

Linux Standard Base (LSB): “Single Linux” Specification and Support Infrastructure

Vladimir Rubanov
Institute for System Programming
Russian Academy of Sciences
email: vrub@ispras.ru

Abstract (English)

Problems with portability of applications across various Linux distributions are the key motivators for standardizing Linux as a single platform. This article overviews the main Linux standard targeted at this - Linux Standard Base (LSB), which is being developed by the Linux Foundation - the leading international consortium to foster the growth of Linux.

Current LSB status is described with the focus on technical infrastructure (central database as a primary source about standard's elements, web-portal for developers, analytical and decision-support systems, frameworks for automated development and execution of LSB conformance tests, certification system, etc.). The article emphasizes the critical role of the infrastructure in development of the standard itself and in successful adoption of it in real world practice.

Keywords: Linux, open interface standard, test development framework, API testing.

Linux Standard Base (LSB): спецификация «единой Linux платформы» и инфраструктура поддержки

Владимир Рубанов
Институт системного программирования
Российская Академия Наук
email: vrub@ispras.ru

Abstract (Russian)

Проблемы переносимости приложений между различными дистрибутивами Linux являются основными движущими факторами для стандартизации Linux в качестве единой платформы для приложений. В этой статье дается обзор основного стандарта, нацеленного на решение этих проблем - Linux Standard Base (LSB). Этот стандарт разрабатывается международным консорциумом Linux Foundation, ведущей мировой организацией, нацеленной на обеспечение развития Linux.

В статье описывается текущий статус стандарта с фокусом на инфраструктуре его поддержки, в которую входят центральная база данных о стандартизованных элементах и окружающей экосистеме, веб-портал для разработчиков, системы анализа и поддержки принятия решений о стандартизации тех или иных элементов, методологии и средства автоматизированной разработки тестов, системы автоматического запуска тестов и поддержки процесса сертификации. Показывается ключевая роль инфраструктуры для развития стандарта и обеспечения реального внедрения в жизнь.

Keywords: Linux, открытый стандарт на интерфейсы, автоматизированная разработка интерфейсных тестов.

1. Introduction

Linux has established a track record as an industrial strength operating system. It is deployed on an ever increasing number of computers from supercomputers, servers and desktops to mobile phones and embedded systems. We are witnessing the extraordinary growth in the Linux community including application developers. However, there are outstanding issues, which need to be addressed. Application developers are faced with a large and constantly increasing number of Linux distributions from different vendors and they need to take into account the many differences between these packages when they want to develop applications “for the Linux platform”. That’s why defining Linux as a uniform platform is crucial for keeping the growth and ensuring vitality of this operating system.

Removing differences in the core platform services provided by different distributions is important for decreasing the costs of developing Linux applications. This directly affects the number of applications available for this platform, which is one of the key characteristics of operating system adoption.

At the moment of writing this article there are 549 (!) different distributions registered at <http://lwn.net/Distributions/>. And this number does not take into account special versions developed for internal company or individual use. But what is Linux from the application developer’s point of view? It is a combination of system components such as kernel and libraries that work jointly to provide application programming interfaces (APIs) for applications. The problem is that every Linux distribution consists of a unique set of specific versions and builds of such components, which results in that distributions may vary in the number of and behavior of the provided to applications interfaces both at the API and binary levels. That’s why applications that work on one distribution may fail on another and supporting multiple distributions becomes a serious problem for application developers. Of course, it is possible to develop specific versions of an application for particular distributions but this is rather expensive and may not be affordable for some developers, which may completely reject supporting Linux platform due to this. This inhibits growth of Linux applications base and the adoption of the platform itself as developers want to develop applications “for Linux” not just for RedHat or Suse.

To approach this problem, it was proposed to standardize a common set of functionality in the

main distributions and recommend using only these functions when developing portable Linux applications. This would enable portability of compliant applications across all compliant distributions.

This article contains two sections. The first one overviews the main industry standard targeted at solving portability problems for Linux applications - Linux Standard Base (LSB) [1]. To successfully develop and promote such a standard, it turned out to be extremely important to have a technical infrastructure that holds various data in a central place that is then used for generating text of the standard itself, for creating standard development environment and helps analyzing the situation in the Linux ecosystem for making right decisions about standardization scope. The last section presents the experience and results of the Institute for System Programming of the RAS in building such an infrastructure for the Linux Foundation.

2. Linux Standard Base (LSB) - the Single Linux Platform

The core idea of the Linux Standard Base (LSB) standard is to describe a subset of Linux interfaces provided by various libraries with the kernel underneath. This subset should be present in most Linux distributions and provide compatible services. Description should include information about binary level symbol of each interface and API-level information (parameters, return values and corresponding types) including behavior of the interfaces. To develop such a standard, a non-profit international consortium was founded in 2000 - Free Standards Group (at present Linux Foundation [2]), supported by the leading IT companies including IBM, Intel, HP, Novell, Oracle. The first version of the standard was published in June 2001 and covered about 3000 interfaces. In the next years the standard grew and matured, each new version covered more and more interfaces (while some selected were excluded). The current LSB version is 3.1 and it includes over 30000 interfaces from more than 40 libraries. Most of the main Linux distributions have now been certified for LSB compliance.

The important thing about LSB is that it does not try to impose something completely new on Linux distribution vendors. In many cases LSB just refers to established industry standards and documentation that existing implementations conform to de-facto. Such referenced specifications include SUSv3 (POSIX), ISO C99, ISO C++ Language and numerous documentation sources

maintained by upstream component developers. And only in the case of missing stable description of an interface, LSB describes it on its own based on real-world implementation of this interface present in the main distributions.

When considering standardization scope, LSB uses “best practice” criterion. This means an interface becomes a candidate for inclusion in LSB when it is present in all the main Linux distributions and is actively used by real applications. In other words, interfaces for standardization should be quite popular both in the distribution and in the application domains. Also, some technical requirements should be met such as that interfaces should have stable implementation, documentation and tests. It is important to note that the role of vendor-independent international consortium helps Linux Foundation to take unbiased decisions when developing LSB.

The modern LSB version 3.1 includes 3 so called modules:

- **LSB Core** - low-level system interfaces in C (libc, libcrypt, libdl, libm, libpthread, librt, libutil, libpam, libz and libncurses libraries).
- **LSB C++** - standard real-time support library for C++ (libstdc++ library).
- **LSB Desktop** - various functions for working with graphical interfaces and auxiliary services (mainly XML, X11, GTK and Qt).

Each module includes both generic (architecture independent) and architecture specific elements that are roughly structured in the following hierarchy: module->library->group->interface->type. LSB 3.1 supports 7 architectures - IA32 (x86), AMD64 (x86_64), IA64 (Itanium), Power PC 32, Power PC 64, IBM S390 и IBM S390X.

An important factor in understanding purpose of LSB is that this standard is not targeted at all Linux distributions and applications. It is for the *most* of them that are quite general purpose. Specialized distributions (like embedded) and some system applications may not need to be fully LSB compliant. Meanwhile, even if an application uses some interfaces beyond LSB then LSB still does matter when developing such application as it allows reducing development costs due to the intersection with LSB that one can rely upon. And to enable compatibility for those interfaces outside LSB, one can use separate methods like static linking of necessary libraries or developing special stub proxies that hide differences between different distributions. LSB just allows reducing the number of interfaces for which such special means are needed.

3. LSB Support Infrastructure

One of the key success factors for developing and supporting an interface standard like LSB is a proper technical infrastructure that automates main processes for maintaining the standard itself and that brings the standard closer to real developers. In particular case of LSB the main components of its infrastructure include generators of the standard’s text itself and associated header files based on a central database, a web-portal for LSB developers, analytical and decision support systems, tests and frameworks for their effective development, execution, result analysis and finally certification. These systems will be overviewed later in this section after history background.

3.1. LSB Infrastructure Program

Russian specialists are actively involved into the LSB community. The first large project in this field was Open Linux VERification (OLVER) – <http://linuxtesting.org/project/olver>. The project was done by our Russian Linux Verification Center [3] at the Institute for System Programming of the Russian Academy of Sciences (ISP RAS) [4]. The project was funded by the Russian Federal Agency for Science and Innovation. We analyzed text of the LSB Core standard for about 1500 Linux system interfaces, delineated atomic assertions and transformed them into formal specifications in SeC language from which we then generated conformance tests for automated testing of Linux distributions against LSB Core requirements [5].

OLVER project results turned out to be of high interest for the community and LSB standard body committee (Free Standards Group at that moment), which proposed ISP RAS long term cooperation for building a new LSB infrastructure to meet the growing needs of the Linux industry for this standard. The weak infrastructure existed at that moment (late 2006) was the most burning point for LSB community that hindered wider LSB adoption. In particular, it was critical to improve the certification framework and strengthen test coverage. In early 2007, Free Standards Group merged with Open Source Development Labs and the newly created Linux Foundation extended cooperation with ISPRAS to cover more infrastructure areas and prepare the jump base for promoting wide LSB adoption. It is important to note that all results (specifications, tools, frameworks) contributed by the Linux Foundation and its partners like ISP RAS to the LSB eco-system are open-source and Linux developers are

encouraged to use them and adapt. The first results of the new LSB Infrastructure program [6] were announced in June 2007 at the Linux Foundation Collaboration Summit in Mountain View, CA and since that moment they have been continuously improved with regular releases. Next in this section we describe the current (October 2007) status of these developments and summarize our experience gained so far.

3.2. LSB Database and Associated Tools

The backbone of the entire technical LSB infrastructure is a central database (MySQL engine is used) that holds integrated information about the LSB standard itself and about its surrounding Linux ecosystem. Since the beginning of the LSB Infrastructure program we have completely redesigned and extended its schema and data. Now the database contains 64 tables with over 13 million records. One can distinguish two parts of the database - standard part and community part. The first part includes information about LSB elements that constitute the essence of the standard itself. The elements include the following:

- grouping elements:
 - modules (collections of libraries and commands);
 - libraries (collections of library groups and headers);
 - headers (collections of interfaces, types and constants);
 - library groups (collections of classes and interfaces);
- and leaf elements:
 - commands;
 - classes;
 - interfaces (global variables and public functions);
 - constants and macros;
 - types.

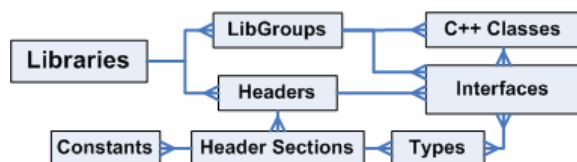


Fig.1. ABI part of the LSB DB.

These elements are interlinked into a graph of dependencies of various kinds (see Fig.1). Information contained in this part of the database is enough to generate complete header files that define all standardized interfaces.

Community part of the database contains information about modern Linux distributions and applications (see Fig.2). The idea of this part is to have a single place with “raw” information about the Linux ecosystem from the platform standardization point of view. Basically it allows understanding which interface elements are provided by particular versions of distributions and which ones are required by various particular applications.

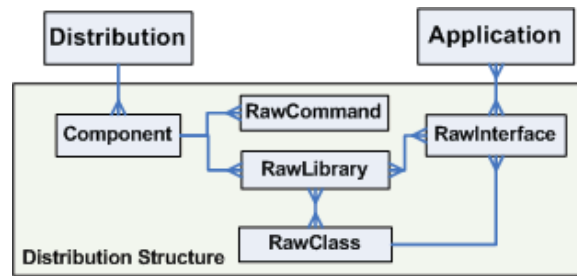


Fig.2. Community part of the LSB DB.

The database is used by various tools among which one should emphasize over 40 scripts that generate LSB deliverables such as the text of the standard itself, header files and various code blocks that are part of implementation of other LSB tools (mainly SDK and tests).

To efficiently use the database information from the human point of view, we have created a web-portal that provides user interface and brings the database closer to people. This is a public portal named *LSB Navigator* (<http://linux-foundation.org/navigator/>) that provides search, filtering & browsing capabilities to effectively find necessary information about LSB and the Linux ecosystem. It can be used by Linux developers, Linux distribution vendors and LSB workgroup to browse, query, analyze and submit various information. It is important that LSB Navigator is used by LSB workgroup for gathering info when taking decisions about the scope for standardization. Selected features of LSB Navigator include:

- Navigation through the LSB elements from modules down to leaf elements like interfaces and constants.
- Global filters for LSB version and hardware architecture.
- Individual home pages for over 1 million of Linux interfaces (each is just 2-click away from the main page via search) that include information on:
 - status of each interface in terms of LSB (in LSB, never been in LSB, planned for inclusion, withdrawn, deprecated);
 - LSB info (module, library, header file, etc.);
 - interface signature (parameters and return value);
 - direct link to documentation of the interface;
 - which distributions provide this interface;
 - which applications use this interface;
 - which tests are available for this interface;
 - community discussions related to the interface.
- Distribution info (provided libraries and interfaces).

- Application info (external libraries and interfaces required by each app).
- Statistics on LSB elements (total numbers of interfaces, commands, classes in each LSB version).
- Statistics on interface usage by applications:
 - which interfaces are most frequently used by different applications (list of interfaces with info on how many applications use each interface);
 - which libraries are most frequently used by different applications (list of libraries with info on how many applications use each library);
 - LSB “rating” of registered applications (list of applications with info on the number of LSB and non-LSB libraries and interfaces used by each application).

3.3. LSB Testing

Ian Murdock, the former LSB Chair, said: “An interface standard is only as good as its test suites”. In late 2006, the LSB test coverage was about 15%, which means that 85% of standardized interfaces did not have tests at all. That is why developing new tests that check conformance of distributions against the requirements of LSB standard was the first priority to enable real life value of this standard. The problem is that the number of interfaces in LSB is too huge to develop tests of excellent quality for all of them. To approach the strategy for developing LSB tests, we defined three testing quality grades (the borders between the testing grades are obscure as the scale is continuous actually):

1. **Shallow** – simple tests with the only guaranteed purpose of ensuring the interface does not crash (sometimes it is additionally checked that the interface doesn’t return an error code) being called with some particular correct parameters and in the correct environment. This is close to “existence” or “smoke” or “sanity” tests (but beware - these terms are interpreted differently by different experts).
2. **Normal** – this is the most reasonable level of testing achievable by tests written in plain C. The tests check the main functionality and may check a few error scenarios. Most of the legacy LSB tests are of this quality.
3. **Deep** – this is the level when most of the specification assertions are tested in various conditions/states. This is usually used for most important and critical software.

To develop tests of corresponding grades, we are using different approaches.

For *shallow tests*, we developed a new technology and tools (**AZOV Framework**) for automatic generation of shallow tests based on some description of the interfaces, their parameters, dependencies and default values in the LSB database. The core idea here is to augment the database to contain enough information about the interfaces and their dependencies that would allow automatically building correct call chains representing typical scenarios of interface usage. The first version of the shallow testing framework is ready and is now being used for Qt3 tests development. The cost of developing shallow tests is mainly in populating the information in the database + debugging automatically generated tests.

Normal tests are basically manual C tests, though we do use some automation here as well. We have been inspired by the generator that we found in the existing LSB tests for GTK – gtkvts. Basically, it allows automatically running the same code with different parameters thus achieving better testing quality without duplicating the code. However, we found particular gtkvts implementation limited and we used only the idea while implemented our own tools and methodology **T2C** (template-to-C). The first version of the T2C Framework is ready and is now being used for developing GTK tests.

Deep level of testing is hard to achieve by manual tests in C; so advanced testing technologies with automated test action generators like ISP RAS **UniTESK** are necessary here. Our former project OLVER was based on this technology. Now we are adapting/improving it to become official LSB tests for LSB Core part.

For the normal and deep tests, it is important to have linkage to the specification assertions’ text. This means when a test fails it should say which particular assertion in the standard’s text is violated as well as provide info about particular mismatch (like XX expected, YY returned). Normal and deep tests being developed in ISP RAS do have this feature and analyzing reports and debugging failures is much easier with this approach. Leveraged by visual execution environment and interactive HTML reports this gives unprecedented comfort for the users. Apart of better reporting, assertion catalog also enables measuring test coverage in terms of the number of assertions checked by the tests, which gives advanced level of test quality measurement.

Currently, there is a testing strategy being implemented by ISPRAS to achieve almost 100% test coverage for LSB interfaces by the middle of 2008 but with most of the tests of shallow quality. The target for the middle of 2009 is to cover most of the libraries with normal quality tests and the most important part (LSB Core) with deep quality tests.

To make LSB certification and testing technically appealing, it is important to have user-friendly systems that support these processes. As a part of LSB Infrastructure program we have been creating a new certification system that will guide people certifying their products through the certification workflow and will keep records for certified products. Currently a prototype is ready with a plan to deploy it for using in real certification for LSB 3.2 (December 2007). The new certification system includes three major parts:

- *Certification Management* provides step-by-step instructions on what to do and enables easy status tracking and collaboration with LF staff during the certification process for companies and individuals who want to certify their Linux distributions or applications against LSB.
- *Product Directory* is a public part of the Certification System that contains the current list of LSB-certified Linux distributions and applications with various views and groupings.
- *Problem Reporting* is for online collaboration on solving problems arising in the process of certification. It also provides a knowledge base of various issues and solutions.

There is also *Administration Mode* that allows managing the certification system from the LF side with full rights on administering all the data.

Finally, at the technical level of certification process it is necessary to be able to smoothly execute automated tests and analyze the results. For this purpose, we have developed *Distribution and Application Testkit Managers* (ATK and DTK Managers) for testing distributions and applications respectively. These tools are web-based with embedded simple web-server. The tools are test execution frameworks that have the following key features:

- integrated user interface for all the LSB test suites (web based and command line);
- selection of tests to run (all, manually selected subset, predefined subsets, etc.);
- saving/loading configured options in user profiles for quick test runs in the future;
- automatic download of missing test suites from the LF FTP site;
- 'one-button' execution of certification tests;
- unified test reports with links to the knowledge base of known issues and to home pages of interfaces in LSB Navigator for more information including community discussions;
- management of test results history.

Also, ATK Manager provides features for application analysis without regard to certification process. These features include viewing the list of all

external libraries and interfaces required by an application with status in LSB perspective (if in LSB or not). In particular, this allows detecting unused libraries present in DT_NEEDED section of application under analysis but that have no actual interfaces used by the application.

The online certification system and ATK/DTK Managers are integrated to provide transparent switches between local and the LF central-server based functionality to enable easy to use complete certification process.

4. Conclusion

Problems of application portability among different Linux distributions prove to be one of the most important factors that inhibit the growth of the number of applications available for Linux and thus prevent developing further success of the Linux platform as a whole. In this paper we considered Linux Standard Base open standard, which is the primary modern effort to address this problem. A number of industry companies initiated this activity to standardize a common subset of Linux functionality that most applications can rely upon. Currently the LSB standard is developed by international consortium Linux Foundation with funding of such companies as IBM, Intel, HP, Novell, Oracle, etc.

In order to develop a good interface standard such as LSB it is crucial to have a proper technical infrastructure that supports decision making processes, brings the standard closer to the industry needs and facilitates using the standard by actual developers. Linux Foundation jointly with the Institute for System Programming of the Russian Academy of Sciences are developing such an infrastructure for LSB standard. The current results of this cooperation include the first versions of the following systems:

- Central LSB Database & various data transformation scripts.
- LSB Navigator – a web portal on top of the central LSB database with advanced navigation, queries, community collaboration mechanisms, developers' feedback and contribution interface to promote information about LSB and surrounding ecosystem and also for making decisions on LSB standardization scope.
- LSB DTK Manager – to automate Linux distribution testing and make it user friendly.
- LSB ATK Manager – to automate Linux application testing and analysis.
- LSB Certification System - to support and facilitate LSB certification process.

- Misc. auxiliary tools for automating investigation and analytical tasks.
- New testing technologies and tools for automated test development of various quality grades:
 - **UniTESK** for deep testing (ISPRAS owned technology with over 10 years history).
 - **T2C** – methodology and tools for normal tests development (developed specially for the LF on top of TET harness and ideas of gtkvts .inp files).
 - **Azov** – innovative methodology and tools for automated massive development of shallow tests based on extended information from the central LSB database (developed specially for the LF).
- New tests for more than 9000 of LSB interfaces.

The developed infrastructure is being used in real life and has got many positive responses. Meanwhile, there are plans to improve many things in 2-year time perspective - first of all mature all the tools and achieve 100% test coverage of LSB

interfaces. This will help making LSB standard the acknowledged “single Linux platform” that will really mitigate application portability problems and will clear the way for further Linux expansion.

5. References

- [1] *Linux Standard Base Homepage.* <http://www.linux-foundation.org/en/LSB/>.
- [2] *Linux Foundation.* <http://linux-foundation.org/>
- [3] *Linux Verification Center.* <http://linuxtesting.org/>
- [4] *Institute for System Programming of the Russian Academy of Sciences.* <http://ispras.ru/>
- [5] V. Kuliamin, A. Petrenko, V. Rubanov, A. Khoroshilov. *Formalization of Interface Standards and Automatic Construction of Conformance Tests.* Proceedings of SECR 2006 conference, Moscow.
- [6] *LSB Infrastructure Program.* <http://ispras.linux-foundation.org/>