Porting Java Applications to Run on SUSE®

Simon Nattrass
Software Engineer, Strategic Partner Engagements
Novell, Inc.

Novell®

Java on the Linux platform

For most people "Java" *is* the official distribution from Sun Microsystems.

Other implementations of both Java Virtual Machine (JVM) and Java Development Kit (JDK) are available

IBM Developer Kit for Linux

Blackdown – a Java Environment specifically for Linux

Kaffe - a clean room implementation of the Java virtual machine, plus the associated class libraries needed to provide a Java runtime environment

Novell.

Java on SUSE

The SUSE distributions offer both the user and developer a choice Java Runtime Environments (JRE) and Java Development Kits (JDK), with a vanilla installation providing the following environments by default:

SLES 9 – IBM JRE 1.4.2

SLES 8 – Sun JRE 1.3.1 and IBM JDK 1.3.1

Novell Linux Desktop – Sun JRE 1.4.2

Novell.

SUSE Java Environment Config

SUSE supports the installation of multiple Java environments

System wide JVM defined by the `/usr/lib/java` symbolic link

The other installed environments are available to applications when specifically requested

The directory `/etc/java` contains configuration files for all installed JDKs and JREs, with one file for each installed JRE or JDK.

Novell.

Example Java Config File

Shell - Konsole

Session Edit View Bookmarks Settings Help

```
# Configuration for java2-jre package

Priority: 40

Vendor  :  Sun
Version :  1.4.2
Devel   :  False

JAVA_BINDIR = /usr/lib/SunJava2-1.4/jre/bin
JAVA_ROOT   = /usr/lib/SunJava2-1.4
JAVA_HOME   = /usr/lib/SunJava2-1.4/jre
JRE_HOME    = /usr/lib/SunJava2-1.4/jre
JDK_HOME    =
SDK_HOME    =

JAVA_LINK   = SunJava2-1.4
java2-jre.conf lines 1-16/16 (END)
```

Shell

Novell.

Config Files - Parameters

Each configuration file contains the following parameters:

Priority - the JDK or JRE with higher priority (lower value) is selected to be the system wide setting

Vendor - defines vendor name and allows selection of Java component based on this name

Version - defines JDK or JRE version. Allows selection of based on version

Devel - either "True" or "False" with “True” defining a JDK component

Novell.

Config Files – Environment Vars

JAVA_BINDIR - defines the path to Java executables and will used for PATH and JAVA_BINDIR environment variables

JAVA_ROOT - defines the path to the Java root directory and will be the same for JDK and JRE of the same version from the same vendor. This defines their JAVA_ROOTDIR environment variable

Novell.

Config Files – Environment Vars

JAVA_HOME - similar to JAVA_ROOT and used for applications use this value to find Java executables in $JAVA_HOME/bin. Will be used for JAVA_HOME environment variable

JRE_HOME - some applications use this value to search for Java runtime executables and libraries in $JRE_HOME/bin and $JRE_HOME /lib. Will be used for JRE_HOME environment variable

Novell.

Config Files – Environment Vars

JDK_HOME - added as supplement to JRE_HOME and is defined for all JDKs. Will be used for JDK_HOME environment variable

SDK_HOME - defined for JDKs with version 1.2 and higher. Will be used for SDK_HOME environment variable

JAVA_LINK - used by scripts `setDefaultJava` and `SuSEconfig`, the link `/usr/lib/java` and other compatibility links will point to this directory name or to this path

Novell.

SUSE Java Environment Config

These configuration files are read by the scripts:

`/usr/sbin/setDefaultJava` - changes the system wide default Java environment via updating the symbolic link **`/usr/lib/java`**

`/usr/bin/setJava` - updates the current shell only

Both are wrappers for **`setJava.pl`** Perl script

Novell.

Switching Between Java Environments

Example: Change link in /usr/lib from JRE to highest priority JDK

```
setDefaultJava --devel
```

Example: Set any Java with version 1.3.x

```
setDefaultJava --strict --version 1.3
```


Novell.

Switching Between Java Environments

Example: Set any Java with version 1.3.1

```
setDefaultJava --strict --version
1.3.1
```

Example: add link “foo” in /usr/lib pointing to current Java environment

```
setDefaultJava --linkname foo
```


Novell.

Switching Between Java Environments

Example: Set any Java from Sun Microsystems

```
setDefaultJava --vendor Sun
```

Example: Set any Java version 1.3 or above

```
setDefaultJava --version 1.3
```


Novell.

Switching Between Java Environments

The **setJava** script is required to be run with the **source** shell command since the result of the script must be sourced to change the settings in the current shells.

Example: Set any Java version 1.3 or above

```
source setJava --version 1.3
```


Novell.

SuSEconfig – Additional Links

The **`SuSEconfig`** script maintains some links which can also help you to find valid JDK or JRE:

- **`/usr/lib/java1`** - any JDK / JRE with version 1.1.x
- **`/usr/lib/java2`** - any JDK / JRE with version 1.2 or higher
- **`/usr/lib/SunJava1`** - any JDK / JRE from Sun Microsystems with version 1.1.x
- **`/usr/lib/SunJava2`** - any JDK / JRE from Sun Microsystems with version 1.2 or higher
- **`/usr/lib/IBMJava2`** - any JDK / JRE from IBM with version 1.2 or higher

SuSEconfig – Additional Links

Not all the links have to exist all the time. SuSEconfig uses the following rules when creating these links:

- create the link when a related JDK / JRE exists
- do not change the link which points to a valid JDK / JRE
- fix the link that points a none existing target or a directory containing no JDK / JRE, yet a related JDK / JRE exists
- remove the link that points a none existing target or a directory containing no JDK / JRE, and a related JDK / JRE does not exist

Config Scripts – Additional Notes

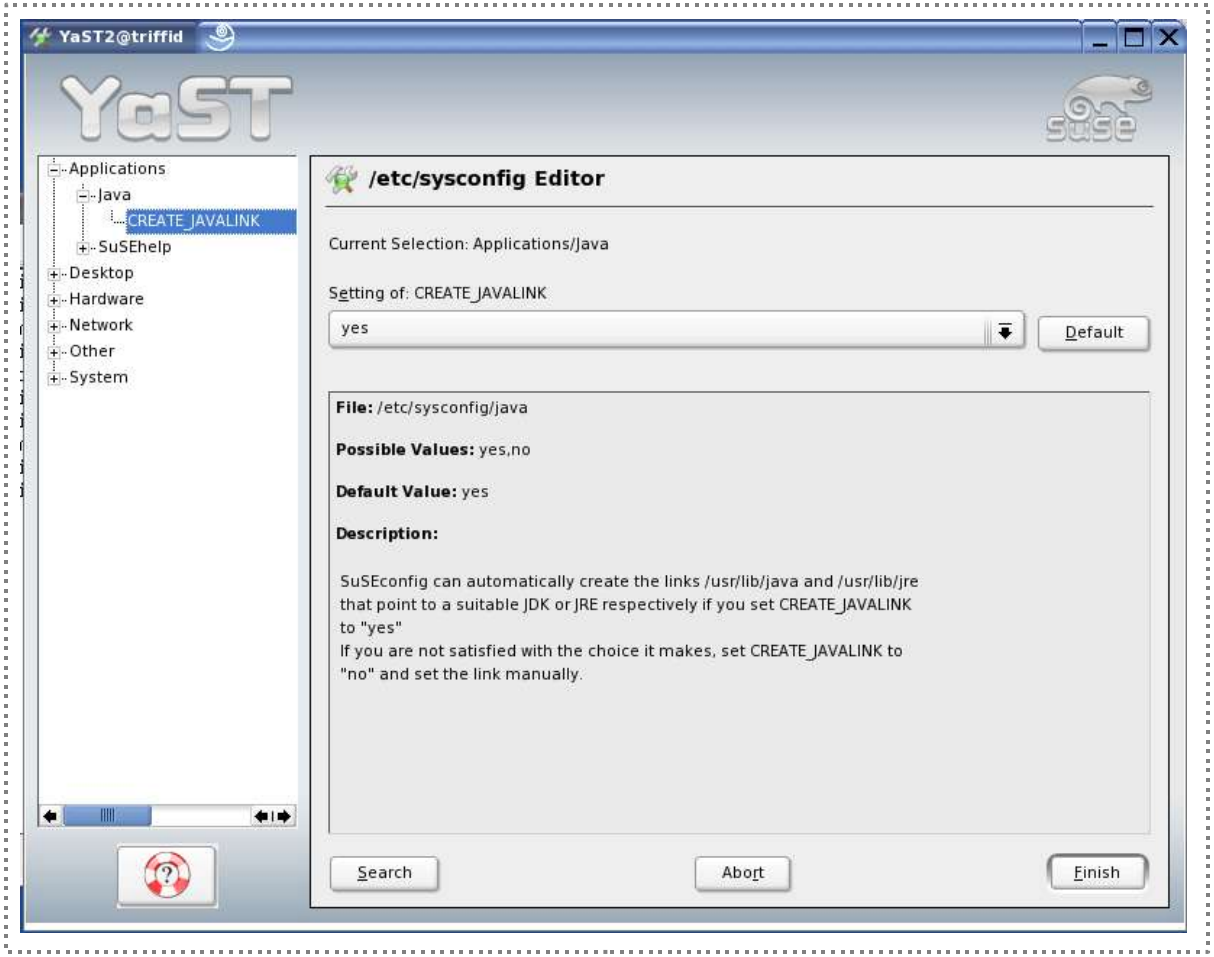
You may disable this SuSEconfig behavior by setting the variable CREATE_JAVALINK="no" in /etc/sysconfig/java or indirectly via YaST (System -> /etc/sysconfig Editor -> Applications -> Java).

Note - Java applications distributed within SUSE may use the above links.

By default SuSEconfig ensures the validity of **/usr/lib/java** and the link will be corrected if it points to an invalid directory

Novell.

YaST Java Config



Novell.

GUI Applications

Largely platform agnostic with a few issues to be aware of when developing for Linux

- Events
- Cut and Paste
- Fonts
- AWT and X11

Novell.

GUI Applications - Events

For pop up menus the trigger event on Linux is a mouse *press* while the same event on the Windows platform is a mouse *release*. Thus both events require checking to see if *MouseEvent.isPopupTrigger()* returns true.

GUI Applications - Events

```
public void mousePressed(MouseEvent e) {
        maybeShowPopup(e); // for Linux
    }

    public void mouseReleased(MouseEvent e) {
        maybeShowPopup(e); // for Windows
    }

    private void maybeShowPopup(MouseEvent e) {
        if (e.isPopupTrigger()) { ... }
    }
```

GUI Applications – Cut and Paste

Under the X Windows this is provided by two mechanisms:

selection: text is selected for copying by dragging the cursor, then copied to other applications by clicking the middle mouse button. The data is stored in the *primary* buffer. Most X applications support primary selection.

clipboard: applications can cut/copy/paste text using menu selections and common key shortcuts – Control-C for copy, Control-X for cut, and Control-V for paste. The data is stored in the clipboard buffer. Many applications do not support this method.

Novell.

GUI Applications – Cut and Paste

The AWT supports the only supports clipboard buffer. For primary selection functionality developers will need to use JNI to make the required calls to the X server.

Novell.

GUI Applications - Fonts

Java defines five *logical* font names: *Serif*, *SansSerif*, *Monospaced*, *Dialog*, and *DialogInput* which are mapped to *physical* fonts in implementation dependent ways via the **`font.properties`** files.

Logical fonts do not always map onto the same physical fonts across all platforms - GUI applications may differ in appearance of their fonts.

Users can edit or create their own **`font.properties`** files to adjust the mappings to their particular system setup (not supported by Sun)

Novell.

font.properties

Shell - Konsole

Session Edit View Bookmarks Settings Help

```
#
# ==========================================================================
# Copyright 2002 Sun Microsystems, Inc. All rights reserved.
#
# ==========================================================================
#



# font.properties
#
# Intended for Linux
# Note: modification of this file is risky and not supported
#     see http://java.sun.com/products/jdk/1.2/docs/guide/internat/fontprop.html
serif.0=-jdk-lucidabright-medium-r-normal--*-%d-75-75-p-*-iso8859-1
serif.1=-jdk-lucidabright-medium-r-normal--*-%d-75-75-p-*-iso8859-15
serif.2=-monotype-timesnewromanwt-medium-r-normal--*-%d-75-75-p-*-microsoft-symb
ol

serif.italic.0=-jdk-lucidabright-medium-i-normal--*-%d-75-75-p-*-iso8859-1
serif.italic.1=-jdk-lucidabright-medium-i-normal--*-%d-75-75-p-*-iso8859-15
serif.italic.2=-monotype-timesnewromanwt-medium-r-normal--*-%d-75-75-p-*-microso
ft-symbol
font.properties lines 12-32/303 9%
```

Shell

Novell.

GUI Applications – AWT and X11

The AWT uses the native graphics resources of the system to implement drawing operations of primitives such as **`drawLine ()`**, **`fillOval ()`**, **`drawString ()`** etc...

Ensures the best performance

Problems when no X11 Display is available (headless environment), for example a J2EE application running returning dynamically generated images like pies, charts, etc...

Novell.

GUI Applications – AWT and X11

A Popular misconception is that the system property **`java.awt.headless=true`** fixes this issuse

Running J2EE code similar to the above without any X11 installation results in **`java.lang.UnsatisfiedLinkError`** to **`libawt.so`**

If we look deeper at **`libawt.so`** we see that there is still a dependency on the X11 libraries:

Novell.

GUI Applications – AWT and X11

```
ldd libawt.so

  libmlib_image.so => not found
  libjvm.so => not found
  libXp.so.6 => not found
  libXt.so.6 => not found
  libXext.so.6 => not found
  libXtst.so.6 => not found
  libX11.so.6 => not found
  libm.so.6 => /lib/i686/libm.so.6
  (0x40306000)
  libdl.so.2 => /lib/libdl.so.2 (0x40329000)
  libjava.so => not found
  libc.so.6 => /lib/i686/libc.so.6
  (0x4032d000)
  /lib/ld-linux.so.2 => /lib/ld-linux.so.2
  (0x80000000)
```

Threading

Both green and native threads are mechanisms to support multi-threaded execution of Java programs. Some JDK distributions include the option to run with either type of threading.

Sun Java version 1.3 supports both threading models, native by default and green via the –**`classic`** flag to the JVM

Java version 1.4 has since dropped green thread functionality

Novell.

Native Threads

Native threads use the operating system's native ability to manage multi-threaded processes

They use the pthread library

Kernel schedules and manages the various threads that make up the process.

Novell.

Green Threads

Green threads emulate multi-threaded environments without relying on any native OS capabilities

They run code in user space that manages and schedules threads

Sun wrote green threads to enable Java to work in environments that do not have native thread support

Novell.

Native and Green Compared

Native threads can switch between threads pre-emptively, switching control from a running thread to a non-running thread at any time. Green threads only switch when control is explicitly given up by a thread (Thread.yield(), Object.wait(), etc.) or a thread performs a blocking operation (read(), etc.).

On multi-CPU machines, native threads can run more than one thread simultaneously by assigning different threads to different CPUs. Green threads run on only one CPU.

Novell.

Native Threads

native threads provides an implementation which is simpler and stable, but has creates the *appearance* that many Java processes are running since each thread takes up its own entry in the process table.

One clue that these are all threads of the same process is that the memory size is identical for all the threads - they are all using the same memory.

Novell.

Native Threads and Process Table

Unfortunately, this behavior limits the scalability of Java on Linux.

Process table is not infinitely large, and processes can only create a limited number of threads before running out of system resources or hitting configured limits.

```
snattrass@blackfish:~> ps -ef | grep java
snattra  27447 20860 96 15:06 pts/6    00:00:36 java -cp . SpawnThreads
snattra  27448 27447  0 15:06 pts/6    00:00:00 java -cp . SpawnThreads
snattra  27449 27448  0 15:06 pts/6    00:00:00 java -cp . SpawnThreads
snattra  27450 27448  0 15:06 pts/6    00:00:00 java -cp . SpawnThreads
snattra  27451 27448  0 15:06 pts/6    00:00:00 java -cp . SpawnThreads
snattra  27452 27448  0 15:06 pts/6    00:00:00 java -cp . SpawnThreads
snattra  27453 27448  0 15:06 pts/6    00:00:00 java -cp . SpawnThreads
snattra  27454 27448  0 15:06 pts/6    00:00:00 java -cp . SpawnThreads
snattra  27455 27448  0 15:06 pts/6    00:00:00 java -cp . SpawnThreads
snattra  27499 27456  0 15:07 pts/7    00:00:00 grep java
```


Novell.

Threading Improvements

Confusing **`ps`** command display issue is fixed in the 2.0.7 version of the **`procps`** package and with the advent of the new 2.6 kernel which provides proper thread grouping and reporting

The 2.6 kernel introduces a new, improved threading model that is implemented through the *Native Posix Thread Library* (NPTL)

The NPTL approach keeps the 1:1 thread mapping, 1 user or Java thread to 1 kernel thread, but optimizes the kernel for thread related operations, including signal handling, synchronization and thread creation speed.

Novell.

Threading

We can see the dependence of the JVM on the pthreads library below:

```
ldd java
        libpthread.so.0 => /lib/i686/libpthread.so.0 (0x40029000)
        libdl.so.2 => /lib/libdl.so.2 (0x4007a000)
        libc.so.6 => /lib/i686/libc.so.6 (0x4007d000)
        /lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x40000000)
```


Novell.

Generating a Java Stack Trace

Java threads are implemented as cloned processes, the task of debugging Java programs running on Linux differs from other platforms

To generate a stack trace from the terminal window in which the application was started, the sequence <CTRL>\ is typed to send the QUIT signal which generates a stack trace

If <CTRL>\ fails to give the expected result try checking the mapping of key sequence to signal via typing: **`stty -a`**. The entry to look for being **`quit = <key sequence>`** where **`^`** typically denotes CTRL

Novell.

Generating a Java Stack Trace

To generate a stack trace from a Java application already running in the background, the QUIT (signal number 3) needs to be explicitly sent to the launcher process via the kill command: kill -3 <process id>.

Example kill -3 1234, where 1234 is the launcher process id.

The resulting stack trace will be a snapshot of the Java program execution detailing the state of each thread. This information can be used to track down deadlocks or busy sections in your application.

Novell.

Links

Sun Java Technologies

http://java.sun.com

The font.properties file

http://java.sun.com/j2se/1.4.2/docs/guide/intl/fontprop.html

Java Technology Forums

http://forum.java.sun.com

Java Technology on the Linux Platform

http://java.sun.com/developer/technicalArticles/Programming/linux

Novell.

Novell®

Novell