CONNOTECH Experts-conseils inc.
PPCMB/850 Product Family Documentation

# The FlashCnL API,

## A Flash Memory Configuration and Log Application Programming Interface

(Embedded Software Document)

Document Number C001270

2003/06/17



Document Revision History

| C-Number | Date | Explanation |
|---|---|---|
| C001268 | 2002/10/31 | First Release, draft form |
| C001270 | 2003/06/17 | Current version |

Table of contents

# 1. Why Use the FlashCnL API Library

In typical embedded systems where the permanent memory subsystem is very limited (due to power, size, weight, reliability and vibration resistance considerations), it is nonetheless required to store some system configuration information, and to a limited extent, service history information.

The FlashCnL API library is a simpler alternative to a full-blown "flash file system" that would emulate a hard disk or floppy disk file system. When the permanent storage application requirements are reasonably simple, the following benefits are to be expected from the FlashCnl API library:

- Self-complete solution for permanent storage of system configuration data and service history information (log).
- Application control of when a flash sector erase is allowed.
- Manageable software source code, leading to ease of integration with system application software.
- The FlashCnL API is a natural companion to boot-from-flash schemes where the application software image resides in dedicated flash sectors (instead of an application executable file in a flash file system under the control of a resident operating system).
- Emphasis on reliability.

# 2. Technical Overview

The FlashCnL API allows application friendly storage of configuration data and recording of system log information in flash memory. It is intended for Embedded systems having external flash memory as their permanent storage area. The FlashCnL API typically controls only a portion of the overall system flash memory, e.g. where the bulk of flash memory is used for software image storage. The FlashCnL initialization function expects a pointer to the start of the relevant flash memory portion.

Conceptually, the FlashCnL API stores configuration data in a reserved section of the flash memory and allows retrieval by configuration structure *name* in a manner similar to the getenv standard C function for environment variables. The length of a FlashCnL configuration structure is explicit (this departs from the gentenv behavior which uses null-terminated strings). Although arbitrary creation and deletion of configuration structure *names* is made possible by the API, it is

generally more efficient to organize configuration data in a set of permanent configuration structure names, and to update the individual structure *contents* during system operation.

Still at the conceptual level, the FlashCnL API provides a system log recording function where system log is a sequence of chunks that can be read back from the oldest entry to the most recent one. The FlashCnL system log function attempts to provide a FIFO processing model in the form of a read-back capability with an implicit *chunk acknowledgment* operating mode. The delineation of chunks in the FlashCnL system log defines the acknowledgment granularity; itis unrelated to end-of-line indications. No attempt is made to provide the application with assurance that the system log is empty at any given time in system operation time. In other words assuming a multi-threading environment, it is possible for a thread to record a system log chunk while another thread attempts to empty the FIFO.

The FlashCnL software manages two dedicated sections in the flash memory to provide the configuration storage and system log services to an application. It hides the limitations of the flash memory technology to a large extent, but an overview of the flash memory usage strategy is helpful to design an application using the FlashCnL API. Basically, the FlashCnL software starts with an erased flash memory section and writes sequentially to it, starting with a “first entry” followed by entries of three different types: log entries, list-of-names entries, and configuration entries. The definition of entries include “displacement ” fields (functionally equivalent to pointers) that are expected to point forward in the flash memory when further entries will be written by the FlashCnL software (as directed by the application). These displacement fields are first left in the virgin state (having an all-one binary value). This creates pointer chains for the FlashCnL software to find its way to the up-to-date configuration structures and the sequence of system log chunks.

Each FlashCnL dedicated section may consist of one or more flash sectors. As previously said, the FlashCnL software manages two such sections. Only one of them is active at a time. When no more space is left in the active FlashCnL section, it becomes necessary to erase the alternate section and to copy the up-to-date configuration structures to this virgin section. This is called a FlashCnL rollover operation. The current active memory section is the one having the highest value in its version number field (present in the memory section first entry).

The FlashCnL memory organization as a pair of active / alternate sections allows an added application benefit in the form of a backup function. If a corruption is detected in the active memory section, it can be rolled back to the configuration state contained in the alternate section. To prevent such corruption detection and (partial) recovery from remaining unnoticed, the rollback operation does not increase the version number field. It increases a rollback count field (also present in the memory section first entry) from its default value of zero.

The rollover and the rollback operations involve the erasing of a flash memory section. With the

current prevailing flash memory technology, this is a time-consuming operation during which the normal flash memory operation is restricted. The FlashCnL software expects the application to determine the proper times for a possible rollover operation (which might turn into a rollback operation if corruption is detected). Depending on system operational requirements, this time can be the system startup time, or periods of inactivity. The application can let the FlashCnL software to determine if the active memory section is filled above a "high water mark" threshold, in which case the rollover operation is performed.

Some log entries are lost if they are not retrieved in good time. Entries not retrieved after two rollover operations are lost. If a rollback occurs due to a detected corruption, the log entries prior to the rollback operations are completely lost.

Note: This last point is a design decision. First, it was observed that the corruption at hand is flash memory contents corruption, which is rare according to experience. Second, it was deemed preferable to hide from the user any doubtful log information (if it was possible for the log entries to have an undisclosed gap due to an intervening rollback operation, any sequence of log entries would be questionable for the user).

The FashCnL software can be used as the sole permanent storage mechanism for operational data in embedded systems. In order to improve reliability, the set of internal memory variables maintained by the FlashCnL software is integrity-protected by a cyclic redundancy check mechanism. The software also provides extensive data integrity validations for the data stored in the flash sections.

The whole FlashCnL detailed design and implementation is oriented towards alleviating the inherent limitation of the flash memory technology: when writing to a flash memory location, no bit value can be changed from a "0" to a "1" (a bit can either be left unchanged or changed from a "1" to a "0"). In order to write a "1" where a "0" has been previously written, a whole flash "sector" has to be "erased," which is a time-consuming and disturbing operation (a typical flash sector size is from 4KB to 128KB). The FlashCnL control information data structures and processing logic are such that no erase operation is needed until the FlashCnL section is filled.

# 3. Implementation Documentation

This section should be read as a complement to the FlashCnL API library source code. The definitive reference for the implementation details is in the source code.

## 3.1 A Few Principles used in the Software Implementation

Basically, a FlashCnL section is a growing sequence of FlashCnL entries. There are four type of entries:

- The FlashCnL section first entry

  This entry is the root for every other entry in the FlashCnL section. It has a special format and includes control information about the section and the other one. In addition, the FlashCnL first entry contains holes (reserved fields) to leave room for control information required by other flash usage conventions (e.g. the first instruction fetched at a specific address by the microprocessor upon reset if a FlashCnL section collides with the "boot sector" in a given microprocessor design).

- Log entries

  The sequence of log entries in the two FlashCnL section contains the log "chunks" inserted by the application software. One log entry may contain one or more log chunks. Each chunk is prefixed by a chunk length encoding, and an acknowledgment bit.

- List-of-name entries

  This entry contains the list of configuration structure names. The individual name encoding contains a name length encoding, the name itself, and a displacement field for the configuration data entry. The name length encoding reserves a bit for a deleted name. Only the last list-of-name entry in the FlashCnL section is relevant.

- Configuration data entries

  A configuration data entry hods the binary configuration data for a named configuration structure. For a given name, only the last configuration data entry is relevant.

Displacement instead of pointers

Instead of pointer and address computations, the FlashCnl software uses displacement values. A displacement field within a FlashCnL entry is relative to the address of the FlashCnLentry in which it occurs. In some cases, a displacement can be negative, but the value -1 is reserved as a virgin displacement. The use of displacement fields makes the FlashCnL memory section position-independent. A displacement field is represented as a 32 bits signed binary number.

Common fields in every FlashCnL entries

The first two fields are common to all FlashCnL entries (**`struct FlashCnL_entry_str`**):

- a **`next_disp`** field contains the displacement to the next entry in the sequence of entries as they occur in the FlashCnL section (a virgin next_disp value occurs only for the current last entry in the section), and
- a **`next_kind_disp`** field containing the displacement to the next entry in a chain of related entries (having the same kind).

After these two fields, the data structure in a FlashCnL entry is dictated by the kind of entry. In a sense, the kind of entry is inherited from the context, namely the displacement (usually a **`next_kind_disp`** field value) from which the entry was reached.

Size indication of a FlashCnL entry

The size of a FlashCnL entry is either set by the **`next_disp`** field or implicitly bounded by the virgin portion of the FlashCnL section in the special case of the last entry when its **`next_disp`** field is virgin (-1). When this special case occurs, the encoding of for FlashCnL entry data is such that the end of data can be identified upon decoding the first virgin byte after the valid entry data.

Some semantic rules for FlashCnL chaining

Semantic rules are such that an FlashCnL entry can be forgotten, or more or less lost, when no currently valid displacement field points to it.

A FlashCnL entry data can be empty. In at least one case, a virgin displacement field is semantically equivalent to a displacement towards an entry with an empty data field.

Implementation Details:

The size of a FlashCnL section is validated to be in the range from 4KB to 64MB (more or less arbitrary numbers).

Oriented towards 32-bits CPUs without word alignment restrictions.

The implementation is biased towards minimal memory usage (no use of dynamic memory).

## 3.2 Data Representation in the Flash Memory

A virgin **`next_kind_disp`** occurs for the current up-to-date entry for configuration data entries of each configuration data name.

A virgin **`next_kind_disp`** occurs in the last log entry in the log sequence.

### 3.2.1 FlashCnL Section First Entry

- FlashCnL first entry root displacements
  - The **`r.first_next_disp`** displacement field, that starts the entry chained in ascending order of addresses according to their **`next_disp`** fields
  - The **`r.next_log_kind_disp`** displacement field, that starts the chained log entries.
  - The **`r.list_of_names_disp`** displacement field, that refers to the first list-of-name entry.
- FlashCnL section context data
  - A **`d.c.magic_number`** field.
  - The **`d.c.self_address_coding`** field is a genuine pointer to the FlashCnL section itself (not a displacement). It is the only indication of the intended FlashCnL section address and should be ignored if the execution environment memory management subsystem might relocate the FlashCnL section in any way.
  - An overall length indication for the FlashCnL section, field name **`d.c.FlashCnL_section_size`**.
  - The version number and rollback count, respectively **`d.c.version`** and **`d.c.rollback_count`**, that keep track of roll-over and roll-back operations.
  - Displacement to the alternate FlashCnL section, field name **`d.c.alternate_section_disp`**, that allows mutual cross-referencing between the two FlashCnL sections.
  - A 32-bit CRC field is used for validation of the other context data.

The first entry structure definition reserves space for other uses of specific flash memory addresses. Somehow arbitrarily, the first entry of a FlashCnL section occupies 0x180 bytes but

actually uses space in the higher displacements below the 0x80 displacement (currently this is bytes from displacement 0x58 to 0x7F). This space utilization and reservation may be adapted to a specific processor and memory system. The current FlashCnL accommodates the Motorola MPC8xx reset vector at displacement 0x100 in the case a FlashCnL section overlaps the exception vectors for this processor family.

### 3.2.2 Log Entries

A log entry consists of one or more log chunk. A virgin **`next_kind_disp`** occurs for the last log entry.

A log chunk has a length encoding followed by the chunk contents itself. There is no need for a chunk contents to be a null-terminated string. The length encoding has a specific bit assigned to the log acknowledgment function. A“more ” bit is used so that the chunk length encding is usually 1 or 2 bytes. A chunk length of zero is not supported. A chunk length is limited to 2^27 (32 bits less 4 more bits less one acknowledgment bit).

For the last log entry, the **`next_disp`** field may be virgin. In this case, the appending of a new log chunk occurs at the end of this last log entry. Otherwise, a new log entry is chained to the current last one and the new chunk is deposited in it.

### 3.2.3 List-of-name Entries

A virgin **`next_kind_disp`** occurs for the last and current up-to-date entry for list-of-names. Only this last entry is to be considered by the FlashCnl software.

The list of names is made of configuration names (encoded exactly like a FlashCnL log chunk) immediately followed by a displacement field to the first configuration data contents (if any) for this name. Configuration names and displacements are packed. A configuration name can not have a zero length.

The acknowledgment bit in the configuration name length encoding is used to mark a deleted configuration name.

For the last list-of-names entry, the **`next_disp`** field may be virgin. In this case, a name creation occurs at the end of this list-of-names. Otherwise, a name creation forces a copy of the current up-to-date list-of-names (configuration names marked for deletion are not copied).

### 3.2.4 Configuration Data Entries

A virgin **`next_kind_disp`** occurs for the last and current up-to-date entry holding the

configuration data for each specific name. Only this last entry is to be considered by the FlashCnl software.

For a configuration data entry, the **`next_disp`** field is a data size indication and hence may be not be virgin.

In updating the configuration data contents for a given name, a new configuration data entry is usually required. The exception occurs when the new configuration data is the same size as the existing one and writing the new data involves only turning "1" bits into "0" (or leaving bits to their current setting). In this special case, the FlashCnl software overwrites the current configuration data with the new one and no additional flash memory space is used. An application can exploit this space-efficient behavior.

## 3.3 Implementation Interfaces

### 3.3.1 Low-level Flash Services

The interface to the low-level flash memory interface occurs in the source code in conditionally compiled code sequences with the following format:

```
#if TEST
/*
   source variant for the test program (software emulation)
*/
#elif (ABCD_TARGET!=ABCD_TARGET_NO)
/*
   source variant for the ABCD Proto-Kernel execution context
*/
#else
/*
   source variant for the other execution context(s)
*/
#endif
```

The "source variant for the other execution context(s)" is left empty in the current FlashCnL software distribution. This is the place where most of the source code personalization should be done.

Need to validate pointers

In the source file flashcnl_init_oper.cpp, it is recommended to add a test for the validity of a

memory address range with respect to flash memory access. This test is then effected whenever the function **`FlashCnL_init_oper`** is called, typically including at system reset time.

Erasure of a FlashCnL section

In the source files flashcnl_init.cpp and flashcnl_roll_sect.cpp (respectively in functions **`FlashCnL_init_syst`** and **`FlashCnL_roll_section`**), it is required to add the source code sequence that erases a FlashCnl section (which may consist of one or more flash low-level sectors).

Flash write function

The FlashCnL source code interfaces the low-level flash programming algorithm through a function declared as follows:

```
extern void flash_program_bytes  ( unsigned char *pt
                                 , const void *src
                                 , size_t len);
```

This declaration should appear in an include file to be **`#include`**'d in most .cpp source files, nearby where the include file abcd_flash_pic_fncs.h is **`#include`**'d for the source variant for the ABCD Proto-Kernel execution context.

### 3.3.2 Mutual-Exclusion Semaphores (mutexes)

The FlashCnL source code uses preprocessor macros for reserving and relinquishing mutual exclusion semaphores (mutexes) as required for serializing access to the FlashCnl global resources, assuming that an underlying kernel or operating system provide such a serialization mechanism.

#### 3.3.2.1 Serialization Principles

The assumed mutual exclusion semaphore mode is fairly simple: the first task or thread that reserved a mutex takes precedence over any other task or thread that attempts to reserve it, until the first one relinquishes the mutex.

Note: A refined serialization model is conceivable. It would allow any number of tasks to reserve *referential integrity* concurrently, but a single one to reserve an update permission for the resource subject to serialization. This refined model was deemed an overkill for the typical application areas for the FlashCnL API library.

There are three such semaphores. Their names, FLASHCNL_NAMES_MUTEX, FLASHCNL_LOG_MUTEX, and FLASHCNL_UPDATE_MUTEX, are arguments to the preprocessor macros used through the FlashCnL source code.

- The FLASHCNL_UPDATE_MUTEX is used for the access permission to the static global variables (control information), either for referential integrity when the contents of the FlashCnL section is queried by an API call, or for update permission for elementary update operations to either or both FlashCnL sections.

- The FLASHCNL_LOG_MUTEX is used while the sequence of log chunks is processed with acknowledgment of log entries. A reservation of the FLASHCNL_LOG_MUTEX occurs for longer periods than reservations of FLASHCNL_UPDATE_MUTEX. When the application software is given the opportunity to process a log chunk, the FLASHCNL_LOG_MUTEX is reserved, while the FLASHCNL_UPDATE_MUTEX is not.

- The FLASHCNL_NAMES_MUTEX is used for referential integrity to the list of configuration structure names, notably when the list of names is processed. A reservation of the FLASHCNL_NAMES_MUTEX occurs for longer periods than reservations of FLASHCNL_UPDATE_MUTEX. When the application software is given the opportunity to process a configuration name, the FLASHCNL_NAMES_MUTEX is reserved, while the FLASHCNL_UPDATE_MUTEX is not.

Note: Perhaps there is yet another serialization requirement, but it is assumed to be met by the low level flash access interface functions: chances are that the flash access has to be serialized at some point. Usually, the flash programming algorithm is a sequences of programming operations to be performed without disruption (atomically) by the CPU with respect to the flash integrated circuits. This issue is not handled by the FlashCnL software.

For the FlashCnL section rollover and rollback operations, the three mutexes are reserved in the following order: FLASHCNL_NAMES_MUTEX, FLASHCNL_LOG_MUTEX, and FLASHCNL_UPDATE_MUTEX. This mutex reservation sequence would be relevant to the application software design only if these mutexes are reserved by application software outside of the FlashCnL API library. Otherwise, the mutex usage by the FlashCnL API library source code is transparent to the application software.

#### 3.3.2.2 Mutex Implementation in the Source Code

The detailed specifications of any mutual exclusion semaphore mechanism is operating system dependent. Each such macro expects a single argument that is one of the three mutex names. In the FlashCnL source code, preprocessor macros are called in appropriate places:

Reservation and relinquish operations

- The FLASHCNL_MTX_RESERVE(MTX) preprocessor macro is called where a mutex reservation should occur.

- The FLASHCNL_MTX_RELINQUISH(MTX) preprocessor macro is called where a mutex relinquish operation should occur.

Declarations and definitions

- The FLASHCNL_MTX_AUTODECL(MTX) preprocessor macro is called at beginning of the C statement block (where an automatic variable definition may appear) for the innermost block where a reservation/relinquish pair occur.

  Note: The mere presence of the FLASHCNL_MTX_AUTODECL(MTX) in the FlassCnL design should not obscure the fundamentals of mutexes. Usually, a mutex is a global operating system entity that spawns task or thread boundaries.

- The FLASHCNL_MTX_MEMBDECL(MTX) and the FLASHCNL_MTX_CONSTR(MTX) preprocessor macros are used where the reservation/relinquish pair occurs in C++ class member functions and a member variable replaces the automatic variable definition allowed with the FLASHCNL_MTX_AUTODECL(MTX) preprocessor macro. The FLASHCNL_MTX_MEMBDECL(MTX) macro is used where the member variable declaration would occur in the class definition, and the FLASHCNL_MTX_CONSTR(MTX) macro is used where a class member initializor is allowed in the class constructor definition.

### 3.3.3 Software Building Issues

The essential FlashCnL API library source is made of
- a collection of .cpp source files (file name format flashcnl_*.cpp), and
- a single include file, flashcnl.h.

Other files needed for the correct software building are
- the source file flashcnl_test.cpp for the test program,
- a single general-purpose include file (abcd_incl.h) that may be replaced, and
- a couple of files for the CRC (Cyclic Redundancy Check) algorithm implementation if it is object-oriented (C++ programming).

Building the test program should represent a trivial assignment: compile every .cpp files in the distribution and link them. The resulting executable is a command-line utility that takes no argument and spits a long sequence of messages. The test program does work. A makefile is provided for the GNU make utility and the GCC compiler. In a leading proprietary execution

environment, the FlashCnL test program has been successfully tried using the WATCOM C/C++ Version 11.0c.

Customization of the source code for other execution environments may comprise the adjustment of some preprocessor symbols in the flashcnl.h include file, namely FLASHCNL_OPTIONS_INIT, FLASHCNL_OPTIONS_C_APPL, FLASHCNL_CTL_VARS_CRC_CHECK, and FLASHCNL_CRC32_CLASS.

### 3.3.4 Interface to Flash Memory Life-cycle Management

The FlashCnL API library is typically used as the single permanent storage for system configuration data, including the serial number of an embedded system and other data that should never be overwritten. For this reason, the very first initialization of the flash sectors assigned to the FlashCnL should be a reserved operation done only at the microprocessor subsystem manufacturing time. From then on, the operational software and the field service software utilities should always restrict themselves to updating the configuration and triggering FlashCnL rollover operations in order to preserve the existing critical system configuration elements.

The preprocessor macro FLASHCNL_OPTIONS_INIT is used to conditionally include the FlashCnL function **`FlashCnL_init_syst`** that performs the very first initialization of the flash sections assigned to the FlashCnL.

The present document does not address the design decisions that lead to the reservation of specific flash sectors for the FlashCnL. The function **`FlashCnL_init_syst`** implements such design decisions. Once the FlashCnl sections in the flash are so initialized, the system reset sequence uses a pointer to the start of either FlashCnL section to find its way to the FlashCnL section contents (see function **`FlashCnL_init_oper`**).

### 3.3.5 Test Program

A test program is provided. It exercises the library with random function calls. The test program uses simple software emulation instead of an actual flash memory.

# 4. Application Perspective on FlashCnl functions

Here is a synopsis of the application perspective on FlashCnl functions.

## 4.1 Log Capability

- A log chunk may be appended to the current log stream. When a log chunk is

handled, its size is indicated explicitly.

```
enum FlashCnL_result_codes FlashCnL_put_log
          ( const void *buf
          , size_t len);
```

- A log chunk is merely defined as the data passed to individual calls to the log appending function.
- Log chunks may be retrieved in the order in which they were appended.
- When retrieving log chunks, they may be acknowledged such that they are not going to be retrieved again.
- This acknowledgment of log chunks must be sequential.

```
class FlashCnL_log_cursor_cl
{
     /* internal details omitted */
public:
     FlashCnL_log_cursor_cl(void);
     FlashCnL_log_cursor_cl(int ack);
     unsigned char *next(size_t *len);
};

void FlashCnL_dump_log
          ( int acknowledge
          , int(*process)(const unsigned char *,size_t));
```

Note: The **`FlashCnL_dump_log`** function is suitable to C-style source code. Acknowledgment occurs only sequentially when the **`acknowledge`** parameter is non-zero and the **`process`** calls returns non-zero.

## 4.2 Configuration Data Management Capability

- A name may be created if it doesn't already exist. A name can not be null. After initial name creation, there is empty configuration data associated with it.

```
enum FlashCnL_result_codes FlashCnL_create_name
          ( const char *name
          , size_t name_len);
```

- The configuration data associated with a name may be set. This includes the capability to empty the configuration data.

```
enum FlashCnL_result_codes FlashCnL_set_name_data
          ( const char *name
          , size_t name_len
          , void *data
          , size_t data_len);
```

- The configuration data associated with a name may be retrieved.

```
enum FlashCnL_result_codes FlashCnL_get_name_data
          ( const char *name
          , size_t name_len
          , void **data
          , size_t *data_len);
```

- A name may be deleted.

```
enum FlashCnL_result_codes FlashCnL_delete_name
          ( const char *name
          , size_t name_len);
```

- The current list of may be retrieved sequentially. The order in which they are retrieved is not specified.

```
class FlashCnL_names_cursor_cl
{
     /* internal details omitted */
public:
     FlashCnL_names_cursor_cl(void);
     void init(void);
     char *next(size_t *len);
};

void FlashCnL_dump_names
          ( void(*process)(const char *,size_t));
```

## 4.3 Initialization and Status

- The FlashCnL system may be initialized, either
  - without allowing any automatic rollback or rollover operation,
  - allowing automatic rollback or rollover operation upon detection of a corrupted FlashCnL section,
  - as above plus allowing automatic rollover operation upon detection of a

FlashCnL section fill ratio above some highwater mark.

```
int FlashCnL_init_oper
          ( unsigned long *starting_pt
          , int rollover_permission);
```

- The FlashCnL system may be queried for its current status, including:
  - any corruption indication for the two FlashCnL sections,
  - the version number and rollback count for the two FlashCnL sections,
  - the current fill ratio for the active FlashCnL section,
  - whether a) the log chunk append function is currently efficient, b) the name creation function is currently efficient, or c) none of the above applies.

```
void FlashCnL_get_status
          (struct FlashCnL_status_str *stat);
```

# 5. FlashCnL Licensing

You'll find below the license terms applicable to the FlashCnL API library.

If it is administratively proper for your organization to pay for the software it acquires, you may send a payment of $149.00 in US currency (quote CONNOTECH part number 970-0001-01, “FlashCnL API Library distribution fee”). You will then receive a formal acknowledgment. Your will also receive notices of important bug fixes and major updates to the FlashCnL API library.

If you otherwise feel that the FlashCnL is useful and worth further development and active support, you might consider making a donation to the developer, CONNOTECH Experts-conseils inc.

In any case, if your organization's legal department feels uncomfortable with the GNU licensing scheme, feel free to contact us for special licensing arrangements.

## 5.1 FlashCnL License Notice

```
------------------------------------------------------------------------------
                   GNU LGPL (Lesser General Public License)
                 with a Waiver on the Opportunity to Relink
------------------------------------------------------------------------------

The FlashCnL API, a flash memory configuration and log application programming
interface library

Copyright (C) 2003 CONNOTECH Experts-conseils inc.
```

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License version 2.1 as published by the Free Software Foundation, except for its preamble and clause 6 that are modified as stated herein:

A) DELETE from the preamble fifth paragraph the sentence that reads
``If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it.''

B) DELETE the preamble before last paragraph that reads
``Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.''

C) At the end of the first paragraph of clause 6, REPLACE the phrase
``, provided that the terms permit modification ... ... debugging such modifications.''
by the following phrase
``, provided that the provisions of the following paragraph are followed.''

D) At the end of the text in the bullet a) in clause 6, DELETE the phrase
``; and, if the work is an executable linked with ... ... use the modified definitions.)''

E) DELETE the complete paragraph after the bullet e) in clause 6, this paragraph reading
``For an executable, the required form ... ... itself accompanies the executable.''

These modifications do not however invalidate any other reasons why derived work might be covered by the exact GNU Lesser General Public License.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License version 2.1 for more details.

You should have received a copy of the GNU Lesser General Public License version 2.1 along with this library; if not, you may write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

HOW TO CONTACT THE COPYRIGHT HOLDER:
CONNOTECH Experts-conseils inc.
home page: http://www.connotech.com
e-mail to: info@connotech.com
mailing address: 9130 Place de Montgolfier, Montreal, Quebec, Canada, H2M 2A1.

---

## 5.2 GNU Lesser General Public License (GNU LGPL) Version 2.1

GNU LESSER GENERAL PUBLIC LICENSE

Version 2.1, February 1999

Copyright (C) 1991, 1999 Free Software Foundation, Inc. 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

```
[This is the first released version of the Lesser GPL. It
also counts as the successor of the GNU Library Public
License, version 2, hence the version number 2.1.]
```

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages--typically libraries--of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. ~~If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it.~~ And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

~~Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.~~

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined

with the library in order to run.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".

A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.

The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)

"Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.

1. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms

of Section 1 above, provided that you also meet all of these conditions:

- a) The modified work must itself be a software library.

- b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.

- c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.

- d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.

  (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)

  These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

  Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.

  In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License

has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.

Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.

This option is useful when you wish to copy part of the code of the Library into a program that is not a library.

4. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.

If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.

5. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

6. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice~~, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications~~*, provided that the provisions of the following paragraph are followed*.

You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

- a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above)~~; and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)~~

- b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.

- c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.

- d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.

- e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

~~For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.~~

It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

7. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

- a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.

- b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

8. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

9. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.

10. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing

compliance by third parties with this License.

11. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

12. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

13. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may

choose any version ever published by the Free Software Foundation.

14. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

15. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS