdir=DIR             ps documentation [DOCDIR]

Program names:
  --program-prefix=PREFIX            prepend PREFIX to installed program names
  --program-suffix=SUFFIX            append SUFFIX to installed program names
  --program-transform-name=PROGRAM   run sed PROGRAM on installed program names

X features:
  --x-includes=DIR    X include files are in DIR
  --x-libraries=DIR   X library files are in DIR

System types:
  --build=BUILD      configure for building on BUILD [guessed]
  --host=HOST        cross-compile to build programs to run on HOST [BUILD]

Optional Features:
  --disable-option-checking  ignore unrecognized --enable/--with options
  --disable-FEATURE       do not include FEATURE (same as --enable-FEATURE=no)
  --enable-FEATURE[=ARG]  include FEATURE [ARG=yes]
  --enable-silent-rules   less verbose build output (undo: "make V=1")
  --disable-silent-rules  verbose build output (undo: "make V=0")
  --enable-dependency-tracking
                          do not reject slow dependency extractors
  --disable-dependency-tracking
                          speeds up one-time build
  --enable-ssl ................................ Enable use of SSL library (MD5 checksums)
  --enable-frontend-x10   Enable the ROSE X10 Frontend (requires the X10
                          compiler, see --with-x10)
```

Figure 1: Example output from configure –help in `ROSE` directory (Part 1).

`configure --help` Option Output (Part 2)

```
  --enable-only-binary-analysis(=yes)
                          Enable ONLY binary support in ROSE (Warning:
                          '--enable-only-binary-analysis=no' and
                          '--disable-only-binary-analysis' are no longer
                          supported)
  --enable-only-c(=yes)   Enable ONLY C support in ROSE (Warning:
                          '--enable-only-c=no' and '--disable-only-c' are no
                          longer supported)
  --enable-only-cxx(=yes) Enable ONLY C++ support in ROSE (Warning:
                          '--enable-only-cxx=no' and '--disable-only-cxx' are
                          no longer supported)
  --enable-only-fortran(=yes)
                          Enable ONLY Fortran support in ROSE (Warning:
                          '--enable-only-fortran=no' and
                          '--disable-only-fortran' are no longer supported)
  --enable-only-java(=yes)
                          Enable ONLY Java support in ROSE (Warning:
                          '--enable-only-java=no' and '--disable-only-java'
                          are no longer supported)
  --enable-only-x10(=yes) Enable ONLY X10 support in ROSE (Warning:
                          '--enable-only-x10=no' and '--disable-only-x10' are
                          no longer supported)
  --enable-only-php(=yes) Enable ONLY PHP support in ROSE (Warning:
                          '--enable-only-php=no' and '--disable-only-php' are
                          no longer supported)
  --enable-only-python(=yes)
                          Enable ONLY Python support in ROSE (Warning:
                          '--enable-only-python=no' and
                          '--disable-only-python' are no longer supported)
  --enable-only-cuda(=yes)
                          Enable ONLY Cuda support in ROSE (Warning:
                          '--enable-only-cuda=no' and '--disable-only-cuda'
                          are no longer supported)
  --enable-only-opencl(=yes)
                          Enable ONLY OpenCL support in ROSE (Warning:
                          '--enable-only-opencl=no' and
                          '--disable-only-opencl' are no longer supported)
  --enable-languages=LIST Build specific languages:
                          all,none,binaries,c,c++,cuda,fortran,java,x10,opencl,php,python
                          (default=all)
  --enable-binary-analysis
                          Enable binary analysis support in ROSE (default=yes)
  --enable-c              Enable C language support in ROSE (default=yes).
                          Note: C++ support must currently be simultaneously
                          enabled/disabled
  --enable-cxx            Enable C++ language support in ROSE (default=yes).
                          Note: C support must currently be simultaneously
                          enabled/disabled
  --enable-cuda           Enable Cuda language support in ROSE (default=yes)
  --enable-fortran        Enable Fortran language support in ROSE
                          (default=yes)
  --enable-java           Enable Java language support in ROSE (default=yes).
                          Note: --without-java turns off support for ALL
                          components in ROSE that depend on Java, including
                          Java language support
  --enable-x10            Enable X10 language support in ROSE (default=yes).
                          Note: --without-x10 turns off support for ALL
                          components in ROSE that depend on X10, including X10
                          language support
  --enable-php            Enable PHP language support in ROSE (default=yes)
  --enable-python         Enable Python language support in ROSE (default=no)
  --enable-opencl         Enable OpenCL language support in ROSE (default=yes)
  --enable-rtedupc        Enable UPC support in ROSE (default=no)
  --enable-compass2       build the Compass2 static analysis tool under
                          projects/
  --enable-projects-directory
                          Toggle compilation and testing of the the
                          ROSE/projects directory (disabled by default)
  --disable-tests-directory
                          Disable compilation and testing of the ROSE/tests
                          directory
  --disable-tutorial-directory
                          Disable compilation and testing of the ROSE/tutorial
                          directory
  --enable-memory-pool-no-reuse
                          Enable special memory pool model: no reuse of
                          deleted memory (default is to reuse memory)
  --enable-smaller-generated-files
                          ROSETTA generates smaller files (but more of them so
                          it takes longer to compile)
```

Figure 2: Example output from configure –help in `ROSE` directory (Part 2).

NMI Platform (OS and Machine) Prerequisites

```
x86_64_deb_5.0 : "gcc-4.2.4, boost-1.36.0, libtool-2.2.6b, automake-1.10, autoconf-2.63, libxml2-2.7.3"
x86_64_fedora_12-updated : "boost-1.36.0, libtool-2.2.6b"
x86_64_macos_10.5-updated : "boost-1.36.0, libtool-2.2.6b, automake-1.10, autoconf-2.63, libxml2-2.7.3, wget-1.9.1"
x86_64_rhap_5 : "gcc-4.2.4, boost-1.36.0, libtool-2.2.6b"
x86_64_rhap_5.2 : "gcc-4.2.4, boost-1.36.0, libtool-2.2.6b"
x86_64_rhap_5.3 : "gcc-4.2.4, boost-1.36.0, libtool-2.2.6b"
x86_64_rhas_4 : "gcc-4.2.4, boost-1.36.0, libtool-2.2.6b, automake-1.10, autoconf-2.63"
x86_deb_5.0 : "gcc-4.2.4, boost-1.36.0, libtool-2.2.6b, libxml2-2.7.3"
x86_macos_10.4 : "boost-1.36.0, libtool-2.2.6b, automake-1.10, autoconf-2.63, libxml2-2.7.3"
x86_rhap_5 : "gcc-4.2.4, boost-1.36.0, libtool-2.2.6b"
x86_rhas_3 : "gcc-4.2.4, boost-1.36.0, libtool-2.2.6b, autoconf-2.59, automake-1.10"
x86_rhas_4 : "gcc-4.2.4, boost-1.36.0, libtool-2.2.6b, automake-1.10, autoconf-2.63"
x86_sles_9 : "gcc-4.2.4, boost-1.36.0, libtool-2.2.6b, automake-1.10, autoconf-2.63, libxml2-2.7.3, tar-1.14"
x86_suse_10.0 : "gcc-4.2.4, boost-1.36.0, libtool-2.2.6b, automake-1.10, autoconf-2.63, libxml2-2.7.3"
x86_suse_10.2 : "gcc-4.2.4, boost-1.36.0, libtool-2.2.6b, libxml2-2.7.3"
```

Figure 3: Example NMI machine preques used for nightly tests.

NMI Platform (OS and Machine) Configure Options

```
x86_64_deb_5.0 : "--with-boost=/prereq/boost-1.36.0 --with-CXX_WARNINGS=-Wall --without-java"
x86_64_fedora_12-updated : "--with-boost=/prereq/boost-1.36.0 --with-CXX_WARNINGS=-Wall --without-java"
x86_64_macos_10.5-updated : "--with-boost=/prereq/boost-1.36.0 --with-CXX_WARNINGS=-Wall --without-java"
x86_64_rhap_5 : "--with-boost=/prereq/boost-1.36.0 --with-CXX_WARNINGS=-Wall --without-java"
x86_64_rhap_5.2 : "--with-boost=/prereq/boost-1.36.0 --with-CXX_WARNINGS=-Wall --without-java"
x86_64_rhap_5.3 : "--with-boost=/prereq/boost-1.36.0 --with-CXX_WARNINGS=-Wall --without-java"
x86_64_rhas_4 : "--with-boost=/prereq/boost-1.36.0 --with-CXX_WARNINGS=-Wall --without-java --disable-rosehpct"
x86_deb_5.0 : "--with-boost=/prereq/boost-1.36.0 --with-CXX_WARNINGS=-Wall --without-java"
x86_macos_10.4 : "--with-boost=/prereq/boost-1.36.0 --with-CXX_WARNINGS=-Wall --without-java"
x86_rhap_5 : "--with-boost=/prereq/boost-1.36.0 --with-CXX_WARNINGS=-Wall --without-java"
x86_rhas_3 : "--with-boost=/prereq/boost-1.36.0 --with-CXX_WARNINGS=-Wall --without-java"
x86_rhas_4 : "--with-boost=/prereq/boost-1.36.0 --with-CXX_WARNINGS=-Wall --without-java"
x86_sles_9 : "--with-boost=/prereq/boost-1.36.0 --with-CXX_WARNINGS=-Wall --without-java"
x86_suse_10.0 : "--with-boost=/prereq/boost-1.36.0 --with-CXX_WARNINGS=-Wall --without-java"
x86_suse_10.2 : "--with-boost=/prereq/boost-1.36.0 --with-CXX_WARNINGS=-Wall --without-java"
```

Figure 4: Example NMI machine configure options used for nightly tests.