RED HAT ENTERPRISE LINUX: APPLICATION COMPATIBILITY GUIDE

December 2013

www.redhat.com

Table of Contents

EXECUTIVE SUMMARY

This guide is intended to educate independent software vendors (ISVs) and customers on Red Hat's guidelines regarding support of third-party applications across multiple releases of the Red Hat Enterprise Linux platform. ISV and customer applications can reduce or avoid migration issues between major Red Hat® Enterprise Linux® versions by following these guidelines during application development.

This document describes the recommended uses of the system application programming (API) and binary interfaces (ABI) that are intended to provide compatibility across Red Hat Enterprise Linux releases. It outlines the tiered framework under which applications are considered compatible and not compatible.

INTRODUCTION

These guidelines seek to provide stability for applications when new releases are deployed and therefore focus primarily on forward-compatibility issues. Although backward compatibility is the goal, development of new capabilities sometimes makes that impractical. Hence, the guidelines and published policy may sometimes be over-ridden by the objective of providing our customers with the most competitive and capable systems possible; situational testing is always recommended.

If clarification of these compatibility guidelines is required, please see the Red Hat Enterprise Linux 6 Developer Guide[1] for additional details, or contact your Red Hat representative.

1 http://docs.redhat.com/docs/en-US/Red_Hat_Enterprise_Linux/6/html/Developer_Guide/index.html

TERMINOLOGY

The following are basic terms used in this document:

- **Binary compatibility**

  Binary compatibility means applications that are compiled on a combination of Red Hat Enterprise Linux and a particular hardware architecture will load and run similarly across different instances of the operating environment. Application binaries consist of executable files and Dynamic Shared Objects (DSO), and the level of compatibility is defined by a specific application binary interface (ABI).

- **Application programming interface (API)**

  An API is an interface implemented by a software program that enables it to interact with other software, including operating system components. The API is enforced at compile time and determines source compatibility, that is, whether application source code will compile similarly across different instances of the operating environment.

- **Application binary interface (ABI)**

  An ABI is a set of runtime conventions that interact with a compiled binary representation of a program. ABI is also a superset of the API and describes the low-level interface between an application and the operating environment. It covers details such as:

  - data type, size, and alignment
  - the calling standard, which defines how function arguments are passed and return values retrieved
  - the binary format of object files and program libraries

  Tools such as the compilers, linkers, runtime libraries, and the operating system itself need to work with the ABI. An ABI operates at runtime.

- **ABI conformance**

  A compiler conforms to an ABI if it generates code that follows all of the specifications enumerated by that ABI. A library conforms to an ABI if it is implemented according to that ABI. An application conforms to an ABI if it is built using tools that conform to that ABI and does not contain source code that changes behavior specified by the ABI or that otherwise bypasses the ABI.

- **Core persistent system infrastructure**

  The core persistent system infrastructure refers to interfaces and externally available data structures that represent system state or provide a means of communicating with the system (for instance, system calls and header files).

- **Compatibility in a virtualized environment**

  Virtual environments emulate bare-metal environments such that unprivileged applications that run on bare-metal environments will run, unmodified, in corresponding virtual environments. Virtual

environments present simplified abstracted views of physical resources, so some differences may exist.

- **Packages in Supplementary and Optional repositories**

  Packages available in the Red Hat Network (RHN) Supplementary and Optional channels are not part of the product but are made available for convenience or to satisfy build dependencies.

- **Major and minor releases**

  A Red Hat major release represents a significant step in the development of a product (wholesale changes are usually reserved for major releases), and is typically designated by a single numeral (e.g., Red Hat Enterprise Linux 6). Minor releases appear more frequently, within the scope of a major release, and generally represent smaller, incremental developmental steps.

- **Red Hat Enterprise Linux Extended Update Support**

  Red Hat Enterprise Linux Extended Update Support (EUS) is an optional support offering that allows a customer to standardize on a specific minor release for an extended period of time. Each Red Hat Enterprise Linux 6 EUS stream is available for 24 months from the availability of the minor release.

- **Red Hat Enterprise Linux Advanced Update Support**

  Red Hat Enterprise Linux Advanced Update Support (AUS) is an optional support offering the allows a customer to stay on a specific minor release for up to 24 months and receive only critical errata.

- **Red Hat Developer Toolset (DTS)**

- Red Hat Developer Toolset (DTS) is an offering that provides the latest stable versions of compilers, debuggers, and select open source development tools. This optional offering has an independent life cycle and is not covered by this Application Compatibility document.

- **Red Hat Software Collections (RHSCL)**

  Red Hat Software Collections delivers the latest, stable versions of some of the most popular web development languages and open source databases for use with Red Hat Enterprise Linux. This optional offering has an independent life cycle and is not covered by the guidelines described in this Application Compatibility document.

- **SystemTap Static Probes**

  SystemTap static probes are part of the SystemTap profiling and tracing framework. The probes are integrated into key system libraries to support profiling and debugging of applications and libraries. No assurances are made at this time that integrated SystemTap static probes will continue to have the same probe name, probe location, or interpretation or number of arguments.

SCOPE OF COMPATIBILITY

Packages in Red Hat Enterprise Linux are classified under one of the following four compatibility levels:

- Compatibility level 1: APIs and ABIs are stable across three major releases: the release that introduces a new or revised API or ABI, and the two following major releases (*n*, *n*+1, *n*+2). In the case of this document, release *n* starts with Red Hat Enterprise Linux 6.

  If a change to a library during its life cycle causes an incompatibility with existing compiled code, a separate version of the library will be provided with the older ABI to run the application without modifications.
- Compatibility level 2: APIs and ABIs are stable within one major release.
  - In the case of this document, that is Red Hat Enterprise Linux 6.
- Compatibility level 3: *Reserved for future use*.
- Compatibility level 4: No compatibility is provided.

For compatibility levels for specific Red Hat Enterprise Linux software packages, please see Appendix A.

Compatibility levels for bare-metal configurations apply to virtualized configurations except for any features that directly interact with hardware. Those features directly related to hardware have no API or ABI compatibility level. For example, applications that rely on Graphics Processing Units (GPU) features cannot expect binary compatibility.

Red Hat strongly recommends that application developers validate that any behavior they depend on is explicitly defined in formal API documentation to prevent introducing dependencies on unspecified implementation-specific semantics or introducing dependencies on bugs in a particular implementation of an API. For example, new releases of the GNU C library may not be compatible with older releases if an application used an undocumented API or one with undefined behavior.

GUIDELINES FOR PRESERVING BINARY COMPATIBILITY

Red Hat recommends that application developers adopt the following principles in order to improve binary compatibility:

1. Build applications using the published interfaces of a library. Non-published (a.k.a. internal) interfaces are subject to change at any time, which can cause instability in the dependent application if improperly linked. Application developers should validate that any behavior they depend on is described in the published API documentation to prevent introducing dependencies on unspecified implementation-specific semantics or introducing dependencies on bugs in a particular implementation of an API. For example, new releases of the GNU C library are not guaranteed to be compatible with older releases if the old behavior was not consistent with a published specification.
2. Avoid static linking of libraries (C/C++). Static linking causes the executables to have their own version of the library. This increases the chance of an application not running predictably on a later

version of the operating system as these library dependencies might have changed along the way. Linking applications dynamically is strongly recommended in order to avoid this problem.

3. Limit linking applications to the core libraries at compatibility level 1. The core libraries (see Appendix A) are intended to preserve binary compatibility across three consecutive major releases.
4. Provide compatibility libraries for applications that have been built with libraries that are not at the desired compatibility level, provided the bundled libraries themselves only use the interfaces provided by the core libraries.
5. Package applications using the RPM mechanism. RPM provides a software-packaging mechanism that includes detailed specification of application dependencies. When creating RPMs, the following should be kept in mind:
   (a) Avoid using RPM triggers whenever possible.
   (b) Explicitly state all required runtime and build dependencies using the appropriate RPM syntax.
   (c) Do not modify, replace, or recompile files managed by Red Hat-provided RPM packages. Doing so may lead to unpredictable behavior.
   (d) When considering dependencies, do not assume that all possible packages will be installed on every Red Hat Enterprise Linux system. The default installed packages may change between major releases, between product variants of the same version, and on a customer's system. For instance, the Workstation product will have a different installed package set than the Server product. Rarely, and due to extenuating circumstances, packages might be removed between minor releases but Red Hat will provide notification if this occurs.
6. Follow the Filesystem Hierarchy Standard (FHS) version 2.3 when installing programs. Third-party software should be installed to the '/opt' subdirectory. More information on the FHS is available at: http://www.pathname.com/fhs/.
7. Applications should be built against libraries that correspond to the native hardware environment rather than a compatibility layer on top of the hardware. This is because the compatibility userspace contains a subset of system libraries as compared to the native userspace.
8. Do not design applications that rely on configuration files used by system packages. These files can change between major releases unless the upstream community is explicitly committed to preserving them.
9. If you require functionality from a package that is currently at a lower compatibility level than you wish, please contact Red Hat for review.

Note: During the life cycle of a major release, Red Hat makes commercially reasonable efforts to maintain binary compatibility for the core runtime environment across all minor releases and errata advisories. If necessary, Red Hat may make exceptions to this compatibility goal for critical impact security or other significant issues. Furthermore, as described above and in Appendix 1, major releases of Red Hat Enterprise Linux contain a limited set of backward-compatible libraries included in previous major releases to allow for the easy migration of applications. Typically, Red Hat applies changes in such a way as to minimize the amount of change and to maintain binary compatibility. Exceptions may apply for controlled package re-bases under certain circumstances.

APPENDIX A: COMPATIBILITY LEVELS FOR SPECIFIC PACKAGES AND LIBRARIES

Compatibility level 1: APIs and ABIs are stable across three major releases (starting with RHEL 6)

alsa-lib	libgcc	libtopology	openmotif
elfutils-libelf	libgfortran	libvirt-client	openssl
glibc	libgomp	libxml2	pam
glibc-utils	libselinux	libxslt	SDL
gtk2	libselinux-python	mesa-libGL	
krb5-libs	libstdc++	mesa-libGLU	

Compatibility level 2: APIs and ABIs are stable within one major release (RHEL 6)

atk	gstreamer-plugins-base	libgnome	pcre
audit-libs	httpd	libgnomeui	perl
audit-libs-python	java-1.6.0-openjdk[2]	libgudev1	perl-Digest-SHA
boost-date-time	kdebase	libhugetlbfs	perl-libs
boost-filesystem	kdebase-libs	libICE	perl-Time-Piece
boost-graph	kdebase-workspace	libjpeg	phonon-backend-gstreamer
boost-iostreams	kdebase-workspace-akonadi	libnotify	polkit
boost-math	kdebase-workspace-libs	libpng	popt
boost-program-options	kdegraphics	librsvg2	postgresql-libs
boost-python	kdegraphics-libs	libSM	pulseaudio
boost-regex	kdelibs	libtiff	pulseaudio-libs
boost-serialization	kdemultimedia	libudev	pulseaudio-libs-glib2
boost-signals	kdemultimedia-libs	libusb	PyQt4
boost-system	kdenetwork	libuuid	python
boost-test	kdenetwork-libs	libX11	python-libs
boost-thread	kdepim	libXau	qt
boost-wave	kdepim-libs	libXaw	qt-mysql
bzip2-libs	kdepimlibs	libXext	qt-odbc
cairo	kdepimlibs-akonadi	libXft	qt-postgresql
corosync	kdesdk	libXi	qt-x11
cups-libs	kdesdk-devel	libXinerama	qt3

2 Red Hat will maintain compatibility for OpenJDK based on the Java SE 6 specification guidelines.

cyrus-sasl-gssapi	kdesdk-libs	libXmu	qt3-MySQL
cyrus-sasl-lib	kdesdk-utils	libXpm	qt3-ODBC
cyrus-sasl-md5	kdm	libXrandr	qt3-PostgreSQL
db4	libacl	libXrender	readline
db4-cxx	libaio	libXt	ruby
dbus-glib	libattr	libXtst	scl-utils
dbus-libs	libblkid	mysql-libs	sqlite
elfutils-libs	libbonobo	ncurses-libs	startup-notification
expat	libcanberra	net-snmp-libs	systemtap
fuse-libs	libcanberra-gtk2	net-snmp-perl	tcl
GConf2	libcap-ng	net-snmp-python	tcp_wrappers-libs
glib2	libcurl	nss	tk
gmp	libgcj	nss-sysinit	unique
gnome-keyring	libgcj-devel	numactl	xz-libs
gnome-keyring-pam	libgcrypt	pango	zlib
gnutls	libglade2	papi	

Compability level 3: Reserved for Future Use

(*this section intentionally left blank*)

Compability level 4: Stability of APIs or ABIs subject to change at Red Hat's discretion

binutils	junit	openldap	postgresql-test
e2fsprogs-libs	libcgroup	postgresql-contrib	pulseaudio-libs-zeroconf
glade3-libgladeui	libcom_err	postgresql-docs	pulseaudio-module-bluetoo
gnome-desktop	libdrm	postgresql-plperl	pulseaudio-module-gconf
gstreamer	libss	postgresql-plpython	pulseaudio-module-x11
gstreamer-devel	mesa-dri-drivers	postgresql-pltcl	pulseaudio-utils
gvfs	mysql-server	postgresql-server	tkinter