Matchbox: A simple filing system for motes

David Gay
Version 1.0

August 21, 2003

1 Overview

Matchbox, the mote filing system is designed to provide a simple filing system for mote-based applications. The design has the following goals:

- Reliability, specifically:
  - data corruption should be detected
  - catastrophic failure (e.g., power off) should never lead to meta-data corruption, data loss should be limited to files being written at the time of failure. This implies that file renaming and deletion are atomic operations.
- Low resource consumption (especially RAM)
- Consider typical restrictions on flash memories (there are limitations on the number of times a block can be erased, on the minimum erasure size, and on multiple writes within a block - these vary depending on flash technology)

Most of the design is based on what is not necessary. We do not need:

- Security in any form
- A hierarchical filing system
- Random file access
- Multiple readers or writers of the same file
- Many files open simultaneously, however we do wish to support (at least) having two different files open, one for reading and one for writing (this supports file copy/edit operations)

Expected clients of this filing system are TinyDB (and the Generic Sensor Kit), for logging data sets to permanent storage and for storing metadata, and the virtual machine for storing programs.

Matchbox is currently implemented for mica family motes (these motes use the Atmel AT45DB041B flash chip).

2 Matchbox

Matchbox is a flat filing system that supports only sequential reads and appending writes. Files are unstructured (simply a sequence of bytes). Space can optionally be preallocated for a file (ensuring both that space exists in permanent storage, and potentially reducing write overhead).

The same file cannot be opened for both reading and writing. Files (that are not open) may be renamed or deleted. In Unix style, renaming a file A as an existing file B also deletes B (the rename will fail if B is open).

3 Matchbox API

The Matchbox component (`Matchbox`) has the following specification:

```
provides {
  interface StdControl;
  interface FileDir;
  interface FileRename;
  interface FileDelete;
  interface FileRead[uint8_t fd];
  interface FileWrite[uint8_t fd];
}
uses {
  event result_t ready();
}
```

This interface is covered in three parts: initialisation, directory operations and file operations. The detailed interface specifications are in Appendix A.

3.1 General Operation

Nearly all file system operations are split-phase, and fail (with result `FAIL`) at initiation if another operation is already in progress. Also, the filing system will be extremely unhappy (i.e., fail) if another component uses the external flash during a file system operation.

The `fileresult_t` type is used in most split-phase completion events to report the success or failure of the requested operations. The following results are defined:

- `FS_OK`: operation succeeded.
- `FS_NO_MORE_FILES`: end of directory reached.
- `FS_ERROR_NOSPACE`: filing system is full (there may be a little space left, but not enough to guarantee completion of the requested operation while preserving the filing system from corruption in case of failure).
- `FS_ERROR_FILE_OPEN`: the requested file is already open for reading or writing.
- `FS_ERROR_NOT_FOUND`: the requested file is not found.
- `FS_ERROR_BAD_DATA`: corrupt data was found (should only occur for file operations).
- `FS_ERROR_HW`: a problem occurred when accessing the flash chip.

The Matchbox files are in the `tos/lib/FS` directory, and depend on files in `tos/lib/Util`. Matchbox programs must therefore be compiled with `-I%T/lib/FS -I%T/lib/Util` options.

3.2 Initialisation

Matchbox is initialised via the usual `StdControl` interface, however it is not immediately ready for use. File system operations can only be initiated after the `ready` event is signaled.

3.3 Directory Operations

Directory contents are listed by calling the `start` command in `FileDir` (not split-phase), then calling `readNext` to get each subsequent file. The files are returned in the `nextFile` event. The file system is considered busy from the call to `start` until `nextFile` signals an error (normally the `FS_NO_MORE_FILES` error).

The number of free bytes in the filing system is returned by the `freeBytes` command.

Files can be renamed and deleted via the split-phase `FileDelete` and `FileRename` interfaces. Open files can not be renamed or deleted. Renaming a file X over an existing file Y will delete Y (but will fail if Y is currently open).

3.4 File Operations

The `FileRead` and `FileWrite` interfaces are parameterised. Each interface instance corresponds to a separate *file descriptor* and allows a separate file to be opened. File descriptors for reading and writing are separate; file descriptors for reading are obtained with `unique("FileRead")`, file descriptors for writing are obtained with `unique("FileWrite")`. For instance, an application that simultaneously needs two files open for reading and one for writing might have the following configuration:

```
configuration MyApp { }
implementation {
  components Matchbox, MyCode, ...;

  MyCode.FileRead1 -> Matchbox.FileRead[unique("FileRead")];
  MyCode.FileRead2 -> Matchbox.FileRead[unique("FileRead")];
  MyCode.FileWrite -> Matchbox.FileWrite[unique("FileWrite")];
  ...
}
```

The `FileRead` interface defines the operations for opening (`open` command, `opened` event), closing (`close` command, which is not split-phase), reading (`read` command, `readDone` event) and finding the remaining bytes in a file (`getRemaining` command, `remaining` event).

The `FileWrite` interface defines the operations for opening (`open` command and `opened` event), closing (`close` command and `closed` event), appending (`append` command and `appended` event), synchronising (`sync` command and `synced` event) and reserving space for a file (`reserve` command and `reserved` event). The `opened` event reports the current size of the file. Appends are only guaranteed to be flushed after a `closed` or a `synced` event. Space can be pre-reserved for a file using the `reserve` command, this guarantees that subsequent appends up to the requested size will not fail due to lack of space, and leads to somewhat faster appends.

4 Remote Matchbox Access

The `Remote` component provides for remote Matchbox access, assuming a reliable communications channel. It is thus easiest to use it via the UART (e.g., when debugging), though it can be used via the radio if some other component is used to provide reliable, bi-directional packet transmission.

The `Remote` component gives access to all file system operations. The java `net.tinyos.matchbox` package provides some low-level infrastructure to talk to the `Remote` component, while the `net.tinyos.matchbox.tools` package provides some simple java programs to list the directory (`Dir.java`), delete or rename a file (`Delete.java` and `Rename.java`), and copy files in and out of Matchbox (`CopyIn.java` and `CopyOut.java`).

5 Acknowledgments

Timothy Roscoe came up with the Matchbox name and Wei Hong helped define the filing system requirements.

A Interfaces

```
// $Id: FileDir.nc,v 1.1.4.2 2003/08/18 22:09:47 cssharp Exp $

/* tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 */
/**
 * List files in filing system
 */
interface FileDir {
  /**
   * List names of all files in filing system. Filing system is busy
   * until FAIL is returned from <code>readNext</code> or no more
   * files remain.
   * @return
   *   SUCCESS: files can be read with <code>readNext</code>
   *   FAIL: filesystem is busy
   */
  command result_t start();

  /**
   * Return next file in filing system
   *
   * @return
   *   SUCCESS: attempt proceeds, <code>nextFile</code> will be signaled
   *   FAIL: no file list operation is in progress.
   */
  command result_t readNext();

  /**
   * Report next file name.
   * @param filename One of the files in the filing system if
```

```
   *   <code>result</code> is FS_OK, NULL otherwise. The storage for
   *   filename only remains valid until the end of the event.
   * @param result
   *   FS_OK filename is the next file.
   *   FS_NO_MORE_FILES No more files.
   *   FS_ERROR_xxx Filing system data is corrupt.
   * @return
   *   SUCCESS: continue reporting file names<br>
   *   FAIL: abort the file lising operation<br>
   * If the <code>result</code> is an error or no more files, the file
   * listing operation terminates.
   */
  event result_t nextFile(const char *filename, fileresult_t result);

  /**
   * @return Number of bytes available in filing system.
   */
  command uint32_t freeBytes();
}
```

```
// $Id: FileDelete.nc,v 1.1.4.2 2003/08/18 22:09:47 cssharp Exp $

/* tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 */
/**
 * Delete a file
 */
interface FileDelete {
  /**
   * Delete a file
   * @param filename Name of file to delete. Must not be stack allocated.
   * @return
   *   SUCCESS: attempt proceeds, <code>deleted</code> will be signaled<br>
   *   FAIL: filesystem is busy
   */
  command result_t delete(const char *filename);

  /**
   * Signaled at the end of a file delete attempt
   * @param result
   *   FS_OK: file was deleted<br>
   *   FS_ERROR_xxx: delete failure cause
   * @return Ignored
   */
  event result_t deleted(fileresult_t result);
}
```

```
// $Id: FileRename.nc,v 1.1.4.2 2003/08/18 22:09:47 cssharp Exp $

/* tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 */
/**
 * Rename a file
 */
interface FileRename {
  /**
   * Rename a file. If a file called <code>newName</code> exists, it is
   * deleted.
   * @param oldName Name of file to rename. Must not be stack allocated.
   * @param newName New name of file. Must not be stack allocated.
   * @return
   *   SUCCESS: attempt proceeds, <code>renamed</code> will be signaled<br>
   *   FAIL: filesystem is busy or newName is ""
   */
  command result_t rename(const char *oldName, const char *newName);

  /**
   * Signaled at the end of a file rename attempt
   * @param result
   *   FS_OK: file was renamed<br>
   *   FS_ERROR_xxx: rename failure cause
   * @return Ignored
   */
  event result_t renamed(fileresult_t result);
}
```

```
// $Id: FileRead.nc,v 1.1.4.2 2003/08/18 22:09:47 cssharp Exp $

/* tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 */
/**
 * File reading interface, supports sequential reads.
 */

interface FileRead {
  /**
   * open a file for sequential reads.
   * @param filename Name of file to open. Must not be stack allocated.
   * @return
   *   SUCCESS: attempt proceeds, <code>opened</code> will be signaled<br>
   *   FAIL: filesystem is busy or another file is open for reading
   */
  command result_t open(const char *filename);

  /**
   * Signaled at the end of a file open attempt
   * @param result
   *   FS_OK: file was opened<br>
   *   FS_ERROR_xxx: open failure cause
   * @return Ignored
   */
  event result_t opened(fileresult_t result);

  /**
   * Close file currently open for reading
   * @return SUCCESS if a file was open, FAIL otherwise
   */
  command result_t close();
```

```
  /**
   * Read bytes sequentially from open file.
   * @param buffer Target to read into
   * @param n Number of bytes to read
   * @return
   *   SUCCESS: attempt proceeds, <code>readDone</code> will be signaled<br>
   *   FAIL: no file was open for reading, or a read is in progress
   */
  command result_t read(void *buffer, filesize_t n);

  /**
   * Signaled when a <code>read</code> completes
   * @param buffer Buffer that was passed to <code>read</code>
   * @param nRead Number of bytes actually read ,
   *   but result will still be FS_OK)
   * @param result
   *   FS_OK: read was successful (if end-of-file is reached,
   *    <code>nRead</code> will be less than the number of bytes requested)<br>
   *   FS_ERROR_xxx: read failure cause.
   * @return Ignored
   */
  event result_t readDone(void *buffer, filesize_t nRead,
  fileresult_t result);

  /**
   * Return number of bytes remaining in file.
   * @return
   *   SUCCESS: attempt proceeds, <code>remaining</code> will be signaled<br>
   *   FAIL: no file was open for reading, or a read is in progress
   */
  command result_t getRemaining();

  /**
   * Signaled when <code>getRemaining</code> completes
   * @param n Number of bytes remaining in file
   * @param result
   *   FS_OK: operation was successful
   *   FS_ERROR_xxx: read failure cause
   * @return Ignored
   */
  event result_t remaining(filesize_t n, fileresult_t result);
}
```

```
// $Id: FileWrite.nc,v 1.1.4.3 2003/08/20 21:51:48 idgay Exp $

/* tab:4
 * "Copyright (c) 2000-2003 The Regents of the University  of California.
 * All rights reserved.
 *
 * Permission to use, copy, modify, and distribute this software and its
 * documentation for any purpose, without fee, and without written agreement is
 * hereby granted, provided that the above copyright notice, the following
 * two paragraphs and the author appear in all copies of this software.
 *
 * IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR
 * DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT
 * OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF
 * CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 * THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.  THE SOFTWARE PROVIDED HEREUNDER IS
 * ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATION TO
 * PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS."
 *
 * Copyright (c) 2002-2003 Intel Corporation
 * All rights reserved.
 *
 * This file is distributed under the terms in the attached INTEL-LICENSE
 * file. If you do not find these files, copies can be found by writing to
 * Intel Research Berkeley, 2150 Shattuck Avenue, Suite 1300, Berkeley, CA,
 * 94704.  Attention:  Intel License Inquiry.
 */
/**
 * File reading interface, supports appending writes.
 */

interface FileWrite {
  /**
   * open a file for sequential reads.
   * @param filename Name of file to open. Must not be stack allocated.
   * @param flags: open options, an or (|) of FS_Fxxx constants.<br>
   *   <code>FS_FTRUNCATE</code> Truncate file if it exists<br>
   *   <code>FS_FCREATE</code> Create file if it doesn't exist
   * @param truncate TRUE if file should be truncated if it exists
   * @return
   *   SUCCESS: attempt proceeds, <code>opened</code> will be signaled<br>
   *   FAIL: filesystem is busy, another file is already open for writing,
   *     filename is ""
   */
  command result_t open(const char *filename, uint8_t flags);

  /**
   * Signaled at the end of a file open attempt
   * @param fileSize size of file (if file was opened)
   * @param result
   *   FS_OK: file was opened<br>
   *   FS_ERROR_xxx: open failure cause
   * @return Ignored
   */
  event result_t opened(filesize_t fileSize, fileresult_t result);
```

```
/**
 * close file currently open for writing
 * @return
 *   SUCCESS: attempts proceeds, <code>closed</code> will be signaled<br>
 *   FAIL: no file was open for writing, or a close/append/reserve/sync
 *     is in progress
 */
command result_t close();

/**
 * Signaled at the end of a file close. File is closed in all cases,
 *   including failure (but in case of failure some data may have been lost).
 * @param result
 *   FS_OK: file was closed without problems. All data has been comitted to
 *     stable storage.<br>
 *   FS_ERROR_xxx: close failure cause
 * @return Ignored
 */
event result_t closed(fileresult_t result);

/**
 * Write bytes sequentially to end of open file.
 * @param buffer Data to write
 * @param n Number of bytes to write
 * @return
 *   SUCCESS: attempt proceeds, <code>appended</code> will be signaled<br>
 *   FAIL: no file was open for writing, or a close/append/reserve/sync
 *     is in progress
 */
command result_t append(void *buffer, filesize_t n);

/**
 * Signaled when a <code>append</code> completes
 * @param buffer Buffer that was passed to <code>append</code>
 * @param nWritten Number of bytes actually written
 *   but result will still be FS_OK)
 * @param result
 *   FS_OK: write was successful
 *   FS_ERROR_xxx: write failure cause. Some bytes may have been written
 *     (as reported by the value of <code>nWritten</code>
 * @return Ignored
 */
event result_t appended(void *buffer, filesize_t nWritten,
fileresult_t result);

/**
 * Reserve space for the currently open file to be <code>newSize</code>
 * bytes long. <code>append</code>s that do not make the file take
 * more than <code>newSize</code> bytes will not fail with FS_ERROR_NOSPACE.
 * Note: you can find the reserved size of a file by requesting a reserve
 * with a newSize of 0. The <code>reserved</code> event will indicate the
 * space currently reserved.
 * @param newSize Size file is expected to grow to
 * @return
 *   SUCCESS: attempt proceeds, <code>reserved</code> will be signaled<br>
 *   FAIL: no file was open for writing, or a close/append/reserve/sync
 *     is in progress
```

```
   */
  command result_t reserve(filesize_t newSize);

  /**
   * Signaled at the end of a space reservation attempt
   * @param maxSize New reserved size (>= requested size)
   * @param result
   *   FS_OK: space was successfully reserved<br>
   *   FS_ERROR_xxx: failure cause
   * @return Ignored
   */
  event result_t reserved(filesize_t reservedSize, fileresult_t result);

  /**
   * Ensure data appended is comitted to stable storage.
   * @return
   *   SUCCESS: attempt proceeds, <code>synced</code> will be signaled<br>
   *   FAIL: no file was open for writing, or a close/append/reserve/sync
   *     is in progress
   */
  command result_t sync();

  /**
   * Signaled at the end of a sync attempt
   * @param result
   *   FS_OK: sync was successful<br>
   *   FS_ERROR_xxx: failure cause
   * @return Ignored
   */
  event result_t synced(fileresult_t result);
}
```