EL/IX Base API Specification
DRAFT - V1.2

Nick Garnett - Red Hat Inc.

September 18, 2000

1 Introduction

> **Note:** This is a draft document that is being distributed to gather public comments. Comments should be sent to the **EL/IX** discussion list at
>
> http://sources.redhat.com/elix.
>
> This website also provides access to the latest versions of this API specification together with other documents about **EL/IX** and archives of the mailing lists.

This document describes the proposed API for **EL/IX** . The goals for this API are:

1. Support development of embedded applications using the Linux desktop environment as both host and target platforms.
2. Provide scalability of that functionality according to the requirements of the embedded application.
3. Provide portability of applications between operating systems that provide the same levels of API functionality.
4. Provide real time functionality on real time operating systems.
5. Respect existing standards by adopting their APIs and functionality where appropriate, but also adopt a pragmatic approach to compatibility.

EL/IX achieves these goals by identifying a subset of the POSIX.1 and ISO C standards, together with some extensions gleaned from Linux/GNU, BSD and SYSV, that are applicable to embedded applications. The result is an API that is somewhat smaller than the union of those standards that eliminates unnecessary or duplicated functionality fairly ruthlessly.

2 Levels and Options

The following API levels are defined and each function in the API is assigned to a minimum level. Functions are present in that level and all higher levels.

Level 1 RTOS compatible layer. Functions available in both Linux and a typical deeply embedded operating system (**eCos** , RTEMS, VxWorks, pSOS, VRTX32 etc.). Some functions may have reduced or modified semantics.

Level 2 Linux single process only. Includes level 1 plus any functions from Linux that are not easily implemented on an RTOS. Also ”full” implementations of reduced functions in Level 1.

Level 3 Linux multiprocess for embedded applications. This is basically POSIX.1 with some of the functions that are obviously not for embedded applications (such as job control) removed.

Level 4 Full POSIX or Linux compliance. Essentially these are functions that are present in a standard Linux kernel but are irrelevant to an embedded system. These functions do not form part of the **EL/IX** API.

The following option letters are added to the level numbers for some API functions when the presence of that function is also controlled by some other component (such as the presence of a file system), or is an optional component that not all implementations provide.

o Optional at the given level. Probably standard part of higher levels.

m Modified semantics. Certain aspects of behaviour may be omitted as a result of restrictions in the underlying operating system. In general behaviour is only modified by omitting functionality rather than by changing behaviour in non-compatible ways. This is mainly only applicable at level 1, where some embedded operating systems do not contain facilities needed for some functions, although it is also possible for functionality to be omitted at higher levels as a result of Linux kernel configuration options.

s Signal handling. Optional support for signals.

f File system support. Only present if there is support for a file system (whether ROM, RAM, FLASH, floppy, HD, NFS etc.). Level 1 file systems will not have all the features. Level 2 and above are assumed to have at least a RAM file system with the full feature set.

In some places this option is additionally annotated with sub-options (in square brackets) that indicate the properties that the file system must support for the given function.

- **d** Directory support, including listing, as part of file system names and as settable working directories.
- **m** Modifiable file system, this would not be necessary, for example, for a ROM file system.
- **p** Permission support, including user and group ids on files and permission masks for managing access.
- **l** Link support, including both hard and symbolic links.

t Terminal support. Only present if there is support for terminal devices. This may include the provision of stdin/stdout for single and multiprocess applications. Even though level 1 will not support the full terminal functionality from POSIX.1, there will still be serial devices whose baud rate and other properties need to be controlled.

n Network support. Only present when networking support in the form of a TCP/IP stack is present.

M Memory management support. Only present if support for memory locking, mapping and protection are present. While some of these interfaces are present at level 1, their full functionality will normally only be available at higher levels. This option is divided into a number of sub-options:

- **l** Memory locking functions. At level 1 these are essentially no-ops and are present only for upward compatibility and POSIX.13 compatibility.

p Memory protection functions.

f Memory mapped files.

o Memory mapped objects. At level 1 these are present to support the definition of shared memory objects that represent memory mapped IO device registers.

r Real Time Support. This does not really constitute an option, but rather marks the subset of the **EL/IX** functions that can be expected to have real time characteristics. Functions that are not marked in this way cannot be expected to be real time, although they may have such properties on some platforms.

l Library-only functions. These are functions that depend on no external services: mainly C library math, char, string and array utility functions. Supplying these functions has no effect on an RTOS or Linux kernels, and they will only affect an application if they are actually used. Hence the presence of these functions is (mostly) benign and there is little to be gained by removing any of them.

c Compatibility function. These are functions that are typically present in POSIX or ISO C but which are either not applicable to an embedded application, or whose functionality is already covered by some other part of the API. Such functions will optionally be provided as stubs that return an error, are minimally functional implementations, or implemented in terms of functions that are part of **EL/IX** .

3 POSIX Standards

3.1 Realtime Profiles - 1003.13

In September 1999 the IEEE published standard 1003.13 "Standardized Application Environment Profile - POSIX Realtime Application Support (AEP)". This is a collection of four subsets of the POSIX.1 standard aimed at real time systems. This section examines the relationship between **EL/IX** and POSIX.13.

POSIX.13 presents four real time system profiles. The Minimal Realtime System Profile provides the basic set of functionality for a single process deeply embedded system. The Realtime Controller System Profile extends the minimal profile with support for a file system and asynchronous I/O. The Dedicated Realtime System Profile extends the minimal profile with support for multiple processes, but has more primitive support for file systems than the controller profile. Finally the Multi-Purpose Realtime System Profile is a superset of the other profiles and essentially consists of the entire POSIX.1, POSIX.1b and POSIX.1c standards.

These four profiles are superficially similar to our four levels although the controller and dedicated profiles are mutually exclusive in some areas. To ensure maximum compatibility, the EL/IX levels have been adjusted to allow each of the POSIX.13 profiles to be a complete subset of the closest matching level.

Each of the POSIX.13 profiles may be manufactured out of the matching **EL/IX** level by enabling certain options:

Minimal Profile This is essentially just **EL/IX** level 1 with the signal and some of the memory management options enabled.

Controller Profile This is level 2 with the signal, memory management, and file system options enabled.

Dedicated Profile This is level 3 with the signal, and some memory management options enabled.

Multi-Purpose Profile This is essentially exactly equivalent to level 4 with the GNU, BSD and SYSV compatibility functions removed.

3.2 POSIX 1003.1-200x

The Austin Common Standards Revision Group has been working on a revision of the POSIX standard that includes most of the subsidiary standards (1003.1j, Advanced Realtime Extensions, being the most relevant) together with the Open Group's Single UNIX Specification and ISO C99. This standard is due for delivery in Q2 2001. Based on a reading of Draft 4 (released August 2000) the **EL/IX** API has been adjusted to take this new standard into account. However, since much of the new functionality in POSIX 1003.1-200x does not yet appear in Linux, the **EL/IX** API remains, at present, a subset of that standard.

4 POSIX.1 Compatibility

The base of **EL/IX** functionality is the POSIX.1003.1-1996 Specification (ISO/IEC 9945-1).
This document follows the sections of the POSIX.1 specification. The reader is referred to that document for details of the syntax and semantics of the functions described here.

4.1 Process Primitives

4.1.1 Creation and Execution

Function	Level	Options	Notes
`fork()`	3		
`execl()`	3		
`execv()`	3		
`execle()`	3		
`execve()`	3		
`execlp()`	3		
`execvp()`	3		
`pthread_atfork()`	3		

4.1.2 Process Termination

Function	Level	Options	Notes
`wait()`	3		
`waitpid()`	3		
`_exit()`	3		

4.1.3 Signals

Signals were introduced into UNIX as a mechanism for delivering exceptions and asynchronous events to single threaded processes. Like `select()` and `poll()` (see later) they are largely unnecessary in a multi-threaded environment. Additionally signal handling has subtle and pervasive effects on many parts of the POSIX standard, not all of which are relevant to, or implementable in, the kind of small footprint embedded operating system that is represented at level 1. Hence the exact functionality available at level 1 will depend on the capabilities of the underlying operating system.
These first few entries describe areas of functionality that are orthogonal to the actual functions.

Function	Level	Options	Notes
Signal masks	1	s m o	1
EINTR wakeup	1	s m o	2
Notification mechanisms	1	s m o	3
Real time signals	1	s o	
Translate exceptions to signals	1	s m o	4
`kill()`	1	s m r	
`sigemptyset()`	1	s m r	
`sigfillset()`	1	s m r	
`sigaddset()`	1	s m r	
`sigdelset()`	1	s m r	
`sigismember()`	1	s m r	
`sigaction()`	1	s m r	
`pthread_sigmask()`	1	s m	1
`sigprocmask()`	1	s m r	
`sigpending()`	1	s m r	
`sigsuspend()`	1	s m r	
`sigwait()`	1	s m	
`sigwaitinfo()`	1	s m	
`sigtimedwait()`	1	s m	
`sigqueue()`	1	s m	3
`pthread_kill()`	1	s m	1

Note 1: At level 1 signal delivery and masking may be restricted to operating at a global level only. Per-thread signal masks, and delivery of signals to specific threads may be restricted by the capabilities of the underlying operating system.
Note 2: This is the termination of many POSIX API calls on delivery of a signal to the thread or process. At level 1 this may require considerable extra code to safely terminate and clean up these functions. Thus this functionality is dependent on the implementation.
Note 3: Some signal-generating functions (timer expiration, asynchronous I/O, message arrival and `sigqueue()`), allow specification of an event notification mechanism using a `sigevent` structure. This allows the event to be notified either by delivery of a signal, or by the immediate invocation of a function in a new thread. At level 1, one or other of these mechanisms may not be present, depending on the ability of the operating system to deliver signals and its ability to create threads dynamically.
Note 4: If a level 1 operating system has the ability to catch synchronous exceptions (divide-by-zero, illegal instruction, FPU exceptions, address/alignment exceptions etc.) then it may be possible to translate these into POSIX signals for delivery to the current thread.

4.1.4 Timer Operations

Function	Level	Options	Notes
`alarm()`	1	s m r	
`pause()`	1	s m r	
`sleep()`	1	m	

4.2 Process Environment

Some of these may be useful to aid porting of existing code, hence they are not all just marked level 3 or 4.

4.2.1 Process Identification

Function	Level	Options	Notes
`getpid()`	1	c	1
`getppid()`	1	c	1

Note 1: At level 1 these functions simply return a constant value.

4.2.2 User Identification

Function	Level	Options	Notes
`getuid()`	1	c	1
`geteuid()`	1	c	1
`getgid()`	1	c	1
`getegid()`	1	c	1
`setuid()`	4		
`setgid()`	4		
`getgroups()`	4		
`getlogin()`	4		
`getlogin_r()`	4		

Note 1: At level 1 these functions simply return a constant value.

4.2.3 Process Groups

Function	Level	Options	Notes
`getpgrp()`	4		
`setsid()`	4		
`setpgid()`	4		

4.2.4 System Identification

Function	Level	Options	Notes
`uname()`	1	m	

4.2.5 Time

Function	Level	Options	Notes
`time()`	1	r	
`times()`	1	m	1

Note 1: At level 1 only some of the return values in the `struct tms` buffer will be filled in. At level 2 there will never be any child processes to be reported in the `tms_cutime` and `tms_cstime` fields.

4.2.6 Environment Variables

Possibly have a configuration- or build- time defined static environment to support this.

Function	Level	Options	Notes
`getenv()`	1		

4.2.7 Terminal Identification

Function	Level	Options	Notes
`ctermid()`	2	t	
`ttyname()`	2	t	
`ttyname_r()`	2	t	
`isatty()`	1	f t n	

4.2.8 Configurable System Variables

Under a level 1 operating system, only some of these variables will be implemented.

Function	Level	Options	Notes
`sysconf()`	1	m r	

4.3 Files and Directories

In many cases the functionality of these functions will be dictated by the properties of the underlying file system rather than by the operating system.

4.3.1 Directories

Function	Level	Options	Notes
`opendir()`	1	f[d] m	
`readdir()`	1	f[d] m	
`readdir_r()`	1	f[d] m	
`rewinddir()`	1	f[d] m	
`closedir()`	1	f[d] m	

4.3.2 Working Directory

Function	Level	Options	Notes
`chdir()`	1	f[d]	
`getcwd()`	1	f[d] o	1

Note 1: The functionality of `getcwd()` may depend on the functionality of the underlying filing system. If the filesystem infrastructure does not track the current directory path, then finding the CWD requires either filesystem support for traversing the directory upwards, or the presence of ".." directory entries, neither of which are a given.

4.3.3 General File Creation

Function	Level	Options	Notes
`open()`	1	f t m	1
`creat()`	1	f[m]	
`umask()`	2	f[mp]	
`link()`	1	f[ml]	

Note 1: At level 1 only a subset of the flags may be supported, and then only if the underlying file system or device provides support.

4.3.4 Special File Creation

Function	Level	Options	Notes
`mkdir()`	1	f[md]	
`mkfifo()`	3	o f[m]	

4.3.5 File Removal

Function	Level	Options	Notes
`unlink()`	1	f[m]	
`rmdir()`	1	f[md]	
`rename()`	1	f[m]	

4.3.6 File Characteristics

Function	Level	Options	Notes
`stat()`	1	f m	1
`fstat()`	1	f m	1
`access()`	1	f	
`chmod()`	2	f[mp]	
`fchmod()`	2	f[mp]	
`chown()`	2	f[mp]	
`utime()`	2	f	
`ftruncate()`	4		
`pathconf()`	2	m	
`fpathconf()`	2	m	

Note 1: At level 1 some fields in the `struct stat` structure may not be filled in, this may also depend on the supporting file system.

4.4 Input and Output Primitives

As with the previous section, the properties of the underlying filesystems or devices will mostly dictate the functionality available here.

4.4.1 Pipes

Function	Level	Options	Notes
`pipe()`	3		

4.4.2 File Descriptor Manipulation

Function	Level	Options	Notes
`dup()`	1	f t n	
`dup2()`	1	f t n	
`close()`	1	f t n	

4.4.3 Input and Output

Function	Level	Options	Notes
`read()`	1	f t n	
`write()`	1	f t n	

4.4.4 Control Operations on Files

Function	Level	Options	Notes
`fcntl()`	1	m	1
`lseek()`	1	f	

Note 1: At level 1 only `F_DUPFD` need be supported. The remaining functionality depend mainly on the implementation of the underlying filesystem.

4.4.5 File Synchronization

Function	Level	Options	Notes
`fsync()`	1	f	
`fdatasync()`	1	f	

4.4.6 Asynchronous Input and Output

Completion of asynchronous IO operations is notified by invoking a `sigevent` object. In certain configurations this may be restricted to only specifying one of SIGEV_SIGNAL or SIGEV_THREAD as the notification type.
Asynchronous IO is largely unnecessary in a multi-threaded environment. Any implementation of these facilities may be expensive and exhibit poor performance. Hence this is an optional facility whose use is deprecated.

Function	Level	Options	Notes
`aio_read()`	1	o f t n	
`aio_write()`	1	o f t n	
`lio_listio()`	1	o f t n	
`aio_error()`	1	o f t n	
`aio_return()`	1	o f t n	
`aio_cancel()`	1	o f t n	
`aio_suspend()`	1	o f t n	
`aio_fsync()`	1	o f t n	

4.5 Device- and Class-Specific Functions

These are only present to control the behaviour of a serial line. Even then, most of the line editing and translation functionality will be omitted (at level 1 at least) and all job control functionality is banished to level 4.

Function	Level	Options	Notes
`tcgetattr()`	1	t	
`tcsetattr()`	1	t	
`cfgetospeed()`	1	t	
`cfgetispeed()`	1	t	
`cfsetospeed()`	1	t	
`cfsetispeed()`	1	t	
`tcsendbreak()`	1	t	
`tcdrain()`	1	t	
`tcflush()`	1	t	
`tcflow()`	1	t	
`tcgetpgrp()`	4		
`tcsetpgrp()`	4		

4.6 C Language Services

POSIX specified modifications and enhancements to the ISO C API.

Function	Level	Options	Notes
`setlocale()`	1	1	1
`fileno()`	1	f t n	
`fdopen()`	1	f t n	
`flockfile()`	1		
`ftrylockfile()`	1		
`funlockfile()`	1		
`getc_unlocked()`	1		
`getchar_unlocked()`	1		
`putc_unlocked()`	1		
`putchar_unlocked()`	1		
`sigsetjmp()`	1	s r	
`siglongjmp()`	1	s r	
`tzset()`	1	o	1
`strtok_r()`	1		
`asctime_r()`	1		
`ctime_r()`	1		
`gmtime_r()`	1		
`localtime_r()`	1		
`rand_r()`	1		

Note 1: Locale support may be limited to just the "C" locale and maybe the "POSIX" locale.
Note 2: At level 1, and maybe others, only a very reduced time zone implementation need be provided. This will probably rely on external information (such as configuration-, build-, install-time options or user input) to determine the UTC offset, DST change dates and time zone abbreviations.

4.7 System Databases

Access to group and password databases.

Function	Level	Options	Notes
`getgrgid()`	4		
`getgrgid_r()`	4		
`getgrnam()`	4		
`getgrnam_r()`	4		
`getpwuid()`	4		
`getpwuid_r()`	4		
`getpwnam()`	4		
`getpwnam_r()`	4		

4.8 Data Interchange Format

This section of POSIX.1 describes the `tar` and `cpio` formats for data archives. These are irrelevant for embedded applications.

4.9 Synchronization

4.9.1 Semaphore Functions

Function	Level	Options	Notes
`sem_init()`	1	r	
`sem_destroy()`	1	r	
`sem_open()`	4		1
`sem_close()`	4		1
`sem_unlink()`	4		1
`sem_wait()`	1	r	
`sem_trywait()`	1	r	
`sem_post()`	1	r	
`sem_getvalue()`	1	r	

Note 1: These functions can be supplied at lower levels if kernel support is present, or if a fake namespace for them to operate in is implemented.

4.9.2 Mutexes

Function	Level	Options	Notes
`pthread_mutexattr_init()`	1	r	
`pthread_mutexattr_destroy()`	1	r	
`pthread_mutexattr_getpshared()`	3	r	
`pthread_mutexattr_setpshared()`	3	r	
`pthread_mutex_init()`	1	r	
`pthread_mutex_destroy()`	1	r	
`pthread_mutex_lock()`	1	r	
`pthread_mutex_trylock()`	1	r	
`pthread_mutex_unlock()`	1	r	

4.9.3 Condition Variables

Function	Level	Options	Notes
`pthread_condattr_init()`	1	r	
`pthread_condattr_destroy()`	1	r	
`pthread_condattr_getpshared()`	3	r	
`pthread_condattr_setpshared()`	3	r	
`pthread_cond_init()`	1	r	
`pthread_cond_destroy()`	1	r	
`pthread_cond_signal()`	1	r	
`pthread_cond_broadcast()`	1	r	
`pthread_cond_wait()`	1	r	
`pthread_cond_timedwait()`	1	r	

4.10 Memory Management

4.10.1 Memory Mapping Functions

Function	Level	Options	Notes
`mlockall()`	1	M[l]	
`munlockall()`	1	M[l]	
`mlock()`	1	M[l]	
`munlock()`	1	M[l]	
`mmap()`	1	M[fo]	
`munmap()`	1	M[fo]	
`mprotect()`	2	M[p]	
`msync()`	2	M[f]	

4.10.2 Shared Memory Functions

Function	Level	Options	Notes
`shm_open()`	1	M[op]	
`shm_unlink()`	1	M[o]	

4.11 Execution Scheduling

At level 1 we define SCHED_OTHER to equal SCHED_RR. This equality is allowed by POSIX.1.

4.11.1 Process Scheduling Functions

Function	Level	Options	Notes
`sched_setparam()`	3	r	
`sched_getparam()`	3	r	
`sched_setscheduler()`	3	r	
`sched_getscheduler()`	3	r	
`sched_yield()`	1	r	
`sched_get_priority_max()`	1	r	
`sched_get_priority_min()`	1	r	
`sched_rr_get_interval()`	1	r	

4.11.2 Thread Scheduling

Function	Level	Options	Notes
`pthread_attr_setscope()`	1	r	1
`pthread_attr_getscope()`	1	r	1
`pthread_attr_setinheritsched()`	1	r	
`pthread_attr_getinheritsched()`	1	r	
`pthread_attr_setschedpolicy()`	1	r	
`pthread_attr_getschedpolicy()`	1	r	
`pthread_attr_setschedparam()`	1	r	
`pthread_attr_getschedparam()`	1	r	
`pthread_getschedparam()`	1	r	
`pthread_setschedparam()`	1	r	

Note 1: At levels 1 and 2 only the only scope permitted may be `PTHREAD_SCOPE_SYSTEM`.

4.11.3 Synchronization Scheduling

At level 1, depending on the capabilities of the underlying operating system, one or both of PTHREAD_PRIO_INHERIT or PTHREAD_PRIO_PROTECT may not be implemented.

Function	Level	Options	Notes
`pthread_mutexattr_setprotocol()`	1	r	
`pthread_mutexattr_getprotocol()`	1	r	
`pthread_mutexattr_setprioceiling()`	1	r	
`pthread_mutexattr_getprioceiling()`	1	r	
`pthread_mutex_setprioceiling()`	1	r	
`pthread_mutex_getprioceiling()`	1	r	

4.12 Clocks and Timers

Note that timers indicate that they have expired by raising a signal. This implies that some form of signal support be present all the time, or that the presence of timers is also dependent on the presence of signal handling. Alternatively, the sigevent structure supplied to `timer_create()` may only specify SIGEV_THREAD as the notification type.
There is also some question over the presence of these functions in current Linux releases.

Function	Level	Options	Notes
`clock_settime()`	1	r	
`clock_gettime()`	1	r	
`clock_getres()`	1	r	
`timer_create()`	1	s r	
`timer_delete()`	1	s r	
`timer_settime()`	1	s r	
`timer_gettime()`	1	s r	
`timer_getoverrun()`	1	s r	
`nanosleep()`	1	r	

4.13 Message Passing

These are more oriented towards inter-process communications rather than intra-process, where the program should be using semaphores, mutexes etc. At level 1 it may be necessary to provide a fake name space for the `mq_open()`, and `mq_unlink()` functions to work within.

Function	Level	Options	Notes
`mq_open()`	1	r o	
`mq_close()`	1	r o	
`mq_unlink()`	1	r o	
`mq_send()`	1	r o	
`mq_receive()`	1	r o	
`mq_notify()`	1	r o	
`mq_setattr()`	1	r o	
`mq_getattr()`	1	r o	

4.14 Thread Management

Function	Level	Options	Notes
pthread_attr_init()	1	r	
pthread_attr_destroy()	1	r	
pthread_attr_setstacksize()	1	r	
pthread_attr_getstacksize()	1	r	
pthread_attr_setstackaddr()	1	r	
pthread_attr_getstackaddr()	1	r	
pthread_attr_setdetachstate()	1	r	
pthread_attr_getdetachstate()	1	r	
pthread_create()	1	r	
pthread_join()	1	r	
pthread_detach()	1	r	
pthread_exit()	1	r	
pthread_self()	1	r	
pthread_equal()	1	r	
pthread_once()	1	r	

4.15 Thread Specific Data

Function	Level	Options	Notes
pthread_key_create()	1	r	
pthread_setspecific()	1	r	
pthread_getspecific()	1	r	
pthread_key_delete()	1	r	

4.16 Thread Cancellation

At level 1, one or both of PTHREAD_CANCEL_ASYNCHRONOUS and PTHREAD_CANCEL_DEFERRED may not be implemented, depending on the capabilities of the underlying operating system.

Function	Level	Options	Notes
pthread_cancel()	1	o r	
pthread_setcancelstate()	1	o r	
pthread_setcanceltype()	1	o r	
pthread_testcancel()	1	o r	
pthread_cleanup_push()	1	o r	
pthread_cleanup_pop()	1	o r	

5 C Library Compatibility

The list of functions here is based the documentation for Glibc V2.1, cross checked with the Linux headers and the ISO C 1999 standard.

A slight difficulty with non-ISO C-standard functionality here is that while on Linux glibc is available, it is not always available on an embedded operating system due to the viral nature of GPL. The demands of Linux compatibility require the C library functionality to be compatible with Glibc, which is primarily ISO C99 compatible. However, the C libraries for many embedded operating systems are only ISO C89 compatible, and much that is present in C99 is not applicable to an embedded environment. Hence at level 1 only C89 compatibility is required, although if C99 compatibility is available it may be used. The distribution of functionality between levels 1 and 2 attempts to this into account.

Where the phrase “ISO C” is used in this document it should usually be read as “ISO C89” at level 1 and “ISO C99” at higher levels.

5.1 Error Reporting

Conversion of error codes to strings: `strerror()` and `perror()`. `strerror_r()` is a GNU extension.
The main problem with these functions is the memory needed to store the array of strings, hence these must be optional.

Function	Level	Options	Notes
`strerror()`	1	o l	
`strerror_r()`	1	o l	
`perror()`	1	o l	

5.2 Memory Allocation

Only the first four of these are ISO C, the rest are GNU or X/Open extensions.

Function	Level	Options	Notes
`malloc()`	1		
`free()`	1		
`realloc()`	1		
`calloc()`	1		
`memalign()`	2		
`valloc()`	2		
`pvalloc()`	2		
`mallopt()`	2		
`mcheck()`	2		
`mprobe()`	2		
`mallinfo()`	1		

5.3 Character Handling

5.3.1 Classification of Characters

ISO C character classification functions, plus GNU extension `isblank()` and BSD/SVID extension `isascii()`.

Function	Level	Options	Notes
ISO C functions	1	l	
`isblank()`	2	l o	
`isascii()`	2	l o	

5.3.2 Case Conversion

ISO C case conversion functions plus BSD/SVID `toascii()` and SVID `_tolower()` and `_toupper()`.

Function	Level	Options	Notes
ISO C functions	1	l	
`toascii()`	2	l o	
`_tolower()`	2	l o	
`_toupper()`	2	l o	

5.3.3 Classification of Wide Characters

ISO C wide character classification functions, plus GNU extension `iswblank()`.

Function	Level	Options	Notes
ISO C functions	2	1	
`iswblank()`	2	1 o	

5.3.4 Wide Character Case Conversion

ISO C wide character case conversion functions.

Function	Level	Options	Notes
ISO C functions	2	1 o	

5.4 String and Array Utilities

5.4.1 String Length

ISO C `strlen()` plus GNU `strnlen()` extension.

Function	Level	Options	Notes
`strlen()`	1	1	
`strnlen()`	1	1 o	

5.4.2 Copying and Concatenation

ISO C mem* and str* copy functions plus some GNU extensions and some BSD compatibility functions such as `bcopy()`.

Function	Level	Options	Notes
ISO C functions	1	1	
`mempcpy()`	2	1 o	
`memccpy()`	2	1 o	
`strdup()`	2	1 o	
`strndup()`	2	1 o	
`bcopy()`	1	1	
`bzero()`	1	1	

5.4.3 String/Array Comparison

ISO C mem* and str* copy functions plus some GNU extensions and some BSD compatibility functions.

Function	Level	Options	Notes
ISO C functions	1	1	
`strcasecmp()`	2	1 o	
`strncasecmp()`	2	1 o	
`bcmp()`	2	1 o	

5.4.4 Collation Functions

ISO C collation functions `strcoll()` and `strxfrm()`.

Function	Level	Options	Notes
`strcoll()`	1	1	
`strxfrm()`	1	1	

5.4.5 Search Functions

ISO C mem* and str* search functions.

Function	Level	Options	Notes
ISO C functions	1	1	

5.4.6 Finding Tokens in a String

ISO C `strtok()` and BSD extension `strsep()`.

Function	Level	Options	Notes
`strtok()`	1	1	
`strsep()`	1	1	

5.4.7 Encode Binary Data

SVID/XPG `l64a()` and `a64l()` functions.

Function	Level	Options	Notes
`l64a()`	2	1	
`a64l()`	2	1	

5.4.8 Argz and Envz Vectors

Various functions for manipulating `argv[][]` style arrays and derived environment string arrays. These will be needed if support for `getenv()` and especially `putenv()` is present.

Function	Level	Options	Notes
`argz_create()`	2	1	
`argz_create_sep()`	2	1	
`argz_count()`	2	1	
`argz_extract()`	2	1	
`argz_stringify()`	2	1	
`argz_add()`	2	1	
`argz_add_sep()`	2	1	
`argz_append()`	2	1	
`argz_delete()`	2	1	
`argz_insert()`	2	1	
`argz_next()`	2	1	
`argz_replace()`	2	1	
`envz_entry()`	2	1	
`envz_get()`	2	1	
`envz_add()`	2	1	
`envz_merge()`	2	1	
`envz_strip()`	2	1	

5.5 Character Set Handling

Various ISO C and X/Open mechanisms for handling multibyte character conversions.

Function	Level	Options	Notes
ISO C functions	2	1	
X/Open functions	2	1	

5.6 Locales

ISO C and X/Open functions for setting and querying the locale.
At level 1 only support for the ”C” locale is required, and maybe the ”POSIX” one, if that ever differs from ”C”.

Function	Level	Options	Notes
ISO C functions	1	1	
X/Open functions	2	1	

5.7 Message Translation

X/Open “catgets” and GNU “gettext” facilities for translating messages into different languages.

Function	Level	Options	Notes
X/Open “catgets” functions	2	1	
GNU “gettext” functions	2	1	

5.8 Searching and Sorting

ISO C `bsearch()` and `qsort()` functions plus GNU extensions for hash and tree sort/search.

Function	Level	Options	Notes
`bsearch()`	1	1	
`qsort()`	1	1	
`hcreate()`	2	1	
`hdestroy()`	2	1	
`hsearch()`	2	1	
`hcreate_r()`	2	1	
`hdestroy_r()`	2	1	
`hsearch_r()`	2	1	
`tsearch()`	2	1	
`tfind()`	2	1	
`tdelete()`	2	1	
`tdestroy()`	2	1	
`twalk()`	2	1	

5.9 Pattern matching

Various GNU-only extensions for wildcards, globbing and regular expression matching.

Function	Level	Options	Notes
`fnmatch()`	2	1	
`glob()`	2	1	
`globfree()`	2	1	
`regcomp()`	2	1	
`regexec()`	2	1	
`regfree()`	2	1	
`regerror()`	2	1	
`wordexp()`	2	1	
`wordfree()`	2	1	

5.10 Stream IO

These are mostly the ISO C set plus some GNU, BSD and SYSV extensions. None of these are necessary if we don't have any devices, file systems or network stacks to use them with.

5.10.1 Opening and Closing

Function	Level	Options	Notes
`fopen()`	1		
`fopen64()`	2		
`freopen()`	1		
`freopen64()`	2		
`fclose()`	1		
`fcloseall()`	2		

5.10.2 Character and Line I/O

Function	Level	Options	Notes
`fputc()`	1		
`putc()`	1		
`putchar()`	1		
`fputs()`	1		
`puts()`	1		
`putw()`	2		
`fgetc()`	1		
`getc()`	1		
`getchar()`	1		
`getw()`	2		
`getline()`	2		
`getdelim()`	2		
`fgets()`	1		
`gets()`	1		
`ungetc()`	1		

5.10.3 Block IO

Function	Level	Options	Notes
`fread()`	1		
`fwrite()`	1		

5.10.4 Formatted Output

The main option on all of these would be to drop the floating point formats to produce an integer only implementation.

Function	Level	Options	Notes
`printf()`	1	m	
`fprintf()`	1	m	
`sprintf()`	1	m	
`snprintf()`	1	m	
`asprintf()`	2	m	
`vprintf()`	1	m	
`vfprintf()`	1	m	
`vsprintf()`	1	m	
`vsnprintf()`	1	m	
`vasprintf()`	2	m	
`obstack_printf()`	4	m	
`obstack_vprintf()`	4		
`parse_printf_format()`	4		

5.10.5 Formatted Input

As with the output functions, the floating point formats in these functions are optional.

Function	Level	Options	Notes
`scanf()`	1	m	
`fscanf()`	1	m	
`sscanf()`	1	m	
`vscanf()`	1	m	
`vfscanf()`	1	m	
`vsscanf()`	1	m	

5.10.6 EOF and Errors

Function	Level	Options	Notes
`clearerr()`	1		
`feof()`	1		
`ferror()`	1		

5.10.7 File Positioning

Function	Level	Options	Notes
`ftell()`	1		
`ftello()`	2		
`ftello64()`	2		
`fseek()`	1		
`fseeko()`	2		
`fseeko64()`	2		
`rewind()`	1		
`fgetpos()`	1		
`fgetpos64()`	2		
`fsetpos()`	1		
`fsetpos64()`	2		

5.10.8 Stream Buffering

Function	Level	Options	Notes
`fflush()`	1		
`setvbuf()`	1		
`setbuf()`	1		
`setbuffer()`	1		
`setlinebuf()`	1		

5.10.9 Other Kinds of Streams

Function	Level	Options	Notes
`fmemopen()`	4		
`open_memstream()`	4		
`open_obstack_stream()`	4		
`fopencookie()`	4		
`fmtmsg()`	4		
`addseverity()`	4		

5.11 Low Level I/O

Most of these functions are also covered by POSIX. Listed here are just those that are extensions to POSIX.

5.11.1 General IO

Function	Level	Options	Notes
`open64()`	2		
`truncate()`	4		
`truncate64()`	4		
`ftruncate64()`	4		
`pread()`	1		
`pread64()`	2		
`pwrite()`	1		
`pwrite64()`	2		
`lseek64()`	1		
`fclean()`	2		
`readv()`	2		
`writev()`	2		
`mremap()`	2		
`select()`	1	c m o	1
`poll()`	1	c m o	1
`sync()`	1		
`ioctl()`	1	c m o	

Note 1: `select()` and `poll()` were invented to enable single-thread UNIX processed to handle multiple events. Neither is at all necessary in a multi-threaded environment. If present they will exist only to support the porting of existing code, and may be severely limited in their applicability and performance. Use of these functions is deprecated.

5.11.2 Asynchronous IO

Function	Level	Options	Notes
`aio_read64()`	2		
`aio_write64()`	2		
`lio_listio64()`	2		
`aio_error64()`	2		
`aio_return64()`	2		
`aio_fsync64()`	2		
`aio_suspend64()`	2		
`aio_cancel64()`	2		
`aio_init()`	4		

5.12 File System Interface

Again, some of these are already defined by POSIX.

5.12.1 Directories

Function	Level	Options	Notes
`getwd()`	2	f[d] c	
`telldir()`	2	f[d]	
`seekdir()`	2	f[d]	
`scandir()`	2	f[d]	
`alphasort()`	2	f[d]	
`versionsort()`	2	f[d]	
`scandir64()`	2	f[d]	
`alphasort64()`	2	f[d]	
`versionsort64()`	2	f[d]	
`ftw()`	2	f[d]	
`ftw64()`	2	f[d]	
`nftw()`	2	f[d]	
`nftw64()`	2	f[d]	

5.12.2 Links and Special Files

Function	Level	Options	Notes
`symlink()`	2	f[ml]	
`readlink()`	2	f[l]	
`remove()`	1	f[m]	
`mknod()`	4	f[m]	

5.12.3 File Attributes

Function	Level	Options	Notes
`stat64()`	2	f	
`fstat64()`	2	f	
`lstat()`	2	f[l]	
`lstat64()`	2	f[l]	
`fchown()`	2	f[mp]	
`getumask()`	2	f[p]	
`utimes()`	2	f	

5.12.4 Temporary Files

Function	Level	Options	Notes
`tmpfile()`	1	f[m]	
`tmpfile64()`	2	f[m]	
`tmpnam()`	1	f[m]	
`tmpnam_r()`	1	f[m]	
`tempnam()`	2	f[m]	
`mktemp()`	2	f[m]	
`mkstemp()`	2	f[m]	

5.13 Pipes and FIFOs

Function	Level	Options	Notes
`popen()`	4		
`pclose()`	4		

5.14 Sockets

The database access functions here should probably have configurations that allow them to work from memory based tables or ”string files” rather than real files. These functions are also all marked as optional at level 1 since their functionality is often not required in an embedded system.

5.14.1 Basic API

Function	Level	Options	Notes
`socket()`	1	n	
`bind()`	1	n	
`getsockname()`	1	n	
`shutdown()`	1	n	
`socketpair()`	1	n	
`connect()`	1	n	
`listen()`	1	n	
`accept()`	1	n	
`getpeername()`	1	n	
`send()`	1	n	
`recv()`	1	n	
`sendto()`	1	n	
`recvfrom()`	1	n	
`getsockopt()`	1	n	
`setsockopt()`	1	n	

5.14.2 Internet Addresses

Function	Level	Options	Notes
`inet_aton()`	1	n	
`inet_addr()`	1	n	
`inet_network()`	1	n	
`inet_ntoa()`	1	n	
`inet_makeaddr()`	1	n	
`inet_lnaof()`	1	n	
`inet_netof()`	1	n	
`inet_pton()`	1	n	
`inet_ntop()`	1	n	

5.14.3 Hosts Database

Function	Level	Options	Notes
`gethostbyname()`	1	n o	
`gethostbyname2()`	1	n o	
`gethostbyaddr()`	1	n o	
`sethostent()`	1	n o	
`gethostent()`	1	n o	
`endhostent()`	1	n o	

5.14.4 Protocols Database

Function	Level	Options	Notes
`getprotobyname()`	1	n o	
`getprotobynumber()`	1	n o	
`setprotoent()`	1	n o	
`getprotoent()`	1	n o	
`endprotoent()`	1	n o	

5.14.5 Services Database

Function	Level	Options	Notes
`getservbyname()`	1	n o	
`getservbyport()`	1	n o	
`setservent()`	1	n o	
`getservent()`	1	n o	
`endservent()`	1	n o	

5.14.6 Networks Database

Function	Level	Options	Notes
`getnetbyname()`	1	n o	
`getnetbyaddr()`	1	n o	
`setnetent()`	1	n o	
`getnetent()`	1	n o	
`endnetent()`	1	n o	

5.14.7 Byte Order Conversion

Function	Level	Options	Notes
`htons()`	1	n	
`ntohs()`	1	n	
`htonl()`	1	n	
`ntohl()`	1	n	

5.14.8 Interface Naming

Function	Level	Options	Notes
`if_nametoindex()`	4		
`if_indextoname()`	4		
`if_nameindex()`	4		
`if_freenameindex()`	4		

5.15 Low Level Terminal Interface

Function	Level	Options	Notes
`cfmakeraw()`	4		
pseudo-terminal functions	4		

5.16 Mathematics

The usual collection of trig, exponent, hyperbolic and random number functions, too many to list here, look at the ISO C standard for a full list.

Function	Level	Options	Notes
ISO C89 functions	1	l	
ISO C99 functions	1	l o	
GNU extensions	2	l o	

5.17 Arithmetic

ISO C floating point arithmetic and FPU control functions plus some extensions.

Function	Level	Options	Notes
ISO C89 functions	1	l	
ISO C99 functions	1	l o	
GNU extensions	2		

5.18 Date and Time

Full time zone support can be very large in code and data sizes. At levels 1 and 2 a considerably reduced implementation may be provided.

Function	Level	Options	Notes
`clock()`	1		
`difftime()`	1		
`gettimeofday()`	1		
`settimeofday()`	1		
`adjtime()`	4		
`localtime()`	1		
`gmtime()`	1		
`mktime()`	1		
`asctime()`	1		
`ctime()`	1		
`strftime()`	1		
`strptime()`	1		
`getdate()`	1		
`getdate_r()`	1		
`ntp_gettime()`	1	n o	
`ntp_adjtime()`	1	n o	
`setitimer()`	1	m o	
`getitimer()`	1	m o	

5.19 Resource Usage and Limits

Function	Level	Options	Notes
`getrusage()`	1	m o	
`getrlimit()`	1	m o	
`getrlimit64()`	2	m	
`setrlimit()`	2	m	
`setrlimit64()`	2	m	

5.20 Non-Local Exits

Function	Level	Options	Notes
`setjmp()`	1		
`longjmp()`	1		

5.21 Signal Handling

Many of these are obsolete or alternative interfaces to the basic POSIX signal functionality. Use of these should be replaced where necessary by the POSIX functions.
`strsignal()` and `psignal()` share the same problems as `strerror()` and `perror()` in needing RAM to store the strings table.

Function	Level	Options	Notes
`strsignal()`	1	o m	
`psignal()`	1	o m	
`signal()`	1	c	1
`raise()`	1	c	2
`sysv_signal()`	4		
`ssignal()`	4		
`gsignal()`	4		
`killpg()`	4		
`sigaltstack()`	4		
`sigstack()`	4		
`sigvec()`	4		
`siginterrupt()`	4		
`sigblock()`	4		
`sigsetmask()`	4		
`sigpause()`	4		

Note 1: This is merely a wrapper for `sigaction()`.
Note 2: This is merely a wrapper for `kill()`.

5.22 Program Startup and Termination

5.22.1 Program Arguments

Function	Level	Options	Notes
`getopt()`	2	1	
`getopt_long()`	2	1	
`argp_parse()`	2	1	
`argp_help()`	2	1	
`getsubopt()`	2	1	

5.22.2 Environment Variables

The only real use for changing the environment is to prepare it for passing on to a sub-process. Hence these functions are only necessary at level 3.

Function	Level	Options	Notes
`putenv()`	3		
`setenv()`	3		

5.22.3 Program Termination

Function	Level	Options	Notes
`exit()`	1		
`atexit()`	1		
`on_exit()`	2		
`abort()`	1		
`_Exit()`	1		

5.23 Processes

Function	Level	Options	Notes
`system()`	4		
`vfork()`	4		
`wait3()`	4		
`wait4()`	4		
`getpriority()`	4		
`setpriority()`	4		
`nice()`	4		

5.24 Users and Groups

Function	Level	Options	Notes
`setreuid()`	4		
`setregid()`	4		
`setgroups()`	4		
`initgroups()`	4		
GNU, BSD and X/Open extensions	4		

5.25 System Information

Function	Level	Options	Notes
`gethostname()`	1	n	
`sethostname()`	1	n	
`gethostid()`	1	n	
`sethostid()`	1	n	
fstab access functions	2	f	
mtab access functions	2	f	

5.26 System Configuration

Function	Level	Options	Notes
`confstr()`	2	o	

5.27 Cryptographic Functions

Some embedded applications will need cryptographic security. However the DES based encryption provided by these functions is probably inadequate and has various legal problems. Those applications that need it will probably pick up a stronger, legal, encryption package elsewhere. Hence these functions are entirely optional.

Function	Level	Options	Notes
`getpass()`	4		
`crypt()`	1	l o	
`crypt_r()`	1	l o	
`setkey()`	1	l o	
`encrypt()`	1	l o	
`setkey_r()`	1	l o	
`encrypt_r()`	1	l o	
`ecb_crypt()`	1	l o	
`cbc_crypt()`	1	l o	
`des_setparity()`	1	l o	

5.28 POSIX Threads

The following are LinuxThreads extensions.

Function	Level	Options	Notes
`pthread_cleanup_push_defer_np()`	2		
`pthread_cleanup_pop_restore_np()`	2		
`pthread_mutexattr_setkind_np()`	2		
`pthread_mutexattr_getkind_np()`	2		
`pthread_kill_other_threads_np()`	2		