



LogicLoader User's Manual

User's Manual for Logic's SOM Development Kits

Michael Erickson, Bruce Rovner

Logic Product Development

Published: January, 2003

This file contains source code, ideas, techniques, and information (the Information) which are Proprietary and Confidential Information of Logic Product Development, Inc. This information may not be used by or disclosed to any third party except under written license, and shall be subject to the limitations prescribed under license.

No warranties of any nature are extended by this document. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed license or agreement to purchase or lease equipments. The only warranties made by Logic Product Development, if any, with respect to the products described in this document are set forth in such license or agreement. Logic Product Development cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special or consequential damages.

Logic Product Development may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering the subject matter in this document. Except as expressly provided in any written agreement from Logic Product Development, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

© Copyright 2003-2005, Logic Product Development, Inc. All Rights Reserved.

REVISION HISTORY

REV	EDITOR	REVISION DESCRIPTION	APPROVAL	DATE
A	Bill O'Donnell	Release	BOD	1/21/03
B	Bruce Rovner	Release	BR	5/05/03
C	Bruce Rovner	Release	BR	7/24/03

D	Hans Rempel	Edit and Section 6 Update; LoLo Ver. 1.2.0	HAR	9/15/03
E	James Wicks	Document Format and Edit	JAW	9/30/03
F	Bruce Rovner	Preliminary Release LogicLoader Version 1.4	BR	3/31/04
G	Bruce Rovner	Document Edit/ Test Commands Update LogicLoader Version 1.5/ MOT Pilot Release	JAW	6/08/04
H	Mike Aanenson	Added SOM to Acronyms; Added: mem-copy	JMC	8/30/04
I	James Wicks	Attached SOM Cover page	JAW	10/14/04
J	Robin Bhattacharyya, Bruce Rovner, Michael Erickson, Hans Rempel	Updated for LogicLoader 2.0 release; Command list is now in separate document	ME	6/3/2005
K	Bruce Rovner	Section 4: Added recommendation to user not to erase LogicLoader's RAM space until boot string has been recovered; Added description of exec command boot string structure	HAR	8/22/2005

Table of Contents

1	Introduction to LogicLoader™	1
1.1	Product Brief	1
1.2	Acronyms	2
1.3	Technical Specifications	2
1.4	LogicLoader Command Description Manual	2
1.5	LogicLoader Addendums	2
2	LogicLoader (LoLo™)	3
2.1	LogicLoader Overview	3
2.2	LogicLoader Basics	3
2.3	Using LogicLoader for Debugging	3
2.4	Manufacturing Advantages with LoLo	3
3	The LogicLoader Shell (losh™)	4
3.1	Losh Overview	4
3.2	Losh Basics	4
3.2.1	Using Losh	4
4	Program Loading	6
4.1	Understanding the 'load' Command	6
4.2	Understanding the 'burn' Command	8
4.3	Understanding the 'jump' and 'exec' Commands	8
4.3.1	The 'jump' Command	8
4.3.2	The 'exec' Command	8
4.3.3	Command Example Using 'load' and 'burn' with 'jump' or 'exec'	8
4.4	Understanding the 'update' Command	9
5	Scripting	10
5.1	Scripting Overview	10
5.1.1	Scripting Rules	10
5.2	Launching Scripts	10
5.3	Persistent Script Storage	10
5.3.1	Persisting Scripts with the Echo Command	11
5.3.2	Serial EEPROM Scripts	11
5.3.3	Configuration Block Scripts	11
5.4	Using Boot-time Scripting	11
5.4.1	Boot-time script guidelines	11
5.4.2	Boot Script Magic Strings	11
5.4.3	Exiting a Boot Script	12
5.4.4	Understanding the Echo Command	12
5.4.5	Boot-time Script Example	12
5.5	Conditional Scripting and Variables	12
5.5.1	Variables	12
6	Video Interface	16
6.1	Video Interface Overview	16
6.2	Using the Video Interface after Initialization	16
7	Configuration Block	17
7.1	Configuration Block Overview	17
7.1.1	Initializing	17
7.1.2	Scripting	17
7.1.3	Video	17
7.1.4	Serial with the Configuration Block	17
7.1.5	Ethernet	17
8	YAFFS (Yet Another Flash File System)	19
8.1	YAFFS Overview	19
8.2	Working with YAFFS in LogicLoader	19

8.2.1	Developing a Partition Scheme	19
8.2.2	Formatting YAFFS Partitions.....	19
8.2.3	Adding YAFFS Type Partitions.....	20
8.2.4	Mounting the Partition.....	20
8.2.5	Accessing YAFFS Partitions in an OS	21
8.3	Summary.....	21
9	Appendix: LwIP License Agreement	22

Table of Figures

Figure 3.1: 'Is' Command Columns 5
Figure 4.1: Downloading to RAM 6
Figure 4.2: Downloading to Flash 7



**Developing Products
is as simple as**

A
B
C

- A Application Development Kits
- B Board Support Packages
- C Card Engines

The LogicLoader™ (bootloader/monitor) provides the capability for loading both operating systems and applications. In addition, it provides a full suite of commands for configuring hardware platforms, in-field device management, hardware debug, manufacturing and test. LogicLoader is a key tool in reducing the time for development, manufacturing, and test. This results in an embedded product development cycle in **less time, less cost, less risk ... more innovation.**



PRODUCT HIGHLIGHTS

- Ships standard on all System on Modules
- Conditional Scripting*
- YAFFS (flash file system)*
- Source code is available for purchase, please contact Logic Sales for more information.

* new in version 2.0

CUSTOMER SERVICE

Logic provides technical support for Application Development Kits. Various support packages are available; contact us for more information.

CONTACT

For more information on our Embedded Product Solutions, please contact Logic Sales at product.sales@logicpd.com or 612.672.9495

■ LogicLoader Description

- Multi-platform support (ARM® / ColdFire™ / SH™ / XScale®)
- Fully integrated TCP/IP stack with DHCP and TFTP support
- Supports CompactFlash® FAT file system
- YAFFS (flash file system)
- Conditional Scripting
- Network bootstrap support including setup and download using TFTP
- Customizable and user extendable
- Source code available

■ Operating System Bootstrap

- Load an operating system (OS) from
 - CompactFlash
 - Serial connection
 - Resident Flash Array
 - Ethernet connection
- Fully configure a hardware platform for the operating system
- System on Module initialization
 - Activate custom software functions to initialize hardware before the OS starts
 - Power-on self test capability
- Load multiple Operating Systems: Linux, Windows CE, etc. See available BSPs.

■ In-Field Device Management

- Modify boot actions at run-time using LogicLoader's configuration utilities
- Remote device management eases debugging and upgrading
- CPLD firmware update capability

■ Hardware Debug

- Link in custom test functions to verify custom hardware
- Use a familiar Unix-like interface for debugging the device
- Seamless integration with Microsoft Platform Builder development tools

■ Custom Applications

- Use LogicLoader to burn and jump to any custom embedded application

■ Download Formats

- SREC
- ELF
- BIN
- RAW

■ Manufacturing and Test

- Add in custom functional test software for your specific device needs
- Take advantage of the fast Ethernet connectivity to reduce manufacturing test time

1.2 Acronyms

API	Application Programming Interface
BIN	Microsoft BIN file format
CPLD	Complex Programmable Logic Device
CF	CompactFlash®
DHCP	Dynamic Host Configuration Protocol
EEPROM	Electrically Erasable Programmable Read-Only Memory
ELF	Executable Linkable Format
FAT	File Allocation Table
GPIO	General Purpose Input Output
GNU	GNU is not UNIX
I/O	Input/Output
IP	Intellectual Property
IP	Internet Protocol
JTAG	Joint Test Action Group
LAN	Local Area Network
LwIP	Lightweight implementation of the TCP/IP protocol stack
OS	Operating System
RAM	Random Access Memory
RAW	RAW file format, e.g. absolute binary
RISC	Reduced Instruction Set Computer
SOC	System-on-Chip
SOM	System-on-Module – family name for Card Engine (SOM-Card Engine) and Fire Engine (SOM-Fire Engine)
SRAM	Static Random Access Memory
SREC	Motorola S-Record file format
TCP/IP	Transport Control Protocol/Internet Protocol
TFTP	Trivial File Transfer Protocol
YAFFS	Yet Another Flash File System

1.3 Technical Specifications

Please refer to the component specifications and data sheets applicable to your SOM:

- SOM IO Controller Specification
- SOM Hardware Specification
- Applicable Processor Manual

1.4 LogicLoader Command Description Manual

For a complete description of LogicLoader's 'losh' commands, please see the *LogicLoader Command Description Manual* on Logic's downloads page at www.logicpd.com. The *LogicLoader Command Description Manual* explains how to use each LogicLoader command.

1.5 LogicLoader Addendums

Logic has written a SOM-specific addendum for each SOM that runs LogicLoader. LogicLoader Addendums are located under the "User Manuals" heading on Logic's downloads page.

2 LogicLoader (LoLo™)

2.1 LogicLoader Overview

The LogicLoader (LoLo) is a bootloader/firmware-monitor program developed by Logic Product Development. LogicLoader is designed to initialize an embedded device, load and bootstrap an operating system, and provide a low-level firmware monitor with debugging functionality.

2.2 LogicLoader Basics

Most operating systems rely on an underlying bootloader to initialize a device from its reset condition. In general, operating systems are designed with the assumption that the system will be in a specific pre-defined state before the operating system is started. Some example assumptions might be that system RAM has been initialized and cleared, processor interrupts are disabled, and a timer has been initialized to provide a system tick for the OS. The LogicLoader program initializes Logic Product Development's SOM platforms and prepares them for use by an operating system.

Another basic function of LogicLoader is the capability to upgrade device software (flash memory, CPLD firmware, serial EEPROM contents) after deployment. This "in-field upgrade" ability requires a bootloader program that is capable of loading software images from various sources as well as committing loaded images to non-volatile memory. LogicLoader implements this by giving the system the ability to load system software from flash memory, a CompactFlash storage card, a Local Area Network, or even from a device attached to the system's serial port. LogicLoader also has the ability to upgrade an existing operating system residing in system flash.

LogicLoader was developed to fulfill the need for an OS and processor independent bootloader that can interface with a variety of hardware transports. The GNU development tool chain used to build LogicLoader is cross-platform capable.

2.3 Using LogicLoader for Debugging

LogicLoader implements a feature-rich firmware monitor. Included with LoLo is the LogicLoader shell, also known as "losh." Losh is a command interpreter providing control over system state prior to loading an OS image. It has features such as command recall, command-line editing, automated control via scripting, and diagnostic routines.

Losh includes many commands designed specifically to help software and hardware engineers debug low-level interfaces. For example, formatted data in arbitrary memory locations can be read from, and written to, by using the 'x' and 'w' commands. Other commands run specific tests designed to verify Logic's SOM hardware platforms. All commands return a value to the command line that can be used to conditionally evaluate the command result. Refer to the *LogicLoader Command Description Manual* on Logic's downloads page for a complete description of available commands.

Developers may code their own test programs using the provided GNU development tool chain and use the LogicLoader to load and run their software. This provides the ability to verify and debug hardware interfaces without the overhead of building, downloading, and running large operating system images.

2.4 Manufacturing Advantages with LoLo

LogicLoader can be used with a desktop software utility to load a device's system software on the manufacturing line. This utility is customizable to suit your desired transfer mechanism and additional needs. LogicLoader can also be augmented with functional test software to completely verify a device before it leaves the manufacturing line. Here is an example scenario: LogicLoader could launch a device's final functional test at the end of a manufacturing line, and then load the device's final software image before packaging. Contact Logic for more information on using LogicLoader to streamline manufacturing.

3 The LogicLoader Shell (losh™)

3.1 Losh Overview

Losh is a command interpreter similar to those found in Unix environments. Losh implements a rudimentary network and file system command set, enhanced with custom diagnostic and memory manipulation commands for debugging hardware.

Developers familiar with a Unix-like command line interface should find the losh implementation familiar and easy to work with. Many of losh's commands are patterned after their Unix counterparts and share the same syntax.

3.2 Losh Basics

Losh uses a standard output stream (stdout). By default, stdout refers to a SOM's debug serial port. The output of any command that displays information to stdout (i.e. the 'cat' command) can be viewed using the terminal emulation program connected to the SOM's debug serial port. Likewise, the standard input stream (stdin) by default also refers to the SOM's debug serial port.

The LogicLoader Shell includes a virtual file system that uses standard Unix path names. The highest-level (or root) directory is designated by the identifier '/'. A special sub-directory of the root with the name 'dev' is used to enumerate and interact with system's various peripherals and their associated device drivers.

3.2.1 Using Losh

The losh shell includes a basic command line editing feature and a command history feature. This provides users with a quick way to repeat commands. Using the up and down arrow keys, the user can scroll through the list of previously executed commands. When a desired command is displayed, press the return key to repeat the command. The right and left arrow keys allow a user to position the cursor as desired on the current line so that text can be modified, deleted, or inserted at the appropriate location without having to "backspace" the entire line to access the portion of the command or command set being entered.

Losh includes a user help feature through the 'help' command. Typing 'help' followed by any command name at the losh prompt will display the command's syntax, usage, and an example. This may be especially helpful to users who are just becoming familiar with the LogicLoader shell.

Commands may be run in the background by adding a '&' suffix.

3.2.1.1 Understanding the 'ls' Command

The 'ls' command lists the contents of the current directory. A sample terminal output that results from running the 'ls' command is shown below:




```

losh> ls
:                NK.BIN  4268863
D:                DOC      0
D:                BOOT     0

```

In this example, the columns displayed when the 'ls' command is executed are (in order from left to right): entity attribute, entity name, and entity size. See Figure 3.1, below.

Figure 3.1: 'ls' Command Columns

<i>losh> ls</i>		
<i>:</i>	<i>NK.BIN</i>	<i>4268863</i>
<i>D :</i>	<i>DOC</i>	<i>0</i>
<i>D :</i>	<i>BOOT</i>	<i>0</i>
		
entity attribute	entity name	entity size

The first column, entity attribute, can be blank, "D", "S", "R", "r" or "H". A blank field indicates a normal attribute, a "D" indicates a directory attribute, a "S" indicates a device driver attribute, an "R" indicates a read-only attribute, an "r" indicates reserved bits are set, and an "H" indicates a hidden attribute.

The second column, entity name, is simply the name of the entity as exists on the file system. This name should be used, with attention to case, in any commands referencing the entity.

The third column, entity size, indicates the size (in bytes) of the entity on the storage device.

4 Program Loading

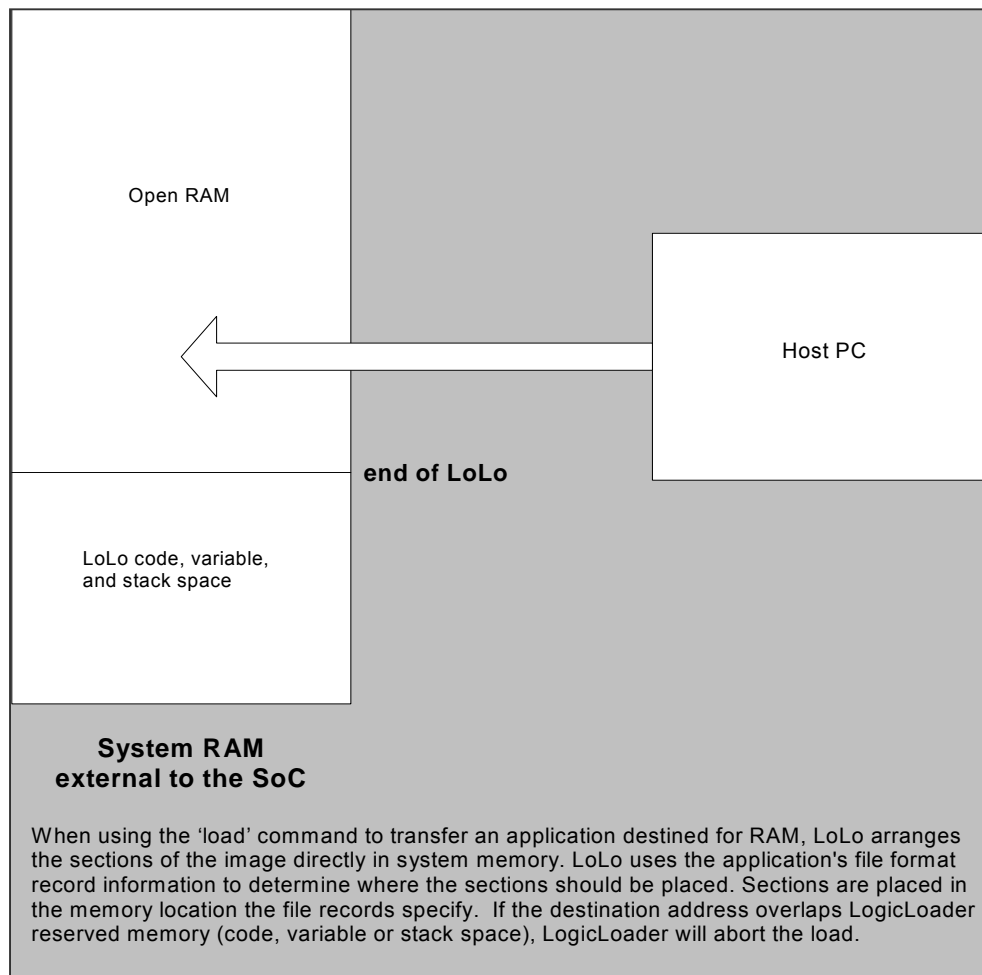
Using LogicLoader to download any application, operating system, or update to a device requires an understanding of the interaction between the 'load', 'burn', 'jump', and 'exec' commands. The purpose of this section is to describe each individual command, and explain the interaction between these commands.

4.1 Understanding the 'load' Command

The purpose of the 'load' command is to transfer an executable image to a device. The image must be in one of the following supported formats: ELF, SREC, RAW, or BIN. The 'load' command uses information inherent to the supported formats (or as entered as part of the command for RAW format) to determine where the downloaded image should be stored in the device's memory. The 'load' command stores the destination address of the downloaded image for later use by the 'burn' command, and stores the program start address for later use by the 'jump' or 'exec' commands. For RAW format, 'load' will store the destination address as the program start address. The image must be destined to reside in either flash memory, system RAM, or on-chip SRAM.

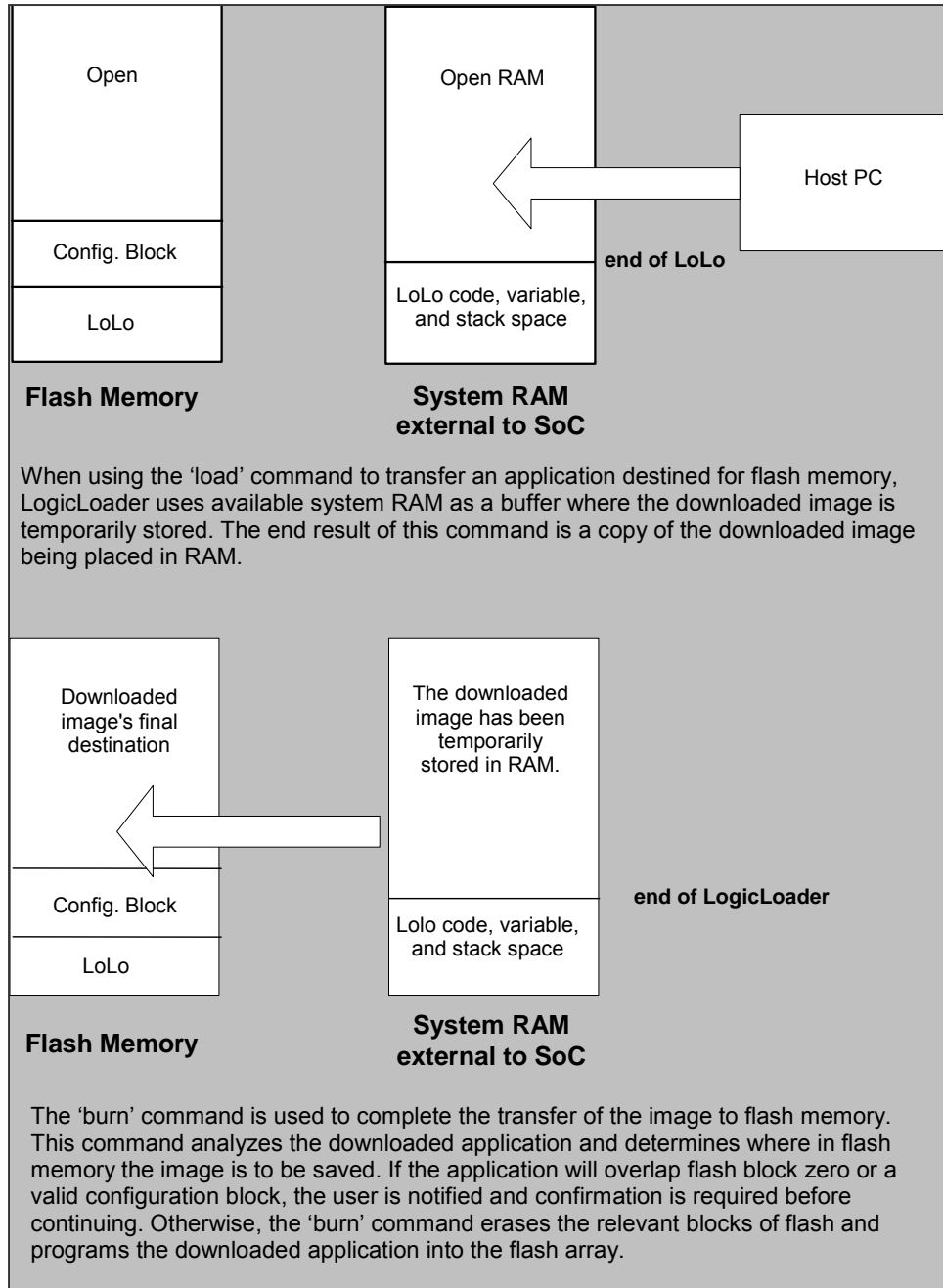
If an image is destined for system RAM or on-chip SRAM, the 'load' command stores the image directly to its run-time location. Refer to *Figure 4.1: Downloading to RAM* for a graphic representation of this process.

Figure 4.1: Downloading to RAM



If a downloaded application is destined for flash memory, the 'load' command transfers the file into a temporary RAM buffer on the device. The transferred image may be programmed into flash using the 'burn' command after the transfer is complete. Refer to *Figure 4.2: Downloading to Flash* for a graphic representation of this process.

Figure 4.2: Downloading to Flash



4.2 Understanding the 'burn' Command

The 'burn' command should only be used following the successful download of a binary image destined for flash. If the 'load' command is used to download a flash image, the image is temporarily stored in a reserved section of system RAM. The 'burn' command is responsible for actually erasing the necessary blocks and programming the downloaded image into flash at the destination address. Refer to *Figure 4:2 Downloading to Flash* for more information.

4.3 Understanding the 'jump' and 'exec' Commands

LogicLoader provides two different ways to transfer execution to your application. The 'jump' command is more useful for launching and debugging an application that will be relying on LogicLoader or an operating system to setup the runtime environment. The 'exec' command is more useful for launching an application such as an operating system that will take over total control of the hardware and the environment. The differences between the 'jump' and 'exec' command are that 'exec' can pass a command line argument to the program being executed and that 'exec' disables interrupts, the cache, and the MMU (if present).

4.3.1 The 'jump' Command

The 'jump' command is an assembly-level jump to the starting instruction of a program. If 'jump' is executed without a parameter, LogicLoader will jump to the program start address of the last program loaded to system RAM (if any). If an address is passed in, the 'jump' command will jump to the specified address. After a 'jump' command is performed, LogicLoader continues to execute in the background. LogicLoader does not set up a run-time environment for a program, rather the program inherits LogicLoader's current environment. It is the software engineer's responsibility to ensure that the hardware is setup in the desired manner.

This example may be used when writing a function that LogicLoader will **jump** to:

```
int my_jump_function(void);
```

4.3.2 The 'exec' Command

The 'exec' command is an assembly-level jump to the starting instruction of a program that will pass in three arguments. If 'exec' is executed without a parameter, LogicLoader will jump to the program start address of the last program loaded to system RAM (if any) and pass in a pointer to an empty string. If both an address and command line are specified, the 'exec' command will jump to the specified address and pass a pointer to the command line provided. The 'exec' command will disable interrupts, the cache, and the MMU (if present) prior to executing the jump.

The 'exec' command passes the command line argument via a pointer to memory that has been allocated from LogicLoader's heap. Any application or OS code must preserve the command line, or finish using the command line arguments, before reclaiming LogicLoader's memory space for its own use. Because the 'exec' command shuts off the MMU, the image must have a virtual address that maps directly to its physical address since the entry address that 'exec' jumps to will always be a physical address.

This example may be used when writing a function that LogicLoader will **exec** to:

```
int my_exec_function(unsigned int arg1, unsigned int arg2, char *cmd_string);
```

The first two arguments have fixed values for legacy reasons: arg1 = 0, and arg2 = 997. The third argument will be a pointer to the command line as described above.

4.3.3 Command Example Using 'load' and 'burn' with 'jump' or 'exec'

An application program that is written for the Zoom Development Kit can be linked to reside in flash or ram.

First, let's assume that we have built an application for flash. To properly store this program in flash, issue the 'load' command followed by the 'burn' command. Make note of the program start address (for example: 0x400d0100) so that you can jump to the program after a reset. Once the image has been burned to flash, you may enter the 'jump' or 'exec' command specifying 0x400d0100 as the argument at anytime but you can take a shortcut if you have not reset the board since the 'load' command will store the program start address. A valid sequence would be as follows:

1. `losh> load elf`
This transfers the image to the device.
2. `losh> burn`
This programs the image into flash at the destination address stored by the 'load' command.
3. `losh> jump or exec`
This will work because the 'load' command stored the program start address. Both the destination address and the program start address will be valid until the next reset, or the next use of the 'load' command.

After a reset the program may be launched at any time using the 'jump' or 'exec' command:

```
losh> jump 0x400d0100
```

or

```
losh> exec 0x400d0100 –
```

Next, let's assume that we have built an application for RAM. To properly load and execute an application out of RAM, issue the 'load' command followed by the 'jump' or 'exec' command. A valid sequence would be as follows:

1. `losh> load elf`
This transfers the image to the device.
2. `losh> jump or exec`
This will work because the 'load' command stored the program start address. The program start address will be valid for this program until the next reset, or the next use of the 'load' command.

Keep in mind that the option of specifying the program start address, as shown in the flash example, is available as well.

4.4 Understanding the 'update' Command

Logic deploys software or firmware updates in the form of update files (.upd extension). To deploy an update file, use the 'update' command. If a filename/path parameter is not passed to the 'update' command, the system will assume that stdin is being used to send the update file to the system. When the update command is activated, after the system has received the .upd file, it automatically launches the file and performs the actions required.

Update files are comprised of self-extracting applications that, once activated by the update command, run and perform whatever function the application was coded to carry out. This allows a single "update" command to perform a variety of different actions from a self-contained file with minimal user interaction.

The procedure to update LogicLoader with the 'update' command differs from the 'load/burn' procedure in this way: only one command implements the entire update process without any user interaction or confirmation.

5 Scripting

5.1 Scripting Overview

Scripts can be used to automate any commands or command sequences entered on the command line. Scripts are comprised of a simple text file with a listing of commands that the user wishes to automatically execute in sequence.

It is possible to write a script that automatically evaluates a command's return parameter and executes an independent control path based on the value it received. This is called conditional scripting. Conditional scripting provides the user with a high level of control over how a system boots up and runs without having to develop additional low-level software. Conditional scripting is described in more detail below.

5.1.1 Scripting Rules

Basic scripting rules are as follows:

- Enter commands into the script file with the same syntax used on the losh command line
- Separate commands with a semi-colon or a newline
- Use the command "exit" to end the script (this tells the command interpreter to stop parsing the script)

5.2 Launching Scripts

The process of launching a script manually or post-boot time employs the 'source' command. For example: the command "source /cf_card/myscript.txt" will execute the script stored in the file "myscript.txt" on a mounted CompactFlash card. For more information on the source command, please reference the *LogicLoader Command Description Manual* document.

The process of auto-launching scripts on startup is referred to as "boot-time scripting." Boot-time scripts are the primary mechanism used for automatically launching an OS or application when deploying a product to the field. Their capability is the same as other scripts, with their ability to be automatically run at startup differentiating them from normal scripts. One can think of a boot-time script fulfilling the same role as an "autoexec.bat" file commonly found on desktop operating systems. Boot-time script usage is described more thoroughly below.

A third way to launch a script is to simply "send" it to the system while LogicLoader is waiting at the losh prompt. If the script file is sent over the terminal emulator connection to the losh shell, the script will be entered on the command line as if typed in by the user. If the script being sent incorporates a carriage return at the end of the script, the command line will launch the script when it receives the carriage return. This type of script launching is primarily used during development when the developer wishes to send a number of development commands to LogicLoader in sequence. For example: a command sequence initializes the Ethernet interface, downloads a Windows CE OS image, and then launches the OS image with a specific command line.

5.3 Persistent Script Storage

In order for a script to persist across power cycles, the script must be stored to a local, non-volatile memory device on the system. There are a number of different persistent storage locations that can be used to store a script on each system. The primary storage mechanisms supported by LogicLoader are the serial EEPROM, the resident flash array (dev/config or YAFFS), and the memory-mapped CompactFlash interface. Because different SOM's may not have one or more of these interfaces available in hardware, please refer to the individual SOM's *LogicLoader User's Manual Addendum* document for specific persistent storage interface support.

5.3.1 Persisting Scripts with the Echo Command

The 'echo' command can be used to store a script in the serial EEPROM or /dev/config. To include a new-line in the first argument to 'echo', it is necessary to enclose the whole argument in double-quotes.

5.3.2 Serial EEPROM Scripts

The system's serial EEPROM is one persistent storage area that supports the storage and execution of scripts. The serial EEPROM is the primary boot-time script storage location. Boot-time scripts stored to the serial EEPROM are typically fairly short and may re-direct to a secondary script on an interface capable of a larger storage capacity.

To store a script to the serial EEPROM interface (/dev/serial_eeprom), use the "echo" command. An example of using the "echo" command to store information to the serial EEPROM is shown below:

```
echo "LOLOmount fatfs /cf; source /cf/B.BAT; exit;" /dev/serial_eeprom
```

5.3.3 Configuration Block Scripts

The system's configuration block is another persistent storage area that supports the storage and execution of scripts. The configuration block is located in system flash memory and is the secondary boot-time script storage location on systems with serial EEPROM.

Store a script to the configuration block interface (/dev/config) by using the 'echo' command or the 'config S' command. Here is an example of using the 'echo' command to store a script to the configuration block:

```
echo "LOLOmount fatfs /cf; source /cf/B.BAT; exit;" /dev/config
```

Note: The configuration block must be initialized before using it for scripting commands. For more information on how to create and use the configuration block, please see Section 0.

5.4 Using Boot-time Scripting

It is possible to execute a script automatically at startup. This is useful for making the device jump into an operating system or other program when powered-on without requiring manual command-line input. This functionality can be described as being equivalent to the system automatically calling the 'source' command on one of the boot-capable devices.

5.4.1 Boot-time script guidelines

All of the commands available in LogicLoader are also available to boot-time scripts. As in normal scripts, a semi-colon must be used to separate commands and the exit command must be used to terminate a boot-time script.

In order for a script to be boot-capable, the script must be stored in a boot-capable location and must contain the necessary "magic" string prefix. A boot-time script may reside either in the on-board serial EEPROM or in the flash-based configuration block. The order of boot-time execution is first the EEPROM, then the configuration block. In order to differentiate between auto-booting scripts and non auto-booting, LogicLoader checks the first four bytes of the boot-capable devices to see if they contain a "magic" string indicating that the following script should run automatically.

5.4.2 Boot Script Magic Strings

The "magic" string for LogicLoader is "LOLO" for silent execution or "VOLO" for verbose script execution. If the string "LOLO" prefixes the boot script, the script's commands and terminal output will be completely silent. By using "LOLO" as the prefix, it is possible to fully boot the system without ever sending any information out the debug serial port. If "VOLO" is used as the

boot script prefix, the boot script commands, return codes, and other “normal” information is displayed via the serial port as if the script was running post-boot time.

5.4.3 Exiting a Boot Script

A common need is to abort the execution of a boot script in order to exit into the command line for additional debugging, development, or simply to change the boot script. The primary way to accomplish this is by holding the 'q' key down in a terminal emulator program attached to the device's debug serial port.

The system does pause for one/half of a second to read from debug serial port to determine if an abort request is being made. Some of Logic's SOM products implement an external mode line that allows LogicLoader to ignore the assertion of the 'q' key – thereby skipping the one/half second wait time and decreasing the overall boot time of the system when a boot script is desired. For more information on the hardware line that provides this functionality, check the *LogicLoader User's Manual* addendum for your hardware

5.4.4 Understanding the Echo Command

The 'echo' command can be used to store a script in the serial EEPROM or /dev/config. To include a new-line in the first argument to 'echo', it is necessary to enclose the whole argument in double-quotes. The editing can be a little tricky, and it is recommended to avoid typing errors while creating this script.

5.4.5 Boot-time Script Example

The following example creates a simple LoLo boot script that first mounts the CompactFlash card and then runs a second script “B.BAT” on the CompactFlash card that automates software.

```
losh>echo "LOLOmount fatfs /cf; source /cf/B.BAT; exit;" /dev/serial_eeprom
```

or

```
losh>echo "LOLOmount fatfs /cf; source /cf/B.BAT; exit;" /dev/config
```

5.5 Conditional Scripting and Variables

5.5.1 Variables

The LogicLoader's shell supports the concept of shell variables. The syntax and usage of these variables are patterned after the BASH shell.

5.5.1.1 Variable Names

A variable name may be any sequence of letters, numbers, or the underscore token.

5.5.1.2 Variable Assignment

A variable is created and assigned a value by using the '=' operator. For example:

```
losh> foo = 1
```

creates a new variable named 'foo' and assigns it the value of '1'. Once a variable has been created, it may be assigned a new value at any time by using the '=' operator again.

5.5.1.3 Internal Representation

Variables are internally represented as strings. For example:

```
losh> foo = 1
```

internally points the variable 'foo' at a sequence of characters equivalent to: 0x31 0x00. Because variables are treated as strings, commands may be aliased as variables. For example:

```
losh> e = echo
losh> msg = "Hello World"
losh> $e $msg
Hello World
```

Notice the quotes used to ignore white space. If the created variable will be assigned to more than one token, the tokens must be included in double-quotation marks.

5.5.1.4 De-referencing a Variable

To dereference a variable, that is, to access a variable's assigned value, use the '\$' operator. For example:

```
losh> foo = "Hello World"
losh> echo $foo
Hello World
```

The '\$' operator causes the shell to substitute the variable with the string value assigned to it. In some cases, a variable's assigned value will be converted into a numeric value. This occurs when the shell is evaluating a conditional expression. This is described in more detail below.

Please note: enclosing a sequence of tokens within double quotes binds them together into a single token. For example:

```
losh> e = "echo Hello World"
losh> $e
echo Hello World: command not found
```

This will not work because the parser only evaluates the string once. Thus, instead of being split up into three distinct tokens, the double quotes cause the tokens to be bound and treated as one.

5.5.1.5 Built-in Variables

The shell contains two built-in variables, '?' and '@'.

The '?' variable is assigned to the return value of the last command executed. By convention, all shell commands return zero to indicate that it completed successfully and a non-zero error code to indicate a failure. To view a command's return value, use the 'echo' command and the value of the '?' variable. For example:

```
losh> mount fatfs /cf # Mount a FAT file system.
losh> echo $? # Display the value returned from the mount command.
```

The '@' variable is an auxiliary variable that is set by some commands. For instance, the 'echo' command sets this value to the number of characters that it wrote. Therefore:

```
losh> echo "Hello"
losh> echo $@
0x5
```

The number '5' is printed because the string "Hello" contains five characters.

Please reference the *LogicLoader Command Description Manual* for specific command descriptions in order to learn which commands set the '@' variable, and if so, the usage of these commands.

5.5.1.6 Conditional Scripting

LogicLoader's shell supports an if-else-endif programming construct. The syntax for an if-statement and an if-else statement is shown below:

```
if (expression)
    action
endif

if ( expression )
    action-1
else
    action-2
endif
```

Note: parentheses are not required around the expression, but they are encouraged to improve readability of the script. Similarly, tabs and new lines are not needed. The various elements of the construct may be separated by the ';' operator if so desired. For example:

```
losh> if expression echo "pass"; else echo "fail"; endif
pass

or

losh> if expression echo "pass"
    else echo "fail";
    endif
```

5.5.1.7 Expressions

An expression is defined as a number or a combination of a logical operator and a number or numbers. If a variable has been defined and is being dereferenced in an expression, its value is converted to a number. An expression evaluates to true if the result is non-zero and false if the result evaluates to zero. Therefore, the simplest expressions would be:

```
if ( 1 ) # evaluates to true.
if ( 0 ) # evaluates to false.
```

5.5.1.8 Using Shell Variables

```
losh> foo=1
losh> bar=0x0
if ( $foo ) # evaluates to true.
if ( $bar ) # evaluates to false.
```

The other logical operators supported by the shell are:

```
'&' bitwise and
'|' bitwise or
'^' bitwise exclusive-or
'~' bitwise not
'!' logical not
```

Proceeding by example:

```
if ( 1 & 0 ) # evaluates to false
if ( 1 | 0 ) # evaluates to true
```

```

if ( 0x01 ^ 0x02 )      # evaluates to true
if ( ~0x0 )             # evaluates to true
if ( ~0xffffffff )     # evaluates to false
if ( ! 1 )              # evaluates to false

```

As mentioned above, the shell exports two built-in variables. These are '?' and '@'. The variable '?' holds the return value of the last command executed. Therefore, constructs like the one below can prove to be very useful:

```

mount fatfs /cf
if ( $? )
    # Save current return values because 'echo' will overwrite them
    s_q = $?
    s_a = $@
    echo "Error, mount failed error codes: "
    echo $s_q
    echo $s_a
else
    echo "Mounted FAT file system at point '/cf'"
endif

```

Note: in the case of an error, the values of the '?' and '@' variables are saved. This is because the first call to the 'echo' command will overwrite the value of those variables.

5.5.1.9 Comments

In order to make it easy to self-document scripts, the shell recognizes and ignores comments. A comment begins with the character '#' and extends to the end of the current line.

5.5.1.10 Numbers

The shell recognizes the following number formats:

- decimal
 - contains the characters 0-9
 - does not start with a zero
- octal
 - contains the characters 0-7
 - starts with a zero
- hexadecimal
 - contains the characters; 0-9, a-f, or A-F
 - starts with the sequence '0x' or '0X'

6 Video Interface

6.1 Video Interface Overview

LogicLoader includes the following video commands to configure the video controller:

- video-clear - clears the default video screen (sets the frame buffer to a monolithic color)
- video-close - turns off and un-initializes the default video device
- video-fb - sets or displays the video frame buffer address
- video-init - connects and initializes default video device settings, but does not enable the controller
- video-off - turns off an initialized display
- video-on - turns on an initialized display
- video-open - connects and initializes default video device settings and enables the display controller (equivalent of video-init and video-on)

6.2 Using the Video Interface after Initialization

Once the display has been initialized with either the 'video-open' or the 'video-init' commands, any of the drawing commands can be used. The 'video-fb' command allows the user to change the frame buffer address.

After executing the 'video-fb' command to change the frame buffer address, all drawing commands will use the new frame buffer address instead of the default. The 'video-init' command can be used to connect and initialize the video controller without enabling the video display. Then use the 'bitmap' command to draw to different areas in memory prior to using the 'video-on' command to turn on the display. A typical command sequence might look like the following:

```
losh> video-init 7 16
video-init display: width: 640 height: 480 bpp: 16 disp: 7
losh> bitmap TEST1.BMP 0xc0400000
losh> bitmap TEST2.BMP 0xc0600000
losh> video-fb 0xc0400000
losh> video-on
.....other command sequences
losh> video-fb 0xc0600000

.....other command sequences
losh> video-off
```

If using the configuration block, up to eight uniquely named custom screen settings can be saved. The stored settings include the frame buffer address so that the frame buffer will be initialized to a user specified address upon executing the 'video-open' or 'video-init' command. The current frame buffer address can be ascertained by issuing the 'video-fb' command.

7 Configuration Block

7.1 Configuration Block Overview

Logic has added an optional configuration block that is located in the first 64K of the second 256K block of flash, immediately following the location of LogicLoader. The purpose of the configuration block is to allow our customers the ability to store larger scripts, change the baud rate of the debug serial port, store default settings for the Ethernet, and save custom LCD controller settings. The script area accommodates 16Kbytes of script storage space, and the peripheral settings allow for up to eight different settings each to be saved for the video, Ethernet, and serial. Custom settings can be downloaded and saved, or pre-programmed at the factory.

The configuration block is designed to be easily accessible from a customer's application. The structure and field definition header files are available from Logic. Each section has its own checksum, and there is also a checksum and version id for the entire configuration block. It is possible to store additional customer-defined settings to the configuration block for later use by customer software. In order to do this, a custom configuration block must be created. Please contact Logic for more information.

The configuration block is optional for most SOM's. Normal operation, with the default settings, is available to customers who do not wish to use the configuration block.

7.1.1 Initializing

The configuration block must be initialized on systems that have never implemented a configuration block. If, for some reason, the system's configuration block needs to be cleared, this initialization step will also provide for that.

Enter 'config CREATE' at the losh prompt to initialize (or re-initialize) the configuration block. The configuration block will be located in the 64K of flash immediately following the LogicLoader flash storage location.

7.1.2 Scripting

The default mode of the configuration block is scripting. Once the configuration block has been initialized, users can 'cat' and 'source' /dev/config in order to view or execute their script. The 'echo' command may be used to create the script, or larger scripts can be downloaded and saved using the 'config' command with the 'S' option. The scripts in the configuration block can be boot scripts or ordinary scripts. For more information on scripting, please refer to Section 0.

7.1.3 Video

A customer may save up to eight custom LCD screen settings, including the frame buffer location, by using the 'config' command with the 'V' option. In order to store a custom setting, set up the controller as desired and then enter 'config V screen_name X Y' to store it (where X and Y are dimensions of the screen – e.g. 640 and 480). Once this configuration step is implemented, the system can use the command sequence 'video-open screen_name depth' to use the new configuration.

7.1.4 Serial with the Configuration Block

A customer can change the baud rate of the debug port with the 'B' option. The setting will be stored under the debug serial port's UART index, and will automatically be used after the next reset. Only valid settings will be allowed.

7.1.5 Ethernet

A customer may save a default Ethernet setting that consists of a MAC address, and IP address, a subnet mask, and a gateway.

To accomplish this, set up the controller using the 'ifmac' and the 'ifconfig' commands, and then use the 'config' command 'E' option with an index (e.g. 'config E 0') to save the settings. The index used must correspond to the hardware name index used in the ifconfig command (e.g. sm0 – where 0 is the valid index).

Use the default setting by typing 'ifconfig sm0 /dev/config' (where sm0 indicates index 0) to reload the values stored in the configuration block. In general, the MAC address is stored in the config block for viewing purposes only, and the actual MAC address used is accessed directly by the Ethernet chip out of its dedicated serial EEPROM. On SOM's without a serial_eeprom, the config block is required if the customer wishes to use the Ethernet feature from within LogicLoader.

The configuration block allows storage for up to eight Ethernet interface configurations, but the config E command is only able to store information for SOM hardware that is supported by LogicLoader.

8 YAFFS (Yet Another Flash File System)

8.1 YAFFS Overview

The acronym YAFFS stands for the phrase "Yet Another Flash Filing System." YAFFS was developed by a company named Aleph One Limited and incorporated by Logic Product Development into the LogicLoader (LoLo) software program.

Logic selected YAFFS to fill its file system requirements due to the flexible nature of the program, its licensing scheme, and the fact that it is available for Linux, Windows CE, and other operating systems. YAFFS also allows LogicLoader and an RTOS to view and modify the same partition. It also makes it easier for customers to work with embedded flash technology and perform in-field updates.

Note: The partition entries for YAFFS partitions are not persistent -- *they must be restored on each boot*. However, the partitions and data remain persistent.

8.2 Working with YAFFS in LogicLoader

8.2.1 Developing a Partition Scheme

The LogicLoader may mount up to four YAFFS partitions at a time. Customers should design a partitioning scheme which suits their individual needs. The following limits are imposed on partitions:

- Each partition must have a unique name.
- Each partition must exist on local flash accessible from LogicLoader's '/dev/flashx' device file (where 'x' is an instance index). For example: /dev/flash0 or /dev/flash1.
- Each partition must span at least 4 physical flash blocks.
- A partition must *not* overlap the flash blocks that contain LogicLoader or its configuration block. For information about the location of these items, check the *LogicLoader User's Manual* addendum for your hardware.

For the remainder of this document, the following partitioning scheme for demonstration purposes will be used:

- A partition named 'boot' which contains a bitmap and operating system image and spans the address space below:
 - * start: 0x000C0000
 - * length: 0x00800000 (8 MBytes)
- A partition named 'data' which contains customer specific data.
 - * start: 0x00900000
 - * length: 0x00400000 (4 MBytes)

8.2.2 Formatting YAFFS Partitions

All file systems need to be formatted before they can be mounted. Because YAFFS was designed from the ground up to work with embedded flash technologies, it understands an 'erased' flash device to be both formatted and empty. To prepare your partition for mounting, simply use LogicLoader's 'erase' command to erase the area of flash where the partition is to be located.

Using the example partition scheme in the "Developing a Partition Scheme" section, above, the partitions could be prepared for initial use by erasing the regions of the flash device spanned by them.

For example (LH7A404-11 system address used):


```

losh> erase 0x000C0000 0x00800000
losh > erase 0x00900000 0x00400000

```

Warning: Erasing flash blocks that will be used for YAFFS partitions will erase everything in those areas of flash. It is not required to format the partition every time the device is rebooted. The partition should only be formatted when an entirely new YAFFS partition is created, or when the data on a stored partition needs to be completely erased.

8.2.3 Adding YAFFS Type Partitions

LogicLoader maintains a partition table in RAM. Before a YAFFS partition can be mounted, it must be added to the partition table. To do this, the 'add-yaffs' command is used. The 'add-yaffs' command takes the following arguments:

- <name> a unique string which identifies the partition
- <type> type of flash device the partition resides on
- <start> the physical starting address of the partition
- <length> the length (in bytes) of the partition

Continuing with the example partitions above, LogicLoader can be instructed to add the partitions by executing the commands as shown below (LH7A404-11 system address used):

```

losh> add-yaffs boot nor 0x000C0000 0x00800000
losh> add-yaffs data nor 0x00900000 0x00400000

```

Note: The above steps must be performed every time LogicLoader boots. Because LogicLoader keeps the partition table in RAM, the existence and locations of YAFFS partitions does not persist across resets or power cycles.

8.2.4 Mounting the Partition

To mount a partition in the partition table, the 'mount' command is used. That command takes the following arguments:

- <fstype> the type of filesystem being mounted ('yaffs' here)
- [drive addr] not used when mounting a YAFFS partition
- <point> the name of the YAFFS partition

For example:

```

losh> mount yaffs /boot
losh> mount yaffs /data

```

Of note is that the 'drive addr' argument is not used when mounting a YAFFS partition. Also of note is that the 'point' argument needs to correspond to the name of the partition (as defined by the add-yaffs command) preceded by a forward slash. LogicLoader needs to mount all YAFFS partitions at the root-directory level. Thus, a partition added using:

```
'add-yaffs boot ...'
```

will be mounted using:

```
'mount ... /boot'.
```

Note the absence of the '/' character during the 'add-yaffs' command and its presence during the 'mount' command.

8.2.5 Accessing YAFFS Partitions in an OS

A key advantage of the read/write YAFFS filesystem capability at the LogicLoader level is the ability to share data stored in the filesystem with an OS environment. If an OS environment (e.g. Linux, WinCE, VxWorks) implements YAFFS as an OS-accessible file-system, any files available to LogicLoader are also available to the OS, and vice-versa.

This contributes to significant benefits in the areas of system software upgrades (including OS upgrades) splash screen changes, script modifications, and other boot-time data that may need to be updated.

8.3 Summary

To use the YAFFS file system within LogicLoader, follow these steps:

- 1) Decide on a partitioning scheme.
- 2) Format the partitions by erasing the associated flash blocks.
- 3) Add the partitions to LogicLoader using the 'add-yaffs' command.
- 4) Mount the partitions using the 'mount' command.

Steps 3 and 4 must be repeated every time the system is booted. If the YAFFS partitions are frequently accessed, consider implementing steps 3 and 4 via a boot script. Step 2 only needs to be performed when creating a brand new partition or when the contents of an existing partition need to be completely erased.

Note: a partition is persistent. Re-adding a partition at boot-time restores access to previously saved data. Flash blocks must be erased to permanently remove a partition; otherwise, it can be recovered across boots.

Keep in mind the following when working with YAFFS and LogicLoader:

- Ensure that a partition name does *not* begin with a '/'. LogicLoader's virtual filesystem uses the forward slash to indicate the root directory.
- Ensure partitions do not overlap each other, LogicLoader, or the configuration block.
- Ensure that a partition is erased before it is mounted for the first time.

9 Appendix: LwIP License Agreement

LogicLoader uses the open source LwIP stack for networking support. The LwIP license requires the inclusion of the following license to satisfy Condition #2 below:

Copyright (c) 2001, 2002 Swedish Institute of Computer Science. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This file is part of the lwIP TCP/IP stack.

Author: Adam Dunkels <adam@sics.se>