



The Linux Standard Base: Reducing Complexity for ISVs Targeting Linux

A White Paper Prepared by the Linux Foundation

OCTOBER 2008

The Linux Standard Base: Reducing Complexity for ISVs Targeting Linux

A White Paper Prepared by the Linux Foundation



An operating system's success is inextricably linked with the number and quality of applications that run on top of it. Linux® and its variances between distributions, however, present independent software vendors (ISVs) and individual developers with a unique set of challenges: different distributions of Linux make use of different versions of libraries, store important files in different locations, and so on. And yet, if an ISV wants to reach a global Linux audience, they must support more than one distribution of Linux. These requirements and variances can make it difficult--and costly--for ISVs to target the Linux platform.

It is somewhat of an irony: choice is what drives Linux adoption and creativity, yet this very choice can make things difficult for application developers. The costs and resources involved in targeting multiple Linux distributions for application development is not something that should be taken lightly.

The Linux Standard Base was created to solve these challenges and lower the overall costs of supporting the Linux platform. By reducing the differences between individual Linux distributions, the LSB greatly reduces the costs involved with porting applications to different distributions, as well as lowers the cost and effort involved in after-market support of those applications.

The Linux Foundation's mandate for the Linux Standard Base (LSB) is to enable ISVs to cost effectively target the Linux platform, reducing their porting, support, and testing costs, while helping them address a global market for their applications.



Benefits of the LSB

The LSB solution not only makes life easier for individual application developers and ISVs, it also makes a huge positive impact on the entire Linux ecosystem by allowing more applications to be widely introduced to the Linux operating system. The Linux Foundation believes that the more applications that are available for an operating system, the better and stronger the operating system is.

That's the long-term benefit of the LSB, but what is the specific benefit? There are actually two. The LSB directly helps vendors and community groups:

- Reduce the costs of porting an application from one Linux distro to another
- Reduce the costs of supporting a Linux application

Reducing Porting Costs

When developers first face the task of developing an application for Linux, they have a broad array of distributions to choose from. Because of the different choices made by distribution vendors to create a great Linux product, different versions of interfaces and libraries may be included in one distribution versus another. Some libraries may not be included at all.

For an application developer, this can be very challenging. In the past, the solution would be to bypass developing for Linux completely or to pick one of the more popular Linux distributions and accept that the application may not be available to a broader Linux user base. Neither of these choices is desirable, especially with the growth in current Linux use and the projection of a \$50 billion Linux ecosystem by 2011.

The good news is the choice of one distribution or another is no longer required. All of the major commercial Linux distributions are LSB compliant. Many of the core interfaces and libraries that an application needs to use are present in each LSB-compliant distro and are located in the same place in the operating system's filesystem. The LSB, then, reduces the number of different deployment requirements across distributions; the number of distro-unique interfaces and libraries an application has to work with is lowered, and portability is easier to accomplish. Figure 1 illustrates how this reduction works.

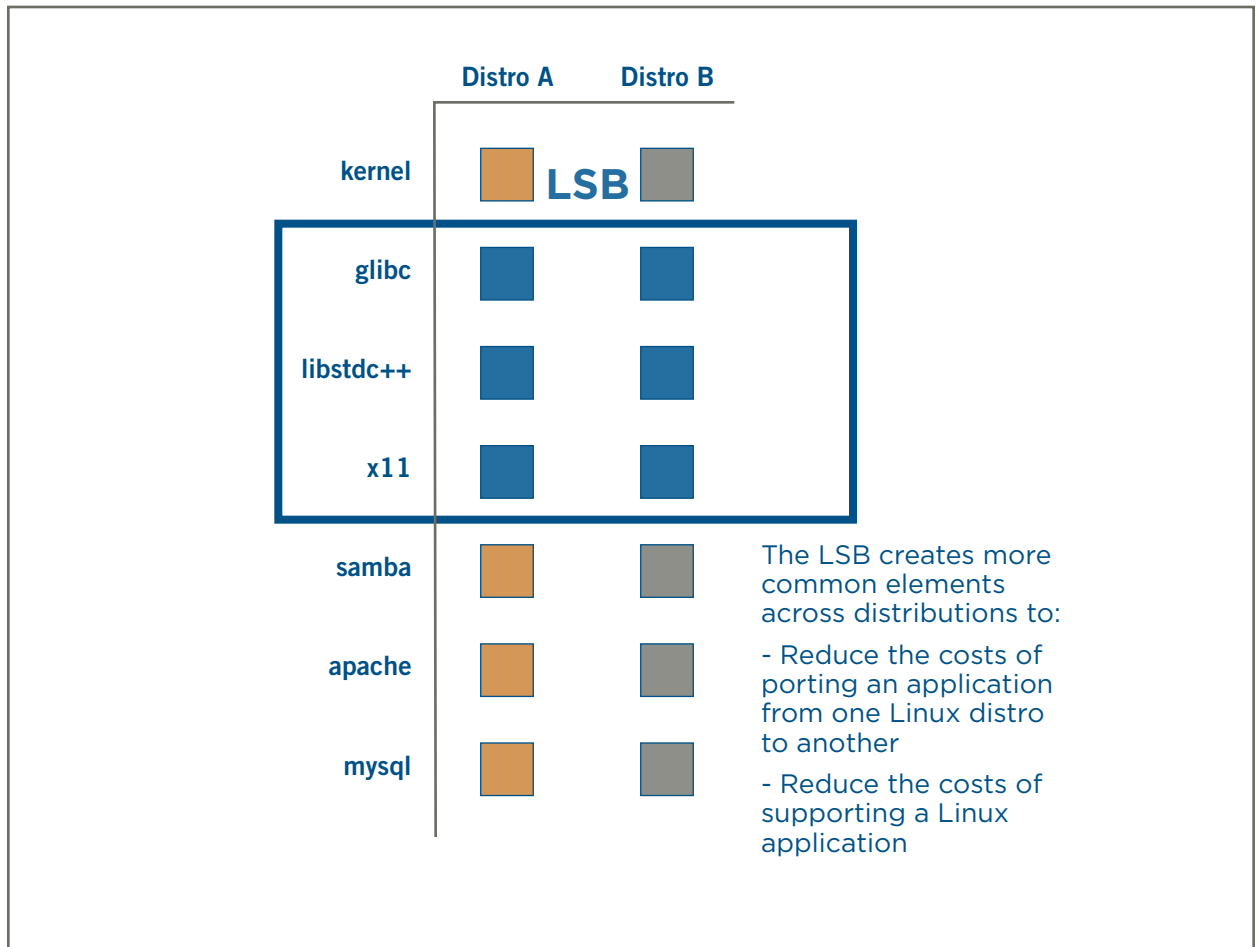


Figure 1: How LSB Reduces Costs of Development

Reducing Support Costs

If a vendor has developed an application for more than one platform, the demands of support for that application grow factorially. Distro-specific coding usually makes code harder to read. Changes in new versions of a distribution must be found and recoded for. Changes made for one distribution may not apply to others, so distribution-specific interactions must be more carefully maintained also.

This is again something the LSB is very good at solving. Since the interfaces and libraries are captured at a defined version level and have a core of the same functions, across the LSB-compliant distributions, identifying a problem on one Linux system means you're likely to identify the problem for all of them.

Other LSB Benefits

Beyond the reduction of porting and support costs, the very existence of the LSB solution means that Linux will also avoid the fate of the UNIX® operating systems, where commercial interests caused the fragmentation of a single UNIX OS into several UNIX variants that, unlike Linux distributions, are very incompatible with each other.

There are other solid benefits of the LSB:

- Greater opportunities for ISVs in an additional geographic market, where a different distribution may be more dominant than one originally supported by the ISV.
- The ability to support additional Linux distributions with only a small increase in support and development costs.
- Support from the Linux Foundation and the Linux Developer Network to make ISVs' development process easier and less costly and their marketing more effective.

This LSB balances the needs of the competitive distribution ecosystem with the requirements of end users and independent software vendors for portability.

What does the world fully enabled for LSB look like? In a real sense, we are starting to see it today.

- A healthy distribution network competing on support, service, security, price, and other factors based on ISV requirements for LSB compliance
- Broad availability of applications for Linux platforms, covering everything from the most robust and complicated data center systems to shrink-wrapped consumer applications available at retail outlets
- An open standard allowing ISVs to write their application to multiple Linux platforms at a low cost
- Reduced support costs for customers, ISVs and systems vendors since all have a clear set of application and distribution guidelines that guide their software and hardware development and delivery
- Reduced development costs for distribution vendors as a base set of commonality, leveraging multiple vendors and allowing them to focus on innovating at the unique higher levels of functionality

Clearly, ISVs, end users, and distribution vendors all benefit from a well-supported Linux standard. A robust and comprehensively supported standard--and services designed to increase Linux application development--will simplify the development process, delivering huge economies to the marketplace, and therefore very substantial incentives to build and use products and services that are based upon the Linux environment.

The Linux ecosystem (and all who depend upon it) thrives when Linux thrives, providing a truly open alternative to the proprietary computing platforms of old.

As you read through this document, it is very important to note that the LSB isn't a new, untried technology; all of the major Linux distributions certify to the LSB, including Red Hat, Novell's SUSE, and Canonical's Ubuntu. This is a mature technology that has, through the Linux Foundation, been supported by key vendors in the Linux ecosystem, such as IBM Corporation, NEC Corporation, Fujitsu, Oracle, Intel, Hewlett Packard, Hitachi, and Novell. Not only is it not an immature technology, the standard itself is a trailing standard, not a leading standard. It codifies well-established practice rather than inventing standards that may or may not be successful in future systems.



Examining the LSB

Understanding the components of the LSB is key to understanding all the things it can do for developers.

The Linux Standard Base is a core standard for the Linux operating system that encourages interoperability between applications and the platform. It includes a written binary interface specification; sets of test suites for both distributions and applications writing to the standard; and a sample implementation for other environmental testing purposes.

Currently, the distribution vendors are the key enablers of the LSB standard. Without their participation, the standard cannot achieve success. Without their participation in the creation of the LSB, their support for it would be less likely. The good news is that all of the major distribution vendors are indeed certified to the LSB.

In order for the LSB to be successful, the Linux Foundation has built an organization that facilitates consensus across technical, marketing and legal Linux groups while balancing the needs of other constituents to have a timely and useful standard. The LF supports conferences and other forums where ISV, customers and Linux distributors can pool their requirements and ideas in individual groups and also across these groups. The LF makes presentations in key venues where industries need information on LSB.

Community support is key, too. The open source community represents an amalgamation of software projects that are integrated into a single computing ecosystem. It is important that the maintainers of those projects be aware of existing computing standards such as the LSB so they can work in a cooperative fashion to accelerate widespread the adoption of their technology. All of these cooperative elements--distribution vendors, community developers, and application developers--work together to make the LSB happen. But what are the technical elements of the LSB? What's under the hood?

Checking Under the Hood

The key component of the LSB is the written binary interface specification, which informs developers of Linux applications the standard ways to build and configure their applications. The specification lists:

- Common Packaging and Install Guidelines
- Common Shared Libraries and their Versions
- Configuration Files
- File Placement
- System Commands
- Application Binary Interfaces (ABIs) for System Interfaces (both application and platform levels)

Built from a foundation of existing standards, LSB delineates the binary interface between an application and a runtime environment on a hardware architecture. Existing standards that the LSB draws from include the Single UNIX Specification (SUS), the Standard C++ ABI, and the System V ABI. Other standards referenced include Pluggable Action Modules (PAM), X11, and the desktop standards hosted on linuxfoundation.org.

LSB formalizes the framework for interface availability within individual libraries and itemizes the data structures and constants associated with each interface. These components include shared libraries required by developers, file system hierarchies (defining where files are located in the system), specifications for the behavior of public interfaces, application packaging details, application behavior pre- and post-installation, and so on. Supported languages include C, C++, Perl, Python, Tcl, and—with the arrival of LSB 4.0—Java.

Application Certification

When an application conforms to all of these specifications that are part of the LSB, it is well on its way to becoming LSB-certified. If application certification is your goal, it is recommended that you start by testing your application. You want to be testing a properly working application. There's no better way to know whether you're compliant, or how far away from compliance your application is. Testing can be done with native Linux system builds “right out of the box,” but that may introduce spurious errors in LSB checking. Developers are encouraged to use the LSB Software Development Kit (SDK) early in their development and testing process to avoid having a library inject unnecessary errors. After building with the LSB SDK, using the AppChecker will identify and then help solve any outstanding portability issues.

Beyond testing the application and verifying that the interfaces used are covered in the standard, certification also requires running the application using the LSB Sample Implementation. This activity checks whether other utilities outside of the standard are used in the application or whether dynamic calls are made at runtime that do not show up as externals that can be checked by the static application checking. In addition to using the LSB-si, the application needs to be demonstrated to run on two different LSB-certified distributions.

An equivalent, though dissimilar, set of steps are used for distribution developers who wish their product to become LSB-compliant.

The “Standard” in the LSB isn't just for show, either. The LSB is a full International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) standard, specifically ISO/IEC standard 23360. This means the LSB is and will always be maintained with the highest quality and will always be open.

Currently, the LSB supports seven architectures. This allows for robust competition among a variety of architectures without locking hardware vendors into any third party or proprietary vendor's software offering. The LSB is targeted to help developers on a variety of platforms, with the following architectures covered: Intel IA32 and IA64; x86-64/EM64T; IBM PPC 32 and PPC 64; IBM 31-bit S/390; and IBM 64-bit zSeries.

The Linux Foundation also coordinates testing and certification programs that verify software compliance with existing standards.

For instance, in the certification program, vendors can submit test results for official certification after registering their product with the certification system, signing the LSB Trademark License Agreement (TMLA), and paying the applicable fees. Once the TMLA has been signed and the fees paid, the Linux Foundation will audit the test results, and if the application or distribution passes, it will become fully LSB certified.



Examining Portability Strategies



The LSB was created to eliminate much of the heavy lifting required by the peculiarities of the Linux market. For ISVs, the LSB is a means to reduce the cost of supporting multiple distributions and multiple versions within a single distribution.

As you seek to create a new Linux application, or improve upon an existing app, trying to achieve the benefits of cross-distribution portability may seem daunting. Fortunately, there are a number of tools and strategies to help you meet this goal.

One strategy is to simply strive to create a fully LSB-compliant application. This involves using the LSB build tools and restricting the application to use only interfaces that are guaranteed to be present on all LSB-compliant distributions. There are advantages to using this strategy; the LSB-compliant application will likely work on all major distributions with only minor tweaks, and it will very likely continue to work on future versions of those distributions. However this strategy may not be right for all applications.

For some applications (especially those that touch the kernel), LSB compliance may not be achievable now or in the foreseeable future. That does not mean, however, that those applications have nothing to gain from following basic LSB guidelines. In fact, the LSB eliminates many of the complex moving parts that challenge application portability for those ISVs today.

This means that even if you decide not to try for full LSB compatibility, you can still achieve a much better degree of general portability for your application just by following more basic guidelines.

This alternate strategy is to use the powerful Linux Application Checker tool to test your application, and explicitly test your application on a number of distributions. Techniques and tools on the Linux Developer Network, described in the “Using the LSB” section, will provide tips and tricks on increasing your application’s general portability. The Linux Application Checker will also provide suggestions for more portable interfaces to use.

Keep in mind these two strategies are not mutually exclusive. Reducing the number of non-LSB libraries used by your application can be used in combination with portability techniques and targeting a strategic set of applications. This would ultimately maximize your applications’ portability and widen your potential end-user base.



Using the LSB

Now that you know what the LSB is and why it is such a great benefit to Linux and Linux application development, what's the best way to get started using it? This is what the Linux Developer Network (LDN) is all about.

First, what are your goals? Do you want to fully certify your application under the LSB, or is that not a choice you want to make at this time? While we encourage certification, we also recognize that not all ISVs will choose to certify to the LSB.

Even if an ISV chooses not to certify, however, that does not mean that those applications have nothing to gain from following LSB guidelines.

With this in mind, there are two basic paths for application developers to take:

- **Maximize Portability.** This solution path allows developers to use the Linux Application Checker provided on the LDN to determine how portable their application is right now. Key elements of the application such as incompatible libraries and interfaces that may hinder portability are identified, allowing developers to make changes that should make the application far more portable. The Linux Application Checker also makes recommendations and guides the developer down a path of best practices for developing their application.
- **LSB Certification.** Tools are available for developers of new Linux applications to build compliant applications from scratch. Tools such as the LSB Software Development Kit (LSB SDK), created and maintained by the LSB working group, consists of a build environment and the required tools to streamline porting software applications to conform with the LSB SDK.

In summary, resources available from the LDN include:

- **Linux Application Checker.** Check your existing application's portability and get recommendations to make it more portable.
- **LSB Database Navigator.** Identify the components of a developing application that may prevent portability and learn alternatives and solutions to achieve that portability. The database navigator is a wealth of information for C/C++ programmers looking to program on Linux.
- **LSB Build Tools.** The LSB SDK enables developers to validate the binaries and RPM packages to ensure LSB compliance. It also enables developers to monitor the API usage by the application while the build is taking place so that conformance is assured.
- **LSB Sample Implementation.** The LSB Sample Implementation (LSB-si), is a minimal LSB-conforming runtime environment used for testing purposes. LSB compliant applications should be tested inside the LSB-si to insure they haven't picked up any distribution-specific quirks. The LSB Certification program requires an application be tested under the LSB-si.

- **LSB Certification and General Marketing Support.** No matter which path you choose, LSB or portability, resources will be available for developers to determine how to maximize their application's exposure in the Linux ecosystem. Certified applications can be included in the product directory.
- **Tutorials and Blogs.** Get the latest howtos and information for application portability, LSB compliance, and general Linux application development.
- **Forums and Mailing Lists.** Get real-time solutions and tips to many development problems.

In the next sections, each of these LSB tools will be highlighted, to give you a better idea of how they can work for your organization.



In Detail: The Linux Application Checker

The Linux Application Checker (also referred to as “AppChecker”) is a powerful new tool designed to help software developers target Linux. It draws on the extensive testing framework developed by the Russian Academy of Sciences and the Linux Foundation and leverages the work of the LSB workgroup.

That’s the official version, but what does the tool’s functionality really mean to application developers who want to write apps for Linux? In a few words: ease of portability.

In the past, the best strategy for making a Linux application portable was to closely follow the LSB model as much as possible. But, as indicated, there are a number of reasons why application vendors might not want to go for full LSB compliance.

To provide as many choices as possible, the new AppChecker tool is for vendors who are not (yet) planning on pursuing LSB certification as well as those who are, as it analyzes and provides guidance on symbols and libraries that go beyond the LSB.

The “ease” in AppChecker starts from the very first exposure to the tool. After simply downloading the architecture-specific version of AppChecker, you merely unpack the tarball into a destination folder and run AppChecker from there. An embedded web server provides a universal interface for any architecture.

Once you point AppChecker to the binary for the application you want to test, the automated system compares the symbol table of the application’s interfaces and libraries to the known values within the 30 LSB-compliant distributions.

After the comparison is complete, a detailed report of which distributions your application should run on is provided. This app/distro matches give the developer a strong indication that, after some additional testing and a little tweaking, that application should run on the indicated matched distros, as shown in Figure 2.

THE LINUX FOUNDATION LINUX APPLICATION CHECKER

Application Check | Result History | Help | About Administration

Analysis Results for Firefox on x86 [Please upload info about your application to the LSB Navigator.](#)

Some compatibility problems detected

- There are 18 of 30 distributions (see below) that provide all the required libraries and interfaces.
- The Application uses 5 interfaces appeared in later version of LSB. Try to check it for compatibility with **LSB 4.0**.
- The Application uses 3 external libraries without using interfaces in those libraries. The Application would be more portable if it is fixed to not use unneeded libraries.

[Distribution Compatibility](#)
[Required Libraries](#)
[Required Interfaces](#)
[LSB Certification](#)

The table below shows compatibility status of your application with the distributions analyzed by the Linux Foundation. Your Application will run on the "green" distributions without loader problems. Compatibility with the "yellow" distributions can be easily achieved by excluding unneeded libraries from the dependencies of your application. Making the application compatible with the "red" distributions may require more efforts to avoid using missing libraries/interfaces or by supplying them as a part of your application package. Please note that functional correctness is not guaranteed by this analysis.

Summary	
Distribution	Status
Asianux Server 3.0	OK
BOSS 2.0	OK
Fedora 7	OK
Fedora 8	OK
Fedora 9	OK
Gentoo 2007.0	OK
Mandriva 2007_Spring	OK
Mandriva 2008	OK
Oracle Enterprise Linux 5	OK
RHEL 5	OK
SLES 10	OK
SLES 10 SP2	OK
Ubuntu 7.04	OK

Figure 2: AppChecker can reveal how portable an application is.

So, while a positive result from AppChecker does not guarantee that your application will run on all distros out-of-the-box, it can lead you down the path of portability and hopefully reduce your support and porting costs. This allows developers to target the largest market possible and provide choice to their customers without too high of a burden or risk.

The AppChecker really shines when a mismatch is found; it not only tells you there is a potential issue with a particular distribution, it also specifically identifies the missing or incompatible interfaces. Links are provided from these reported interfaces so the developer can dig deeply into exactly how to modify or replace the interface to maximize portability, if they wish, as shown in Figure 3.

THE LINUX FOUNDATION **LINUX APPLICATION CHECKER**

Application Check | Result History | Help | About Administration

Analysis Results for **Firefox on x86** [Please upload info about your application to the LSB Navigator.](#)

Some compatibility problems detected

- There are **18 of 30** distributions that provide all the required libraries and interfaces.
- The Application uses **5** interfaces appeared in later version of LSB. Try to check it for compatibility with **LSB 4.0**.
- The Application uses **3** external libraries without using interfaces in those libraries. The Application would be more portable if it is fixed to not use unneeded libraries.

Distribution Compatibility
 Required Libraries
 Required Interfaces
 LSB Certification

The table below lists all external interfaces required by your Application (based on ELF symbols). There is LSB and Distribution presence status for each interface.

Component: Library: Show: 41 interfaces shown.

Interface	Version	Dependent Application Components	In Libraries	Presence in Distributions	In LSB 3.2?	More Info
__libc_start_main	GLIBC_2.0	1 component (...)	libc.so.6	30 of 30 (list...)	Yes	Details...
__realpath_chk	GLIBC_2.4	1 component (...)	libc.so.6	18 of 30 (list...)	In LSB 4.0	Details...
__snprintf_chk	GLIBC_2.3.4	1 component (...)	libc.so.6	26 of 30 (list...)	In LSB 4.0	Details...
__sprintf_chk	GLIBC_2.3.4	1 component (...)	libc.so.6	26 of 30 (list...)	In LSB 4.0	Details...
__stack_chk_fail	GLIBC_2.4	1 component (...)	libc.so.6	18 of 30 (list...)	In LSB 4.0	Details...
__vprintf_chk	GLIBC_2.3.4	1 component (...)	libc.so.6	26 of 30 (list...)	In LSB 4.0	Details...
__xstat	GLIBC_2.0	1 component (...)	libc.so.6	30 of 30 (list...)	Yes	Details...
access	GLIBC_2.0	1 component (...)	libc.so.6	30 of 30 (list...)	Yes	Details...
closedir	GLIBC_2.0	1 component (...)	libc.so.6	30 of 30 (list...)	Yes	Details...
dclose	GLIBC_2.0	1 component (...)	libdl.so.2	30 of 30 (list...)	Yes	Details...
dlopen	GLIBC_2.1	1 component (...)	libdl.so.2	30 of 30 (list...)	Yes	Details...
dlsym	GLIBC_2.0	1 component (...)	libdl.so.2	30 of 30 (list...)	Yes	Details...
fclose	GLIBC_2.1	1 component (...)	libc.so.6	30 of 30 (list...)	Yes	Details...

Figure 3: The AppChecker can allow you to drill down to specific interfaces to enhance portability.

And, on top of all of this, the AppChecker still performs its original function--reporting on how far along an application is toward LSB compliance, if that is still a goal for the application vendor.



In Detail: LSB Database Navigator

The LSB Database is a central place for storing information about the LSB and the surrounding Linux ecosystem. This is, simply put, the authoritative reference tool of all of the distributions, interfaces, and libraries contained within the LSB specification. It is broader than the LSB, as it contains information on objects and libraries outside the LSB specification. This makes it an invaluable tool for any Linux application developer, regardless of how portable they want the application.

The LSB Navigator represents a web-based interface for the LSB Database. It allows developers and ISVs to drill down to the particular interface/library they want to learn more about. The Navigator will list those distributions where the object is used, whether the object is LSB approved, and other useful information about the object. To give an idea of how much depth of information the Navigator can reveal, Figure 4 displays the Navigator listing for the libGL library.

The screenshot shows the LSB Database Navigator interface. At the top, there are logos for 'THE LINUX FOUNDATION' and 'LSB DATABASE NAVIGATOR'. A search bar is present with a dropdown menu set to 'Interface' and a 'Search' button. Below the search bar is a navigation menu with tabs for 'LSB Elements', 'Distros and Apps', 'Workgroup Services', 'Help', and 'About'. A secondary menu lists 'Applications', 'Distributions', 'Components', 'Libraries', 'Commands', 'Classes', and 'Interfaces'. The main content area is titled 'Library 'libGL'' and shows the following details:

LSB Status: **Included**

Arch	Runtime Name	Appeared in	Mandatory since	Deprecated since	Withdrawn in	LibGroups	Interfaces
Generic	libGL.so.1	1.0	1.0			2 (list...)	451 (list...)

This library is assigned to LSB-Desktop module (LSB_Graphics submodule).

Presence in Distributions
All distributions registered in the database have this library.
▶ [Details](#)

Data completeness: This library is included in the [approved](#) list. This means that we have checked every uploaded distribution for this library presence.

Usage in Applications
There are **71** different versions of **70** distinct applications registered in the database that use interfaces from this library (list...).
There are **74** different versions of **73** distinct applications registered in the database that require this library (list...).

Copyright © 2007, 2008 Linux Foundation. All rights reserved.
LSB is a trademark of the Linux Foundation. Linux is a registered trademark of Linus Torvalds.

Figure 4: A Sample Listing from the LSB Navigator.

The Navigator can be used by Linux developers, Linux distribution vendors, and members of the LSB workgroup to browse, query, analyze, and submit various information.



In Detail: LSB Build Tool

In order to build LSB-compliant binaries and packages, tools that are not in the LSB specification are often required. The LSB Software Development Kit (SDK) supplies some of these tools and are designed to make it easier to generate LSB-compliant binaries and packages. The SDK contains the LSB Development Environment (shared library stubs, headers, `lsbcc`, and `lsbcc++`) for building LSB-compliant binaries.

Using the build environment, there are two main methods for generating LSB-compliant packages. With both methods, source code is compiled against LSB header files and linked against special stub libraries.

The primary method uses a wrapper around the compiler to apply the LSB build rules within a full host environment. This is the quickest way to try building a conforming application, and is useful if the build will require various other software that may be installed on the system. Since non-LSB libraries, except those supplied with the application, must be statically linked, the LSB compiler automatically transforms the command line to arrange this before passing it on to the regular C compiler. This means you need to make sure the necessary static libraries are installed--many systems by default install only the dynamic versions. The downside of using the LSB-provided compiler is that it is easier to fool; complex build procedures, especially those that use a tool (such as `libtool` or `pkg-config`) to locate elements to link, will often bypass the compiler's rewriting rules.

The alternate method uses `chroot` to run in a restricted environment that excludes most of the non-LSB functionality. This is a safer way to build conforming binaries, as it's almost impossible to pull in non-LSB components by accident. The downside is that many builds require additional software beyond what is in the LSB for building, and setting this up will be a bit more work. However, this setup usually only needs to be done once.

Since no build tool can do a perfect job at assuring LSB conformance, a test tool is also provided.



In Detail: LSB Sample Implementation

The LSB-si is a limited test environment and a proof-of-concept of building to the LSB spec. Contrary to how it might seem, it is not a stand-alone Linux distribution.

Most Linux distributions supply lots of packages, and which of them gets installed, and later added, removed or upgraded is in the local user's or sysadmin's control. This is all well and good, except when a developer has to build a portable application that will run on any Linux. How can the developer be sure she hasn't linked in something that is not guaranteed to be present on every system upon which it will be installed?

The solution is clear, if the developer takes advantage of the contract LSB conforming systems offer: a core set of libraries and programs will be present, and will work in the way that is documented in the LSB.

The LSB-si is a minimal implementation of that set that can be used to test programs before releasing them to the field.

For example, some software has auto-detecting build scripts (such as GNU configure), which will enable the build to make use of many things installed on a system, whether those are part of the LSB. Running the resulting binary in the LSB-si will help detect cases where non-LSB features have crept in, as those features will likely not be present in the LSB-si.

The LSB-si also serves as a proof-of-concept of building an LSB conforming system. The policy is to use released packages from the package maintainers, with a truly minimal set of patches. Some patches are necessary to build in the LSB-si environment (often to account for the absence of some tool that is assumed by the build). A few patches are supplied to fix bugs or add LSB conformance, but generally the preference is for such patches to be picked up by the upstream maintainers and folded into their next release.

Finally, the LSB-si is able to stay in lockstep with the evolving LSB specification. New features appear in test versions of the LSB-si as they are finalized into the specification, and the LSB-si aims to release versions matching fully approved new specifications at roughly the same time as the spec. This gives developers the opportunity to preview new features before they become a fully-supported part of their favorite distributions.



In Detail: LSB Certification and General Marketing Support

The Linux Foundation offers certification for distributions and applications that comply with the LSB specification. Only certified products are permitted to use the LSB Certified trademark. This mark assures developers and end-users that any “LSB Certified” application will work correctly on every “LSB Certified” distribution.

When you are ready to submit your test results for official certification, you will need to register your application with the certification system. This can be done online, or within the AppChecker. Information gathered about the application or private data about the applicant will not be saved or used in any way beyond the certification process.

After your product is registered, you will be able to upload your test results and sign up for the LSB Trademark License Agreement (TMLA).

Once the TMLA has been signed and the fees paid, the Linux Foundation will audit your test results. (If the test results do not pass the audit, you will need to fix any problems uncovered by the audit, retest your product, and resubmit your test results.) Any tests you have failed that have already been waived will be marked as such in the AppChecker.

Once the product has passed the audit, it will be entered into the register of LSB Certified products. ISVs are encouraged to use the trademarked LSB Certified name and logo on their products to inform users of the product's proven compatibility.



Looking Toward the Future

The Linux Standard Base--supported by compliance testing and a certification program --is the firewall that we can build that will protect vendors and users alike from both deliberate as well as negligent degradation and fragmentation of Linux.

Our vision is to achieve the goal of increased portability while being sensitive to the needs of the distribution companies; the LSB aims to look at only what needs to be standardized to minimize the effort required by ISV's to market across Linux yet allows the distribution companies to differentiate and add value.

The Detailed Roadmap

The LSB aims to provide a “highest common denominator” across the various Linux distributions---in other words, to provide a single target for ISVs writing or porting to the Linux platform, where “the Linux platform” is defined by a short (and potentially different from ISV to ISV) list of distributions on which their applications must run.

To serve as an effective highest common denominator, it needs to be easy to map from LSB versions to distributions and vice versa; and it needs to be possible to target a version of the LSB with assurance that the application will work not only on that version but on future versions as well (i.e., an LSB-3.0 application will run on LSB 3.0-, 3.1-, 3.2-, and 4.0-compliant distributions).

To satisfy the “easy mapping” requirement, each major version of the LSB corresponds to a major version, or “generation”, of the enterprise distributions. So, for example, LSB 3.x corresponds to the previous generation (Red Hat Enterprise Linux 4, SUSE Linux Enterprise 9, etc.), whereas LSB 4.x corresponds to the current generation (RHEL 5, SLES 10, etc.), etc. To satisfy the application compatibility requirement, LSB versions both major and minor beginning with 3.0 are strictly backward compatible with previous versions.

At a high level, then, the LSB roadmap looks something like Table 1.

LSB 3.x (2006-2008)	LSB 4.x (2008-2010)
Asianux 2.0	Asianux 3.0
Debian 4.0 (“etch”)	Mandriva Corporate 5.0
Mandriva Corporate 4.0	Red Hat Enterprise Linux 5
Red Hat Enterprise Linux 4 and 5	SUSE Linux Enterprise 10
SUSE Linux Enterprise 9 and 10	Ubuntu LTS 8.04
Ubuntu 6.06 LTS	[...]
[...]	

Table 1: The LSB Roadmap for Distributions

Currently, preparations are being made to finalize the release of LSB 4.0. This new specification will include some exciting new features, such as:

- **A new sample implementation:** The 4.0 SI will be based on an entirely new toolset: the rPath’s Conary and rMake frameworks. This will be an entirely new way of building the SI, starting from scratch and porting over little pieces of the old LFS SI as needed into the new framework.
- **A revised LSB cryptography implementation:** NSS will be part of the new standard, moving away from OpenSSL.
- **Improved application and distribution test tools:** the new Linux AppChecker and an updated dist-check tool will enable developers to more easily obtain LSB 4.0 compliance.
- **New multi-version tools:** It should be possible to build to multiple versions of the LSB without re-installing the SDK. Optionally, new tools will identify which version of the LSB is needed by the application, and build appropriately.
- **A new dynamic linker:** The ELF dynamic linker will be the same as for all other binaries on the system, but the LSB SDK will embed code into the executable--either via crt1.o or via an init function called early--that checks if the executable needs to be run with the LSB dynamic linker instead, and re-execs the binary if necessary.
- **A specification for Java:** A new specification for describing Java in the LSB will be included.



Summary

The benefits of the LSB for ISVs are clear. By leveraging all of the advantages of the LSB, ISVs and individual developers can reduce their porting, testing, and support costs, while improving the marketability of their applications. We urge all ISVs to make use of the resources offered by the Linux Foundation. Not only will it help ISVs, it will also enhance the Linux platform and change the history of computing.

You can learn more about the Linux Foundation and the LSB at www.linuxfoundation.org and ldn@linuxfoundation.org.

The Linux Foundation

1796 18th Street, Suite C
San Francisco, CA 94107
Phone/Fax: +1-415-723-9709
<http://www.linuxfoundation.org>
info@linuxfoundation.org

Linux Foundation Japan

3-30 Kioicho, Yamamoto Building 5F
Chiyoda-ku, Tokyo, Japan, zip 150-0043
Phone: +81-3-3288-5651
Fax: +81-3-3288-5691
<http://www.linux-foundation.jp>