ATA-FS API Specification

Version 2.22

By
Susmit Pal

Internet Audio Group
Texas Instruments, Inc

March 19, 2015

TABLE OF CONTENTS

1. Scope

This document talks about ATA API revision 2.22. This doc covers API definition for upper level ATA-FS API code. The following points cover the scope of document.

- The code is identical for both 54x & 55x.
- Library name – ATA.l55l, ATA.l55, ATA.l54, ATA.l54f are for 55x large memory model, 55x small memory model, 54x near mode & 54x far mode respectively.
- Header file – ata.h
- Clearcase Tag - \ia_modules\ia_int_branch\LATEST
- Clearcase Label – ATA_2.22_23_SEP_2002
- Version – ATA 2.22 23_SEP_2002

2. Features of this release & points to be noted.

Features :

- Have byte level file read/write/seek support. This feature is minimally tested, use at your own risk. The APIs are documented in sections 5.41-5.44.
- Absolute seek & relative seek support.
- APPEND, OVERWRITE & OVERWRITE+APPEND mode support for file writing.
- Extensively tested & bug fixed version.
- Fix for Mantis ids – 1462,1451,1450,1448,1447,1438,1437,1427,1421,1399,1395,1367,1366,1363,1360,1359,1354,1353,1322,1291,1231,1496,1499.
- Improved support for multiple instances.
- Support for Long Directories.
- ATA_write speed improvements. Now ATA_write cycle counts are one-sixth of what was in ATA-FS2.2.
- New ERROR code ATA_ERROR_DISK_FULL.
- Overall code size reduction of 2kb. Better arrangement of APIs so as to let the application spend memory only for the features it wants.
- LFN Bug fixes & disk size api & some other bug fixes.
- Support for multiple AtaStates on the same physical media.
- Full support for FAT12/16/32.
- Capability to create, delete & rename long filenames.
- Dynamically enable/disable simultaneous multiple file writes (MFW). When you enable/disable multiple file writes, you need to ensure that no file handles are open for writing & currently no writing is going on & all the writeable files are closed. If you are using file handles only for reading, then that doesn't need to change.

  The MFW support includes : -

  - Simultaneous different file writes.
  - Simultaneous read & write on different portions of the same file. Of course, the reading must be from a former part & the writing should be to a latter part of the file.
  - You can seek back on a file handle, already open for writes & write at an older location.
  - No limits on number of MFWs.
- LFN support for FAT32.
- Media Format (Quick/Full) support.
- Better performance than ATA-FS1.3 & ATA-FS2.1.
- Support for both DMA & non-DMA versions of low level media drivers.
- Support for creation of case sensitive short filenames but the application needs to call (ATA_setLongFileName + ATA_createLong) instead of (ATA_setFileName + ATA_create) to have this facility.

Points to be noted :

- Release was tested only with CCS2.1 with cgtools 2.04.
- 55x Large memory model library was tested extensively. Testing on other libraries were minimal.
- ATA-FS code was extensively tested only with MMC in native mode as the low level media.
- ATA-FS code was somewhat tested with NAND & IDE on C55x as low level media.
- ATA_TEST_MMC is supplied as a somewhat real-world testing. The user just needs to run it. After some time it should say all tests passed. The CCS output window will tell that files are getting created, written into, read, verified & deleted. If there is a need to run this testcase with a separate media driver, the ENABLE_MMC preprocessor definition needs to be undefined, the mmc_native.l55l needs to be removed from the project & new media driver library which exports the functions inside AtaState needs to be added.
- ATATEST project is the project customers should use to check the APIs or make his own test environment with. This is documented at the end of this API doc. This project is a readymade project with all the required initializations in place & to run an API you just need to copy-paste the examples from this API doc in the test.c of ATATEST project & just say run. Things will run after that. This test project can be used to verify sample application code.
- All ATA-FS APIs might not return all error codes. Error codes only valid for any API are returned. We do not have a document which states which API can return what all error codes.
- MMC native mode libraries for 55x small & large memory model is supplied with this release only for facilitating testing for the end customer.
- The application has to take care that the file handle passed to ATA-FS apis are okay. The apis do not check the validity of a passed handle to reduce code size.

3. Introduction

The ATA (Advanced Technology Attachment) Media API and Driver is a scalable and thin set of routines optimized for a small memory footprint providing FAT formatted media access. These APIs provide ATA level access to both ATA devices like CF & IDE disks & non-ATA devices like MMC, SD, NAND, MS. A driver interface was added to allow the same API to provide access to all of these various media types with pointers to driver functions.

Now there is a higher-level library (ATA.lib) which will define all the higher level APIs & which will be same for all the sorts of media. Accompanying these, we have a lower level driver library for each of the different media, for example, mmc_native.lib or nand.lib.

For general overview of how a FAT (File Allocation Table) looks like, please consult Appendix C.

4. Data Structures

The data structures used are **AtaFile** & **AtaState**. The definition of these data structures is in header file "Ata.h" which is supplied with this release.

AtaState Structure

```
typedef struct
{
  AtaSector     BootPhySector;
  AtaUint16     FatType;
  AtaUint16     WordsPerLogSector;
  AtaUint16     LogSectorsPerCluster;
  AtaUint16     RootDirEntries;
  AtaUint32     LogSectorsPerFat;
  AtaSector     FirstFatSector;
  AtaSector     RootDirSector;
  AtaSector     FirstDataSector;
  AtaSector     TotalSectors;
  AtaUint16     Data;
  AtaUint16     CurrentWord;
  AtaSector     CurrentPhySector;
  AtaUint16     WordsPerCluster;
  AtaUint16     NumberOfFats;
  AtaUint16     MFWFlag;
  AtaError      Err_Status;
  void          *pAtaMediaState;
  AtaError      (*AtaReset)(void *pAtaMediaState);
  int           (*AtaCommandReadyStatusCheck)(void *pAtaMediaState);
  AtaError      (*AtaBusyStatusCheck)(void *pAtaMediaState);
```

```
  AtaUint16     (*AtaGetSerialID)(void *pAtaMediaState, AtaSerialID *pID);
  AtaError      (*AtaIssueReadCommand)(AtaSector PhySector, void *pAtaMediaState,
AtaUint16 SectorCount);
  AtaError      (*AtaDataReadyStatusCheck)(void *pAtaMediaState);
  AtaError      (*AtaReadNextWord)(void *pAtaMediaState, AtaUint16 *pWord,
AtaUint16 Count);
  AtaError      (*AtaReadNextNWords)(void *pAtaMediaState, AtaUint16 *pWord,
AtaUint16 Count);
  AtaError      (*AtaWriteSector)(AtaSector PhySector, void *pAtaMediaState,
AtaUint16 *pWord, int ByteSwap);
  AtaError      (*AtaWriteSectorFlush)(void *pAtaMediaState);
  AtaError      (*AtaInitAtaMediaState)(void *pAtaState);
  Void          (*AtaErrorCallback)(unsigned short Error);
  AtaUint16 (*get_mod_time)();
  AtaUint16 (*get_mod_date)();
  AtaUint16 (*get_time)();
  AtaUint16 (*get_date)();
  AtaUint16 *_AtaWriteBuffer;
  AtaSector _AtaWriteCurrentPhySector;
} AtaState;
```

This structure consists of some variables and function pointers. The description of this variable and function pointer is below.

The description of variables used in AtaState structure:

- **BootPhySector** – It is physical sector number for Boot sector.
- **FatType** – FAT type (12/16/32)
- **WordsPerLogSector –** Numbers of words, which makes one logical sector.
- **LogSectorsPerCluster** - Numbers of sectors, which makes one logical Cluster.
- **RootDirEntries** - Number of entries in root directory.
- **LogSectorsPerFat –** Number of logical sectors in FAT.
- **FirstFatSector –** The sector number for first FAT entry.
- **RootDirSector –** The sector number for root directory.
- **FirstDataSector –** The sector number for first data.
- **TotalSectors –** Total number of sectors.
- **Data –** Buffer used to data read/write from media
- **CurrentWord –** The current word in a sector.
- **CurrentPhySector –** Current physical sector number.
- **WordsPerCluster –** Number of words per cluster.
- **NumberOfFats –** Number of FATs in media.
- **MFWFlag –** Flag to enable/disable multiple file write capability.
- **Err_Status –** Internal variable to check for error codes.
- **pAtaMediaState –** Pointer to media specific structure.

The descriptions of function pointers are used in AtaState:

- **AtaReset -** Resets the media.
- **AtaGetSerialID –** Gives serial ID of media
- **AtaBusyStatusCheck –** Checks the status of media for sending command. Zero means media is not busy.
- **AtaCommandReadyStatusCheck –** Checks the status of media before sending command. Zero means media is ready to take command.
- **AtaDataReadyStatusCheck –** Checks the status of media for sending/receiving data. Zero means media is ready to take/give data.
- **AtaIssueReadCommand** – Sends the command for reading from media.
- **AtaReadNextWord-** Reads the one word from media.
- **AtaReadNextNWords-** Reads the multiple no. of words (less than or equal to one sector) from media.
- **AtaWriteSectorFlush –** Flushes the cached data in media.
- **AtaWriteSector –** Writes one sector to media.
- **AtaInitAtaMediaState –** Used to intialize media specific values. This is a function which can initialize AtaMediaState variable. It can be used for other purposes also. Like in MMC we use it to point internal variables to user supplied buffer.
- **AtaErrorCallback –** Error Handling routine.
- **get_mod_time –** Function pointer to give user specified forced time.
- **get_mod_date –** Function pointer to give user specified forced date.
- **get_time –** Function pointer to give system time.
- **get_date –** Function pointer to give system date.
  These 4 functions described above need to be implemented by the application.
- **_AtaWriteBuffer –** Buffer allocated by application to be used in write operations. The needed size of this buffer is 256words.
- **_AtaWriteCurrentPhySector –** Internal state variable which maintains the current physical sector of the disk which is getting written to.

The function pointers in the AtaState structures have to be initialized like in the example given below. The example given is for MMC.

```
#include "ata.h"
#include "ata_mmc.h"
int get_time();
int get_date ();
int get_mod_time();
int get_mod_date();
int BufMMC[100];
int BufMMCR[256];
int BufMMCWR[256];
#ifndef DMA_SUPP
AtaMMCState MyMMCState =
```

```
{
                                        BufMMC,
                                          2,
                                        0x4c00,
                                        0x6c00,
                                          0,
                                          2,
                                        65535
};
#else
AtaMMCState MyMMCState =
{
                                        BufMMC,
                                          2,
                                        0x4c00,
                                        0x6c00,
                                          0,
                                          2,
                                    0, //infinite timeout
                                          2,
                                          4,
                                       BufMMCR,
                                      BufMMCWR,
                                      MMC_DMA_callback
};
#endif

void main() {

  /* Declare variables */
  AtaState state;
  AtaError error = ATA_ERROR_NONE;
  AtaState *pstate=&state;
  AtaUint16 _AtaWriteBuffer[256];
  /* Initialize AtaState function pointers */
#ifndef DMA_SUPP
 state.AtaInitAtaMediaState=MMC_initState;
 state.pAtaMediaState=&MyMMCState;
 pstate->AtaInitAtaMediaState(pstate);
#else
 state.AtaInitAtaMediaState=MMC_DMA_initState;
 state.pAtaMediaState=&MyMMCState;
 pstate->AtaInitAtaMediaState(pstate);
#endif
```

```
/*AtaState initializations*/
  state._AtaWriteBuffer = _AtaWriteBuffer;
  state.get_date = get_date;
  state.get_time = get_time;
  state.get_mod_date = get_mod_date;
  state.get_mod_time = get_mod_time;
/*AtaState initializations*/
          CpuClockInit(10,0);
error = ATA_systemInit(&state);
//Use the following line only if you need simultaneous multiple file write support.
//error = ATA_enableMFW(&state);
//go on doing your own stuff
}

/*SUSMIT - This function sets up the CPU clock. In this example it is set to 120Mhz*/
CpuClockInit(int pllmult ,int plldiv)
{
/* force into BYPASS mode (b4=0) */
CLKMD = 0;
/* wait for BYPASS mode to be active */
while (CLKMD & (1<<LOCK));
CLKMD |= ((pllmult<<PLL_MULT)|(plldiv<<PLL_DIV)|(1<<PLL_ENABLE));
/* wait for PLL mode to be active */
while (!(CLKMD & (1<<LOCK)));
}
```

AtaFile Structure

```
typedef struct
{
  char Filename[9];
  char Ext[4];
  AtaUint16 Attributes;
  AtaUint16 Time;
  AtaUint16 Date;
  AtaUint16 reserved1;
  AtaCluster StartCluster;
  AtaFileSize Size;
  AtaCluster Cluster;
  AtaUint16 WordInCluster;
  AtaCluster NextBrokenCluster;
  AtaCluster PrevBrokenCluster;
  AtaSector CurrentDirSector;
  AtaDirEntry CurrentDirEntry;
  AtaCluster CurrentDirSectorTag;
```

```
  AtaCluster StartDirCluster;
  AtaFileSize CurrentByte;
  AtaState
} AtaFile;
```

The following variables used in AtaFile.

- **Filename –** Unpacked, null-terminated character string for filename (00XX, where XX is the ASCII character and last character is 0000).
- **Ext –** Unpacked, null-terminated character string for extension (00XX, where XX is the ASCII character and last character is 0000).
- **Attributes –** File attribute byte in LSB.
- **Time –** Time record from file directory entry.
- **Date –** Date record from file directory entry.
- **StartCluster –** File starting cluster number
- **Size –** File size in bytes
- **Cluster –** Current cluster number
- **WordInCluster –** Current byte offset from start of cluster
- **NextBrokenCluster –** Next cluster where fragment occurs. The assertion is that the file is unfragmented between PrevBrokenCluster and NextBrokenCluster.
- **PrevBrokenCluster –** First cluster where contiguous cluster chain started. The assertion is that the file is unfragmented between PrevBrokenCluster and NextBrokenCluster.
- **CurrentDirSector –** Sector of start of current directory entries
- **CurrentDirEntry –** Entry number in current directory
- **CurrentDirSectorTag –** No. of clusters from start of current directory.
- **StartDirCluster –** Directory starting cluster number.
- **CurrentByte –** Current byte in the file
- **pDrive –** Pointer to drive state structure (AtaState). This is used to provide access to the media state and the media driver functions.

5. ATA API's

5.1. ATA_systemInit

Name

AtaError ATA_systemInit (AtaState *pAtaDrive)

Synopsis

To initialize the media hardware and the software variables needed for navigating the file system.

Input parameters

- AtaState *pAtaState – This parameter is a pointer to a single system drive/media.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  All the elements of the pAtaState structure are modified by this function. They are:

  BootPhySector, FatType, WordsPerLogSector, LogSectorsPerCluster, RootDirEntries, LogSectorsPerFat, FirstFatSector, RootDirSector, FirstDataSector, TotalSectors, CurrentWord, CurrentPhySector, WordsPerCluster, NumberOfFats, MFWFlag, _AtaWriteCurrentPhySector, Err_Status

- **Functional description:**

  The ATA_systemInit function does the following:

  - Initializes the media (CF, MMC, SD etc) by starting from a full reset. It calls AtaReset for this purpose. The AtaReset call gets transformed to the respective media calls such as MMC_reset, which will be in the MMC_native.lib, which in turn calls low level initialization functions for the MMC media.
  - Sets up the operating mode of the media (example NATIVE mode)
  - Initializes ATA-FS state variables including FAT type for the associated file system. Reading the boot sector does this.
  - Verifies the correctness of the received data from the boot sector
  - Performs signature word check.

  Note:

  1. The FAT type determination is based on the total cluster count, and not based on the entry in the master boot sector
  2. ATA_systemInit needs a valid partition table i.e. a valid MBR.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
/*Do media specific intialization*/

  /* Initializes all hardware and software variables of file navigation */
 error = ATA_systemInit (&state);
```

5.2. ATA_getSerialID

Name
AtaUint16 ATA_getSerialID (AtaState *pAtaDrive, AtaSerialID *pID)

Synopsis
Obtains the media ID of a given drive..

Input parameters

- AtaState *pAtaState – This parameter is a pointer to a drive.
- AtaSerialID *pID – pointer to temporary memory storage for media ID

Return value
Media ID length in number of words.

Description

- **State variables modified by function**
  None

- **Functional description:**
  The Ata_getSerialID function obtains the media is of a given drive & stores it in a temporary memory storage of type AtaSerialID a pointer to which is passed as an argument in the function. It is mandatory to call ATA_sytemInit before calling this function. Basically the low-level media specific function of getting serial ID is called

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
AtaState state;
AtaError error;
AtaUint16 len;
AtaSerialID serialID;
/*Do media specific intialization*/

  /* Initialize all hardware and software variables of file navigation */
error = ATA_systemInit (&state);

/* get serial ID */
len =  ATA_getSerialID (&state, serialID);
```

5.3. ATA_fileInit

Name
AtaError ATA_fileInit (AtaState *pAtaDrive, AtaFile *pAtaFile)

Synopsis
Finds the first file in the root directory of a media.

Input parameters

- *pAtaFile – This parameter is a pointer to the AtaFile state structure.

- *pAtaDrive – This parameter is a pointer to the media.

Return value
Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  None

- **Functional description:**
  The ATA_cdRoot function resets the position of the cluster pointer to start cluster of the root directory (zero for FAT12/16), and the current directory sector to the first data sector of the file system. Then the ATA_findFirst function finds the first file in the root directory. If the media is empty then the API returns ATA_ERROR_NONE instead of ATA_ERROR_FILE_NOT_FOUND. To check whether there is any file or not use ATA_findFirst or ATA_findNext.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
This function also assigns a AtaState variable to an AtaFile variable.

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
/*Do media specific intialization*/

    /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit (&state);

  /* Initialize the file structure with first entry in root directory */
```

```
error = ATA_fileInit (&state, &mysong);
```

5.4. ATA_read

Name

AtaError ATA_read (AtaFile *pAtaFile, AtaUint16 *Data, AtaUint16 NumWords)

Synopsis

To read “NumWord” number of words from *pAtaFile, and place it in the buffer *Data.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.
- *Data – This parameter is a pointer to the receiver data buffer
- NumWords – This parameter specifies the number of words to be read from the file.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  WordInCluster, Cluster, NextBrokenCluster, CurrentPhySector, CurrentWord, CurrentByte

- **Functional description:**

  The ATA_read function calculates the location of the current word in the file using information from both AtaState and AtaFile structures. After calculating the current sector number, it issues the read command to the media and uses low level functions to read the required data off the sector. It keeps track of the cluster number using the “Cluster” and “NextBrokenCluster” variables. After reading data from the media, a byte swap is performed on the data to converts it to big endian. On leaving the function, the variables point to the current word that has just been read from the file. It also updates the currentbyte pointer of the file handle.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
We should not call ATA_close after ATA_read. If we do not modify the file, there is no need to close it.

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile  mysong;
  int test;
  AtaUint16 Data[256];
/*Do media specific intialization*/

    /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit (&state);

  /* Initialize the file structure with first entry in root directory */
  error = ATA_fileInit(&state, &mysong);

 /* Test if first entry in root directory is directory or file
  return value 1 for directory , 0 for file */
  test =  ATA_isDir (&mysong);

  if(!test)
  {
   /* read 256 words from first file in root directory and place in Data array. The reading
is done assuming big endian*/
   error =  ATA_read (&mysong, Data, 256);
  }
```

5.5. ATA_readLittleEndian

Name
AtaError ATA_readLittleEndian (AtaFile *pAtaFile, AtaUint16 *Data, AtaUint16 Words)

Synopsis

To read “NumWord” number of words from *pAtaFile, and place it in the buffer *Data in little endian format.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.
- *Data – This parameter is a pointer to the receiver data buffer
- NumWords – This parameter specifies the number of words to be read from the file.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  WordInCluster, Cluster, NextBrokenCluster, CurrentPhySector, CurrentWord, CurrentByte

- **Functional description:**

  The ATA_readLittleEndian function calculates the location of the current word in the file using information from both AtaState and AtaFile structures. After calculating the current sector number, it issues the read command to the media and uses low level functions to read the required data off the sector. It keeps track of the cluster number using the “Cluster” and “NextBrokenCluster” variables. On leaving the function, the variables point to the current word that has just been read from the file. It also updates the currentbyte pointer of the file handle.

Bugs & Limitations

None

References

Refer to appendix C.

Hardware Dependencies

None

Notes

None

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
  int test;
  AtaUint16 Data[256];
/*Do media specific intialization*/

   /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit(&state);

  /* Initialize the file structure with first entry in root directory */
  error = ATA_fileInit(&state, &mysong);

 /* Test if first entry in root directory is directory or file
  return value 1 for directory , 0 for file */
  test =  ATA_isDir(&mysong);

  if(!test)
  {
  /* read 256 words from first file in root directory and place in Data array.
   The reading is done assuming little endian*/
   error =  ATA_readLittleEndian(&mysong, Data, 256);
  }
```

5.6. ATA_seek

Name

AtaError ATA_seek (AtaFile *pAtaFile, AtaFileSize OffsetFromStart)

Synopsis

To move the current word pointer of the file *pAtaFile, to “OffsetFromStart” words from the start of the file.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to seek.
- OffsetFromStart – This parameter specifies the offset in number of words from the start of the file

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization

For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  WordInCluster, Cluster, CurrentByte

- **Functional description:**
  The ATA_seek function resets the position of the cluster pointers to the value specified in the OffsetFromStart value of the function call. It does not read the words in between the offset and the start. It also updates the currentbyte pointer of the file handle.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;
 int test;
 AtaUint32 offset=256;
 /*Do media specific intialization*/

  /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit (&state);

 /* Initialize the file structure with first entry in root drectory */
 error = ATA_fileInit (&state, &mysong);

/* test if first entry in root directory is directory or file
 return value 1 for directory , 0 for file */
 test =  ATA_isDir (&mysong);
```

```
if(!test)
{
/* move by offset from start in first file in root directory */
error = ATA_seek (&mysong, offset);
}
```

5.7. ATA_seek_relative

Name

AtaError ATA_seek_relative (AtaFile *pAtaFile, AtaFileSize OffsetFromStart)

Synopsis

To move the current word pointer of the file *pAtaFile, to "OffsetFromStart" words from the start of the file.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to seek.
- OffsetFromStart – This parameter specifies the offset in number of words from the current position of the file

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  WordInCluster, Cluster, CurrentByte

- **Functional description:**

  The ATA_seek_relative function does a relative seek on the file i.e. seeks from the current location of the file. It modifies the position of the cluster pointers to the value specified in the OffsetFromStart value of the function call. It does not read the words in between the offset and the current location. It also updates the currentbyte pointer of the file handle.

Bugs & Limitations

None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;
 int test;
 AtaUint32 offset=256;
 /*Do media specific intialization*/

  /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit (&state);

 /* Initialize the file structure with first entry in root drectory */
 error = ATA_fileInit (&state, &mysong);

/* test if first entry in root directory is directory or file
 return value 1 for directory , 0 for file */
 test =  ATA_isDir (&mysong);

 if(!test)
 {
 error = ATA_seek (&mysong, 10);
 /* move by offset from current location in first file in root directory */
 error = ATA_seek_relative (&mysong, offset);
 }
```

5.8. ATA_sleep

Name
AtaError ATA_sleep (AtaState *pAtaDrive)

Synopsis
To shut down a particular media.

Input parameters

- *pAtaState – This parameter is a pointer to the media to be shut down.

Return value
Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  None

- **Functional description:**
  Flushes the current sector from the ATA device & puts it in idle mode.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;
/*Do media specific intialization*/

  /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit(&state);

  /* Do some processing */

 /* Flash the current sector from media */
  error = ATA_sleep(&state);
```

5.9. ATA_cdRoot

Name

AtaError ATA_cdRoot (AtaFile *pAtaFile)

Synopsis

Low-level function that initializes AtaFile structure variables to the root directory

Input parameters

- *pAtaFile – This parameter is a pointer to the AtaFile state structure.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  pAtaFile->CurrentDirSector
  pAtaFile->StartDirCluster
  pAtaFile->CurrentDirSectorTag
  pAtaFile->CurrentDirEntry

- **Functional description:**

  The ATA_cd function changes the current directory sector of the file *pAtaFile to point to the required new value by using the StartCluster and the logical sectors per cluster variables. For FAT32 media, this also gets the start cluster of the root directory.

Bugs & Limitations

References

Refer to appendix C.

Hardware Dependencies

None

Notes

None

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
  int test;
/*Do media specific intialization*/

   /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit (&state);

  /* Initialize the file structure with first entry in root drectory */
  error = ATA_fileInit (&state, &mysong);

 /* test if first entry in root directory is directory or file
  return value 1 for directory , 0 for file */
  test =  ATA_isDir (&mysong);

   while(!test)
   {
    /* find next file in root directory */
    error =  ATA_findNext (&mysong);

   /* test if first entry in root directory is directory or file
    return value 1 for directory , 0 for file */
    test =  ATA_isDir (&mysong);

   }

   /* change to first directory in root directory */
   test =  ATA_cd (&mysong);

   /* change to root directory */
   error = ATA_cdRoot(&mysong);
```

5.10. ATA_cd

Name

AtaError ATA_cd (AtaFile *pAtaFile)

Synopsis

Used to change directories

Input parameters

- *pAtaFile – This parameter is a pointer to the state structure of the current file or directory.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  pAtaFile->CurrentDirSectorTag
  pAtaFile->StartDirCluster

- **Functional description:**

  The ATA_cd function changes the current directory sector of the file *pAtaFile to point to the required new value by using the StartCluster and the logical sectors per cluster variables.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
  int test;
/*Do media specific intialization*/

   /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit (&state);
```

```
 /* Initialize the file structure with first entry in root drectory */
 error = ATA_fileInit (&state, &mysong);

/* test if first entry in root directory is directory or file
 return value 1 for directory , 0 for file */
 test =  ATA_isDir (&mysong);

  while(!test)
  {
   /* Find next file in root directory */
   error =  ATA_findNext (&mysong);

  /* Test if first entry in root directory is directory or file
   return value 1 for directory , 0 for file */
   test =  ATA_isDir (&mysong);

  }

  /* Change to first directory in root directory */
 error  =  ATA_cd (&mysong);
```

5.11. ATA_findFirst

Name

AtaError ATA_findFirst (AtaFile *pAtaFile)

Synopsis

To find first file in a specific directory of the media.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  pAtaFile->CurrentDirEntry

- **Functional description:**
  After setting the pAtaFile->CurrentDirEntry variable, this function finds the first entry in a specific directory.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
  int test;
  AtaUint32 offset=256;
/*Do media specific intialization*/

    /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit(&state);

  error = ATA_fileInit (&state, &mysong);

  /* Do something else ….*/
  /* Change to root directory */
  error = ATA_cdRoot(&mysong);

  /* Find the first file entry in root directory and fill mysong variable */
  error = ATA_findFirst(&mysong);
```

5.12. ATA_findNext

Name
AtaError ATA_findNext (AtaFile *pAtaFile)

Synopsis
To find the next file in a specific directory

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.

Return value
Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  pAtaFile->CurrentDirEntry

- **Functional description:**
  After incrementing the pAtaFile->CurrentDirEntry variable, this function finds the next entry in a specific directory.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
/* Declare variables */
AtaState state;
AtaError error;
```

```
  AtaFile mysong;
/*Do media specific intialization*/

   /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit(&state);

  /* Initialize the file structure with first entry in root drectory */
  error = ATA_fileInit(&state, &mysong);

  /* Find next file in root directory */
  error =  ATA_findNext(&mysong);
```

5.13. ATA_isDir

Name

int ATA_isDir(AtaFile *pAtaFile)

Synopsis

To determine whether the current file is a directory

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.

Return value

If the file is a directory a 1 is returned otherwise a 0 is returned.

Description

- **State variables modified by function**
  None
- **Functional description:**
  This function ands the pAtaFile->Attributes with ATA_ATTR_DIRECTORY (0x10) to determine whether the current file is a directory. The result of the AND operation sets the return value of the function.

Bugs & Limitations

None

References

Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;
 int test;
/*Do media specific intialization*/

  /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit(&state);

 /* Initialize the file structure with first entry in root drectory */
 error = ATA_fileInit(&state, &mysong);

 /* Test if first entry in root directory is directory or file
 return value 1 for directory , 0 for file */
 test =  ATA_isDir(&mysong);
```

5.14. ATA_getLongName

Name
AtaError ATA_getLongName (AtaFile *pAtaFile, char *LongName, unsigned int StartOffset, unsigned int Length)

Synopsis
To return the full length name of a file/directory.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.
- *LongName –The variable, which holds the long name.
- StartOffset – Offset in actual long filename from where we want to read into the Longname array.
- Length – Maximum length to be read from the offset.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  None
- **Functional description:**
  Gets the long file name of a file/directory.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile  mysong;
  char longfilename[256];
/*Do media specific intialization*/

    /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit(&state);

  /* Initialize the file structure with first entry in root drectory */
  error = ATA_fileInit(&state, &mysong);

  /* Read Long file name from first character upto 20 characters
     returns 0 for successful execution     */
 error =  ATA_getLongName(&mysong, longfilename , 0, 20);
```

5.15. ATA_tell

Name

AtaError ATA_tell (AtaFile *pAtaFile, AtaFileSize *pOffsetInFile)

Synopsis

Used to find out the current offset of the read pointer from the start of the file. Returns the offset in words.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.
- pOffsetInFile – Offset value of file read pointer.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  None
- **Functional description:**
  This function calculates the offset of the file pointer from the start of the file.

Bugs & Limitations

None

References

Refer to appendix C.

Hardware Dependencies

None

Notes

None

Example

```
/* Declare variables */
AtaState state;
```

```
  AtaError error;
  AtaFile mysong;
  AtaFileSize offset;
/*Do media specific intialization*/

   /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit(&state);

  /* Initialize the file structure with first entry in root drectory */
  error = ATA_fileInit(&state, &mysong);

  /* Do some read and write */

  /* Get file pointer offset from start in offset */
  error = ATA_tell(&mysong, &offset);
```

5.16. ATA_create

Name

AtaError ATA_create (AtaFile *pAtaFile)

Synopsis

To create and open a new file on the media

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be created.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  CurrentDirEntry, PrevBrokenCluster, NextBrokenCluster, WordInCluster.

- **Functional description:**
  Creates or renames a file & then updates the directory entry. Need to close the file before removing the media. So it is always necessary to call ATA_close after calling ATA_create.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
ATA_create doesn't check for validity of file handle passed.

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;
 AtaFileSize offset;
 /*Do media specific intialization*/


   /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit(&state);

 /* Initialize the file structure with first entry in root drectory */
 error = ATA_fileInit(&state, &mysong);

/* Go to root directory for creating file. First go to directory in which you want to
  create file */
   error = ATA_cdRoot(&mysong);

ATA_setFileName(&mysong, "test", "txt");

 /* Create a file, returns 0 for successful creation*/
 error = ATA_create(&mysong);
 error = ATA_close(&mysong);
```

5.17. ATA_createDirectory

Name
AtaError ATA_createDirectory (AtaFile *pAtaFile)

Synopsis

To create a new directory on the media.

Input parameters

- *pAtaFile – This parameter is a pointer to the directory to be created.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  CurrentDirEntry, PrevBrokenCluster, NextBrokenCluster, WordInCluster.

- **Functional description:**
  Creates directories & subdirectories with help of ATA_create function. Also creates the "." & ".." entries inside a subdirectory.

Bugs & Limitations

None

References

Refer to appendix C.

Hardware Dependencies

None

Notes

ATA_createDirectory doesn't check for validity of file handle passed.

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
/*Do media specific intialization*/

    /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit(&state);

  /* Initialize the file structure with first entry in root drectory */
```

```
error = ATA_fileInit(&state, &mysong);

/* Go to root directory for creating file. First go to directory in which you want to
  create file */
   error = ATA_cdRoot(&mysong);

  error = ATA_setDirectoryName(&mysong,"test");

 /* Create a directory, returns 0 for successful creation*/
 error = ATA_createDirectory(&mysong);
```

5.18. ATA_delete

Name
AtaError ATA_delete (AtaFile *pAtaFile)

Synopsis
To delete a file/directory from the media.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be deleted.

Return value
Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  pAtaFile->PrevBrokenCluster, pAtaFile->NextBrokenCluster, pAtaFile->Cluster
- **Functional description:**
  Deletes a file/directory on the media.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
ATA_delete doesn't check for validity of file handle passed.

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;
 /*Do media specific intialization*/

   /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit(&state);

 /* Initialize the file structure with first entry in root drectory */
 error = ATA_fileInit(&state, &mysong);

/* Go to root directory for creating file. First go to directory in which you want to
  create file */
   error = ATA_cdRoot(&mysong);

 /* to delete file */
error = ATA_findFirst(&mysong);

 /* Delete first file in directory */
 error = ATA_delete(&mysong);
```

5.19. ATA_rename

Name
AtaError ATA_rename (AtaFile *pAtaFile)

Synopsis
To rename a file/directory on the media.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be renamed.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  None
- **Functional description:**
  Renames a file/Directory on the media using ATA_create function.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
ATA_rename doesn't check for validity of file handle passed.

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
/*Do media specific intialization*/

    /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit(&state);

  /* Initialize the file structure with first entry in root drectory */
  error = ATA_fileInit(&state, &mysong);

 /* Go to root directory for creating file. First go to directory in which you want to
   create file */
    error = ATA_cdRoot(&mysong);

   /* Rename first file to test.txt */
  ATA_setFileName(&mysong,"test", “txt”);
```

```
/* Renames first file in directory to test.txt*/
error = ATA_rename(&mysong);
```

5.20. ATA_write

Name

AtaError ATA_write (AtaFile *pAtaFile, AtaUint16 *Data, AtaUint16 Words)

Synopsis

Writes “Words” no. of words from the array pointed by Data to a file.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.
- *Data – pointer to the data to be written to the media.
- Words – number of words to be written.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  pAtaFile->Size, pAtaFile->WordInCluster, pAtaFile->Cluster, pAtaFile->CurrentByte

- **Functional description:**

  Writes “Words” no. of words from the array pointed by Data to a file. It also updates the cluster information & updates the fat with the size. There is no need to flush after this because it calls flush internally. We need to call ATA_close once the writing is complete before removing the media.

Bugs & Limitations

None

References

Refer to appendix C.

Hardware Dependencies
None

Notes
ATA_write now supports three writing modes (OVERWRITE, APPEND & OVERWRITE+APPEND) internally. The user need not worry about these. But when writing is finished in any of these three modes, ATA_close needs to be called.

ATA_write doesn't check for validity of file handle passed.

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
   unsigned int Data[256] = {0xbeef};
/*Do media specific intialization*/

    /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit(&state);

  /* Initialize the file structure with first entry in root drectory */
  error = ATA_fileInit(&state, &mysong);

 /* Go to root directory for creating file. First go to directory in which you want to
   create file */
    error = ATA_cdRoot(&mysong);

  ATA_setFileName(&mysong,"test", “txt”);

  /* Create a file , returns 0 for successful creation*/
  error = ATA_create(&mysong);
  errot = ATA_close(&mysong);
  /*Find the file again*/
  /* Write 256 word of data in test.txt, returns 0 on successful writing */
  error =  ATA_write(&mysong, Data, 256);
  error = ATA_close(&mysong);
```

5.21. ATA_close

Name
AtaError ATA_close (AtaFile *pAtaFile)

Synopsis
Closes open file after updating the fat.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.

Return value
Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  pAtaFile->PrevBrokenCluster, pAtaFile->WordInCluster

- **Functional description:**
  Updates file size in directory entry, replaces directory entry sector & closes the file.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
ATA_close doesn't check for validity of file handle passed.
After doing a ATA_close, file handle might become invalid, please do not use this handle at a later point of time.

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;
/*Do media specific intialization*/

  /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit (&state);
```

```
/* Initialize the file structure with first entry in root drectory */
error = ATA_fileInit(&state, &mysong);

/* Go to root directory for creating file. First go to directory in which you want to
 create file */
  error = ATA_cdRoot(&mysong);

ATA_setFileName(&mysong,"test", “txt”);

/* Create a file, returns 0 for successful creation*/
error = ATA_create(&mysong);

/* Close the file, returns 0 for successful deletion, update the FAT entry */
error =  ATA_close(&mysong);
```

5.22. ATA_setFileName

Name

AtaError ATA_setFileName(AtaFile *pAtaFile, char *name, char *ext);

Synopsis

Lets the user to define a filename & extension which he can create using ATA_create or rename using ATA_rename.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.
- name – Character array of 9 which contains the name.
- ext – Character array of 4 which contains the extension.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  pAtaFile->Filename, pAtaFile->Ext, pAtaFile->Attributes, pAtaFile->Time, pAtaFile->Date.

- **Functional description:**
  Puts the Filename & Extension specified in the required AtaFile structure

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
Application has to define following two functions, which gives time and data for file to be created for writing.

- int get_time()
  {
  return TIME;
  }

- int get_date ()
  {
  return DATE;
  }

(ref. Appendix E)

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;
 char name[9]={“TEST”};
 char ext[4]={“txt”};
/*Do media specific intialization*/

   /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit (&state);

 /* Initialize the file structure with first entry in root drectory */
 error = ATA_fileInit(&state, &mysong);

/* Go to root directory for creating file. First go to directory in which you want to
  create file */
```

```
error = ATA_cdRoot(&mysong);
ATA_setFileName(&mysong, name, ext);
```

5.23. ATA_setDirectoryName

Name

AtaError ATA_setDirectoryName(AtaFile *pAtaFile, char *dirname)

Synopsis

Lets the user to define a directoryname which he can create using ATA_createDirectory.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.
- dirname – Character array of 9 which contains the name.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  pAtaFile->Filename, pAtaFile->Ext, pAtaFile->Attributes, pAtaFile->Time, pAtaFile->Date.

- **Functional description:**

  Puts the Directory name specified in the required AtaFile structure

Bugs & Limitations

None

References

Refer to appendix C.

Hardware Dependencies

None

Notes
Application has to define following two functions, which gives time and data for directory to be created.

- int get_time()
  {
  return TIME;
  }

- int get_date ()
  {
  return DATE;
  }

(ref. Appendix E)

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;
 Char dirname[9]={TESTA};
/*Do media specific intialization*/

   /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit (&state);

 /* Initialize the file structure with first entry in root drectory */
 error = ATA_fileInit(&state, &mysong);

/* Go to root directory for creating file. First go to directory in which you want to
  create file */
   error = ATA_cdRoot(&mysong);
  ATA_setDirectoryname(&mysong, dirname);
```

5.24. ATA_diskSize

Name
AtaUint32 ATA_diskSize(AtaState *pAtaState);

Synopsis
Returns the size of the media in KB.

Input parameters

- *pAtaState – This parameter is a pointer to the media.

Return value
AtaUint32
Disk size in Kb.

Description

- **State variables modified by function**
  None

- **Functional description:**
  Calculates size of the whole media.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
  AtaUint32 size;
/*Do media specific intialization*/

    /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit (&state);

  size = ATA_diskSize(&state);
```

5.25. ATA_diskUsed

Name
AtaUint32 ATA_diskUsed(AtaState *pAtaState, AtaError *ret_stat);

Synopsis
Returns the space used of the media in KB.

Input parameters

- *pAtaState – This parameter is a pointer to the media.
- *ret_stat – Returns the error code, if any to the application.

Return value
AtaUint32
Media space used in Kb.

Description

- **State variables modified by function**
  None

- **Functional description:**
  Calculates space used of the whole media. Error code will be passed in ret_stat.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;
 AtaUint32 size;
/*Do media specific intialization*/
```

```
  /* Initialize all hardware and software variables of file navigation */
error = ATA_systemInit (&state);

size = ATA_diskUsed(&state, &error);
```

5.26. ATA_diskFree

Name
AtaUint32 ATA_diskFree(AtaState *pAtaState, AtaError *ret_stat);

Synopsis
Returns the free space of the media in KB.

Input parameters

- *pAtaState – This parameter is a pointer to the media.
- *ret_stat – Returns the error code, if any to the application.

Return value
AtaUint32
Media free space in Kb.

Description

- **State variables modified by function**
  None

- **Functional description:**
  Calculates free space of the whole media. Error code is passed through ret_stat.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
  AtaUint32 size;
/*Do media specific intialization*/

   /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit (&state);

  size = ATA_diskFree(&state, &error);
```

5.27. ATA_format

Name

AtaError ATA_format(AtaState *pDrive, int flag);

Synopsis

Formats the media.

Input parameters

- *pAtaState – This parameter is a pointer to the media.
- Flag =1 means full format, zero means quick format.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  All

- **Functional description:**
  Formats the whole media. It is necessary to do a ATA_systemInit after the ATA_format call to reinitialize the state variables of the AtaState structure.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
This API needs a big stack size. Stack requirement is about 270W.

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;
 AtaUint32 size;

   /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit (&state);
  /*format media*/
 error = ATA_format(&state);
   /* Reinitialize all hardware and software variables of file navigation */
 error = ATA_systemInit (&state);
```

5.28. ATA_setDateTime

Name
AtaError ATA_setDateTime(AtaFile *pAtaFile);

Synopsis
Forcefully sets the date & time for a particular file or directory.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.

Return value
Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  pAtaFile->Time & pAtaFile->Date.

- **Functional description:**
  Sets the date & timestamp for a particular file or directory. This is done by getting the forced date & time from two user implemented functions get_mod_date() & get_mod_time().

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;

   /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit (&state);

 /* Initialize the file structure with first entry in root drectory */
 error = ATA_fileInit(&state, &mysong);

/*Change the timestamp of the first file in the Root Directory*/
   error = ATA_setDateTime (&mysong);
```

5.29. ATA_setAttr

Name

AtaError ATA_setAttr(AtaFile *pAtaFile, AtaUint16 Attr);

Synopsis

Sets the user specified absolute attribute of a particular file or directory.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.
- Attr – User specified attribute which will be set to the file.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  pAtaFile->Attributes.

- **Functional description:**

  Sets the attribute for a particular file or directory. For supported attribute list please consult Appendix B. These attributes can be OR-ed together & passed to this API to set all the attributes at the same time.

Bugs & Limitations

None

References

Refer to appendix C.

Hardware Dependencies

None

Notes

This function sets attribute in absolute mode, so older attributes won't be retained. So, if somebody has a read-only file & wants to hide it also, this function should be used with a parameter of ATA_ATTR_READ_ONLY|ATA_ATTR_HIDDEN.

Example

```
/* Declare variables */
AtaState state;
AtaError error;
AtaFile mysong;

  /* Initialize all hardware and software variables of file navigation */
```

```
error = ATA_systemInit (&state);

/* Initialize the file structure with first entry in root drectory */
error = ATA_fileInit(&state, &mysong);

/*Change the attribute of the first file in the Root Directory*/
  error =ATA_setAttr(&mysong, ATA_ATTR_READ_ONLY|ATA_ATTR_HIDDEN);
```

5.30. ATA_vol

Name

AtaError ATA_vol(AtaState *pAtaDrive, char *volname);

Synopsis

Gets the volume label of the current media.

Input parameters

- *pAtaState – This parameter is a pointer to the media.
- volname – Character array pointer to get the volume label of a drive.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  None.

- **Functional description:**
  Gets the volume label of the current media pointed to by pAtaDrive. This is done by reading off the directory entry having volume label attribute from the disk.

Bugs & Limitations

None

References

Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example
```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
 Char volname[11];

    /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit (&state);

/*get the volume label*/
    error =ATA_vol(&state, volname);
```

5.31. ATA_label

Name
AtaError ATA_label(AtaState *pAtaDrive, char *volname);

Synopsis
Sets the volume label of the current media.

Input parameters
- *pAtaState – This parameter is a pointer to the media.
- volname – Character array pointer to pass the volume label of a drive.

Return value
Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  None.

- **Functional description:**
  Sets the volume label of the current media pointed to by pAtaDrive. This is done by writing into the directory entry having volume label attribute of the disk.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
 Char volname[11]={SUSMIT};

    /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit (&state);

/*set the volume label*/
    error =ATA_label(&state, volname);
```

5.32. ATA_enableMFW

Name
void ATA_enableMFW(AtaState *pAtaState);

Synopsis
Dynamically enables simultaneous multiple file support for the media pointed by AtaState.

Input parameters

- *pAtaState – This parameter is a pointer to the media.

Return value

Return type – Void.

Description

- **State variables modified by function**
  pAtaState->MFWFlag

- **Functional description:**
  Dynamically enables simultaneous multiple file support for the media pointed by AtaState. This actually sets a flag named MFWFlag inside AtaState structure which ATA_write uses.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
When dynamically enabling MFW, all open file handles for write need to be closed.

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;

   /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit (&state);

/*Enables MFW*/
   ATA_enableMFW(&state);
```

5.33. ATA_readSector

Name

AtaError ATA_readSector(AtaSector PhySector, AtaState* pAtaState, AtaUint16 *Word, int ByteSwap);

Synopsis

Reads a sector specified by PhySector from the media pointed by pAtaState & puts it in a buffer pointed by Word.

Input parameters

- PhySector – Sector number to read.
- *pAtaState – This parameter is a pointer to the media.
- *Word – Buffer pointer to keep the data read from the sector.
- ByetSwap – If set, byte swaps the data.

Return value

Return type – AtaError
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  None

- **Functional description:**

  Reads 256 words from the sector supplied by the PhySector parameter from the media pointed to by pAtaState & puts the data in array pointed by Word.

Bugs & Limitations

None

References

Refer to appendix C.

Hardware Dependencies

None

Notes

None

Example

```
/* Declare variables */
AtaState state;
AtaState *pstate = &state;
AtaError error;
AtaFile mysong;
int buffer_to_read[256];
```

```
   /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit (&state);

/*Reads the Root Directory Sector*/
 error = ATA_readSector(pstate->RootDirSector, pstate, buffer_to_read, 0);
```

5.34. ATA_writeSector

Name

AtaError ATA_writeSector(AtaSector PhySector, AtaState* pAtaState, AtaUint16 *Word, int ByteSwap);

Synopsis

Writes 256 words to a sector specified by PhySector to the media pointed by pAtaState from a buffer pointed by Word.

Input parameters

- PhySector – Sector number to write to.
- *pAtaState – This parameter is a pointer to the media.
- *Word – Buffer pointer to keep the data to write to the sector.
- ByetSwap – If set, byte swaps the data.

Return value

Return type – AtaError
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  None

- **Functional description:**

  Writes 256 words to the sector supplied by the PhySector parameter of the media pointed to by pAtaState from the data in array pointed by Word.

Bugs & Limitations

None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
 /* Declare variables */
 AtaState state;
 AtaState *pstate = &state;
 AtaError error;
 AtaFile mysong;
 int buffer_to_write[256] = {0xBeef};

   /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit (&state);

/*Writes to the Sector Number 200*/
 error = ATA_writeSector(200, pstate, buffer_to_write, 0);
```

5.35. ATA_setLongFileName

Name
AtaError ATA_setLongFileName(AtaFile *pAtaFile, char *LongName);

Synopsis
Lets the user to define a long filename which he can create using ATA_createLong.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.
- LongName – Character array of upto 256 chars which contains the LFN.

Return value
Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  pAtaFile->Filename, pAtaFile->Ext, pAtaFile->Attributes, pAtaFile->Time, pAtaFile->Date.

- **Functional description:**

  Generates a 8.3 filename from the specified long filename & puts the filename & extension in the required AtaFile structure

Bugs & Limitations

Can have only 10 LFNs in the same directory whose first 6 alphanumeric characters are same.

References

Refer to appendix C.

Hardware Dependencies

None

Notes

The application has to take care of validity of long filename, both in terms of length & characters in LFN. About length, application has to take note of the depth at which the file is going to reside & decide on length of LFN.

Application has to define following two functions, which gives time and data for file to be created for writing.

- int get_time()
  {
  return TIME;
  }

- int get_date ()
  {
  return DATE;
  }

(ref. Appendix E)

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;
/*Do media specific intialization*/
```

```
  /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit (&state);

 /* Initialize the file structure with first entry in root drectory */
 error = ATA_fileInit(&state, &mysong);

/* Go to root directory for creating file. First go to directory in which you want to
  create file */
   error = ATA_cdRoot(&mysong);
 error = ATA_setLongFileName(&mysong, "Chura Lia Hai Tumne Jo Dilko.mp3");
```

5.36. ATA_setLongDirectoryName

Name

AtaError ATA_setLongDirectoryName(AtaFile *pAtaFile, char *LongName);

Synopsis

Lets the user to define a long directory name which he can create using ATA_createDirectoryLong.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.
- LongName – Character array of upto 256 chars which contains the LFN.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  pAtaFile->Filename, pAtaFile->Ext, pAtaFile->Attributes, pAtaFile->Time, pAtaFile->Date.

- **Functional description:**

  Generates a 8.3 directory name from the specified long filename & puts the directory name in the required AtaFile structure

Bugs & Limitations
Can have only 10 LFNs in the same directory whose first 6 alphanumeric characters are same.

References
Refer to appendix C.

Hardware Dependencies
None

Notes
The application has to take care of validity of long filename, both in terms of length & characters in LFN. About length, application has to take note of the depth at which the directory is going to reside & decide on length of LFN.
Application has to define following two functions, which gives time and data for file to be created for writing.

- int get_time()
  {
  return TIME;
  }

- int get_date ()
  {
  return DATE;
  }

(ref. Appendix E)

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;
/*Do media specific intialization*/

   /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit (&state);

 /* Initialize the file structure with first entry in root drectory */
 error = ATA_fileInit(&state, &mysong);

/* Go to root directory for creating file. First go to directory in which you want to
  create file */
   error = ATA_cdRoot(&mysong);
```

```
 error = ATA_setLongDirectoryName(&mysong, "Chura Lia Hai Tumne Jo
Dilko.mp3");
```

5.37. ATA_createLong

Name

AtaError ATA_createLong(AtaFile *pAtaFile, char *longname);

Synopsis

To create and open a new file on the media. Also create LFN entry for the file with the supplied long filename.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be created.
- *longname – Long Filename to be created. This can be upto 256 chars.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  CurrentDirEntry, PrevBrokenCluster, NextBrokenCluster, WordInCluster.

- **Functional description:**

  Creates an 8.3 file & then updates the directory entry. The 8.3 Filename is generated from the supplied LFN. Also, creates the needed long filename entries according to the long filename supplied through LongName. Need to close the file before removing the media. So it is always necessary to call ATA_close after calling ATA_createLong.

Bugs & Limitations

None

References

Refer to appendix C.

Hardware Dependencies

None

Notes
None

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
/*Do media specific intialization*/

    /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit (&state);

  /* Initialize the file structure with first entry in root drectory */
  error = ATA_fileInit(&state, &mysong);

 /* Go to root directory for creating file. First go to directory in which you want to
   create file */
    error = ATA_cdRoot(&mysong);
  error = ATA_setLongFileName(&mysong, "Chura Lia Hai Tumne Jo Dilko.mp3");

  /* Create a file, returns 0 for successful creation*/
  error = ATA_createLong(&mysong, "Chura Lia Hai Tumne Jo Dilko.mp3");
  error = ATA_close(&mysong);
```

5.38. ATA_createDirectoryLong

Name

AtaError ATA_createDirectoryLong(AtaFile *pAtaFile, char *longname);

Synopsis

To create and open a new long directory on the media. Also create LFN entry for the directory with the supplied long filename.

Input parameters

- *pAtaFile – This parameter is a pointer to the directory to be created.
- *longname – Long directory name to be created. This can be upto 256 chars.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization

For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  CurrentDirEntry, PrevBrokenCluster, NextBrokenCluster, WordInCluster.

- **Functional description:**
  Creates an 8.3 directory & then updates the directory entry. The 8.3 directory name is generated from the supplied LFN. Also, creates the needed long filename entries according to the long filename supplied through LongName.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
/*Do media specific intialization*/

    /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit (&state);

  /* Initialize the file structure with first entry in root drectory */
  error = ATA_fileInit(&state, &mysong);

 /* Go to root directory for creating file. First go to directory in which you want to
   create file */
    error = ATA_cdRoot(&mysong);
  error = ATA_setLongDirectoryName(&mysong, " LONGDIRTEST ONGOING ");

  /* Create a file, returns 0 for successful creation*/
```

```
error = ATA_createDirectoryLong(&mysong, "LONGDIRTEST ONGOING");
```

5.39. ATA_deleteLong

Name

AtaError ATA_deleteLong(AtaFile *pAtaFile)

Synopsis

To delete a file/directory with a long filename from the media.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be deleted.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  pAtaFile->PrevBrokenCluster, pAtaFile->NextBrokenCluster, pAtaFile->Cluster

- **Functional description:**
  Deletes a long file/directory on the media. Also deletes the corresponding long filename entries. This function should be used when we need to delete a long filename from the media. Using ATA_delete on a long filename will only delete the 8.3 entry of the filename, but not the long filename entries.

Bugs & Limitations

None

References

Refer to appendix C.

Hardware Dependencies

None

Notes

None

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
/*Do media specific intialization*/

    /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit (&state);

  /* Initialize the file structure with first entry in root drectory */
  error = ATA_fileInit(&state, &mysong);

 /* Go to root directory for creating file. First go to directory in which you want to
   create file */
    error = ATA_cdRoot(&mysong);
  error = ATA_setLongFileName(&mysong, "Chura Lia Hai Tumne Jo Dilko.mp3");

  /* Create a file, returns 0 for successful creation*/
  error = ATA_createLong(&mysong, "Chura Lia Hai Tumne Jo Dilko.mp3");
  error = ATA_close(&mysong);
/*Find the file again*/
  error = ATA_deleteLong(&mysong); /*delete the newly created LFN*/
```

5.40. ATA_renameLong

Name

AtaError ATA_renameLong(AtaFile *pAtaFile, char *longnamenew);

Synopsis

To rename a long file/directory on the media.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be renamed.
- longnamenew – New LFN.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  pAtaFile->Filename, pAtaFile->Ext, pAtaFile->Attributes, pAtaFile->Time, pAtaFile->Date.

- **Functional description:**

  Renames a long file/directory on the media using ATA_createLong function. Computes & writes new 8.3 entry for the new LFN. Also deletes entries for old LFN & creates entries for new LFN.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
/*Do media specific intialization*/

    /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit (&state);

  /* Initialize the file structure with first entry in root drectory */
  error = ATA_fileInit(&state, &mysong);

 /* Go to root directory for creating file. First go to directory in which you want to
   create file */
    error = ATA_cdRoot(&mysong);
  error = ATA_setLongFileName(&mysong, "Chura Lia Hai Tumne Jo Dilko.mp3");

  /* Create a file, returns 0 for successful creation*/
  error = ATA_createLong(&mysong, "Chura Lia Hai Tumne Jo Dilko.mp3");
  error = ATA_close(&mysong);
```

```
/*Find the file again*/
error = ATA_renameLong(&mysong, "Khoya khoya Chand.mp3");
```

5.41. ATA_read_b

Name

AtaError ATA_read_b (AtaFile *pAtaFile, AtaUint16 *Data, AtaUint16 bytes)

Synopsis

To read “bytes” number of bytes from *pAtaFile, and place it in the buffer *Data.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.
- *Data – This parameter is a pointer to the receiver data buffer
- bytes – This parameter specifies the number of bytes to be read from the file.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  WordInCluster, Cluster, NextBrokenCluster, CurrentPhySector, CurrentWord, CurrentByte

- **Functional description:**

  The ATA_read_b function calculates the location of the current byte in the file using information from both AtaState and AtaFile structures. After calculating the current sector number, it issues the read command to the media and uses low level functions to read the required data off the sector. It keeps track of the cluster number using the “Cluster” and “NextBrokenCluster” variables. After reading data from the media, a byte swap is performed on the data to converts it to big endian. On leaving the function, the variables point to the current word that has just been read from the file. It also updates the currentbyte pointer of the file handle. If the ending currentbyte pointer is odd, then wordincluster variable points to the next word.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
We should not call ATA_close after ATA_read_b. If we do not modify the file, there is no need to close it.

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile  mysong;
  int test;
  AtaUint16 Data[256];
/*Do media specific intialization*/

    /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit (&state);

  /* Initialize the file structure with first entry in root directory */
  error = ATA_fileInit(&state, &mysong);

 /* Test if first entry in root directory is directory or file
  return value 1 for directory , 0 for file */
  test =  ATA_isDir (&mysong);

  if(!test)
  {
   /* read 11 bytes from first file in root directory and place in Data array. The reading is
done assuming big endian*/
   error =  ATA_read_b (&mysong, Data, 11);
  }
```

5.42. ATA_readLittleEndian_b

Name
AtaError ATA_readLittleEndian_b (AtaFile *pAtaFile, AtaUint16 *Data, AtaUint16 bytes)

Synopsis

To read “bytes” number of bytes from *pAtaFile, and place it in the buffer *Data in little endian format.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.
- *Data – This parameter is a pointer to the receiver data buffer
- bytes – This parameter specifies the number of bytes to be read from the file.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  WordInCluster, Cluster, NextBrokenCluster, CurrentPhySector, CurrentWord, CurrentByte

- **Functional description:**

  The ATA_readLittleEndian_b function calculates the location of the current byte in the file using information from both AtaState and AtaFile structures. After calculating the current sector number, it issues the read command to the media and uses low level functions to read the required data off the sector. It keeps track of the cluster number using the “Cluster” and “NextBrokenCluster” variables. On leaving the function, the variables point to the current word that has just been read from the file. It also updates the currentbyte pointer of the file handle. If the ending currentbyte pointer is odd, then wordincluster variable points to the next word.

Bugs & Limitations

None

References

Refer to appendix C.

Hardware Dependencies

None

Notes

None

Example

```
  /* Declare variables */
  AtaState state;
  AtaError error;
  AtaFile mysong;
  int test;
  AtaUint16 Data[256];
/*Do media specific intialization*/

   /* Initialize all hardware and software variables of file navigation */
  error = ATA_systemInit(&state);

  /* Initialize the file structure with first entry in root directory */
  error = ATA_fileInit(&state, &mysong);

 /* Test if first entry in root directory is directory or file
  return value 1 for directory , 0 for file */
  test =  ATA_isDir(&mysong);

  if(!test)
  {
  /* read 11 bytes from first file in root directory and place in Data array.
   The reading is done assuming little endian*/
   error =  ATA_readLittleEndian_b(&mysong, Data, 11);
  }
```

5.43. ATA_seek_b

Name

AtaError ATA_seek_b (AtaFile *pAtaFile, AtaFileSize ByteOffsetFromStart)

Synopsis

To move the current byte pointer of the file *pAtaFile, to “ByteOffsetFromStart” words from the start of the file.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to seek.
- ByteOffsetFromStart – This parameter specifies the offset in number of bytes from the start of the file

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization

For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**
  WordInCluster, Cluster, CurrentByte

- **Functional description:**
  The ATA_seek_b function resets the position of the cluster pointers to the value specified in the ByteOffsetFromStart value of the function call. It does not read the bytes in between the offset and the start. It also updates the currentbyte pointer of the file handle.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
None

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;
 int test;
 AtaUint32 offset=13;
 /*Do media specific intialization*/

  /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit (&state);

 /* Initialize the file structure with first entry in root drectory */
 error = ATA_fileInit (&state, &mysong);

/* test if first entry in root directory is directory or file
 return value 1 for directory , 0 for file */
 test =  ATA_isDir (&mysong);
```

```
if(!test)
{
/* move by offset from start in first file in root directory */
error = ATA_seek_b (&mysong, offset);
}
```

5.44. ATA_write_b

Name

AtaError ATA_write_b (AtaFile *pAtaFile, AtaUint16 *Data, AtaUint16 bytes)

Synopsis

Writes “bytes” no. of bytes from the array pointed by Data to a file.

Input parameters

- *pAtaFile – This parameter is a pointer to the file to be read.
- *Data – pointer to the data to be written to the media.
- bytes – number of words to be written.

Return value

Return type – AtaError.
ATA_ERROR_NONE on successful initialization
For details about what kind of errors codes can be returned & their reasons, please see appendix A.

Description

- **State variables modified by function**

  pAtaFile->Size, pAtaFile->WordInCluster, pAtaFile->Cluster, pAtaFile->CurrentByte

- **Functional description:**

  Writes “bytes” no. of bytes from the array pointed by Data to a file. It also updates the cluster information & updates the fat with the size. There is no need to flush after this because it calls flush internally. We need to call ATA_close once the writing is complete before removing the media. If the file is being appended to & the filesize is in odd number of bytes, this function takes care of updating the last word of original file with first byte of data & then resumes writing.

Bugs & Limitations
None

References
Refer to appendix C.

Hardware Dependencies
None

Notes
ATA_write_b supports three writing modes (OVERWRITE, APPEND & OVERWRITE+APPEND) internally. The user need not worry about these. But when writing is finished in any of these three modes, ATA_close needs to be called.

ATA_write_b doesn't check for validity of file handle passed.

Example

```
 /* Declare variables */
 AtaState state;
 AtaError error;
 AtaFile mysong;
  unsigned int Data[256] = {0xbeef};
/*Do media specific intialization*/

   /* Initialize all hardware and software variables of file navigation */
 error = ATA_systemInit(&state);

 /* Initialize the file structure with first entry in root drectory */
 error = ATA_fileInit(&state, &mysong);

/* Go to root directory for creating file. First go to directory in which you want to
  create file */
   error = ATA_cdRoot(&mysong);

 ATA_setFileName(&mysong,"test", “txt”);

 /* Create a file , returns 0 for successful creation*/
 error = ATA_create(&mysong);
 errot = ATA_close(&mysong);
 /*Find the file again*/
 /* Write 11 bytes of data in test.txt, returns 0 on successful writing */
 error =  ATA_write_b(&mysong, Data, 11);
 error = ATA_close(&mysong);
```

6. APPENDIX A – ERROR CODES

No.	Error	Code	Description
1	ATA_ERROR_NONE	0	Successful operation
2	ATA_ERROR_UNSUPPORTED	1	The features not supported.
3	ATA_ERROR_NESTED	2	Multiple commands sent to media (MMC).
4	ATA_ERROR_TIMEOUT	4	Media (MMC) didn't responded within stipulated time.
5	ATA_ERROR_BAD_MEDIA	8	Boot Sector corrupted.
6	ATA_ERROR_EOF	16	End of file is encountered.
7	ATA_ERROR_FILE_NOT_FOUND	32	Search for given file failed.
8	ATA_ERROR_ID_NOT_FOUND	64	Media ID not found.
9	ATA_ERROR_MEDIA_NOT_FOUND	128	Media not found.
10	ATA_ERROR_MEDIA_REMOVED	256	Media removed.
11	ATA_ERROR_DISK_FULL	512	Media full.

7. APPENDIX B – CONSTANTS

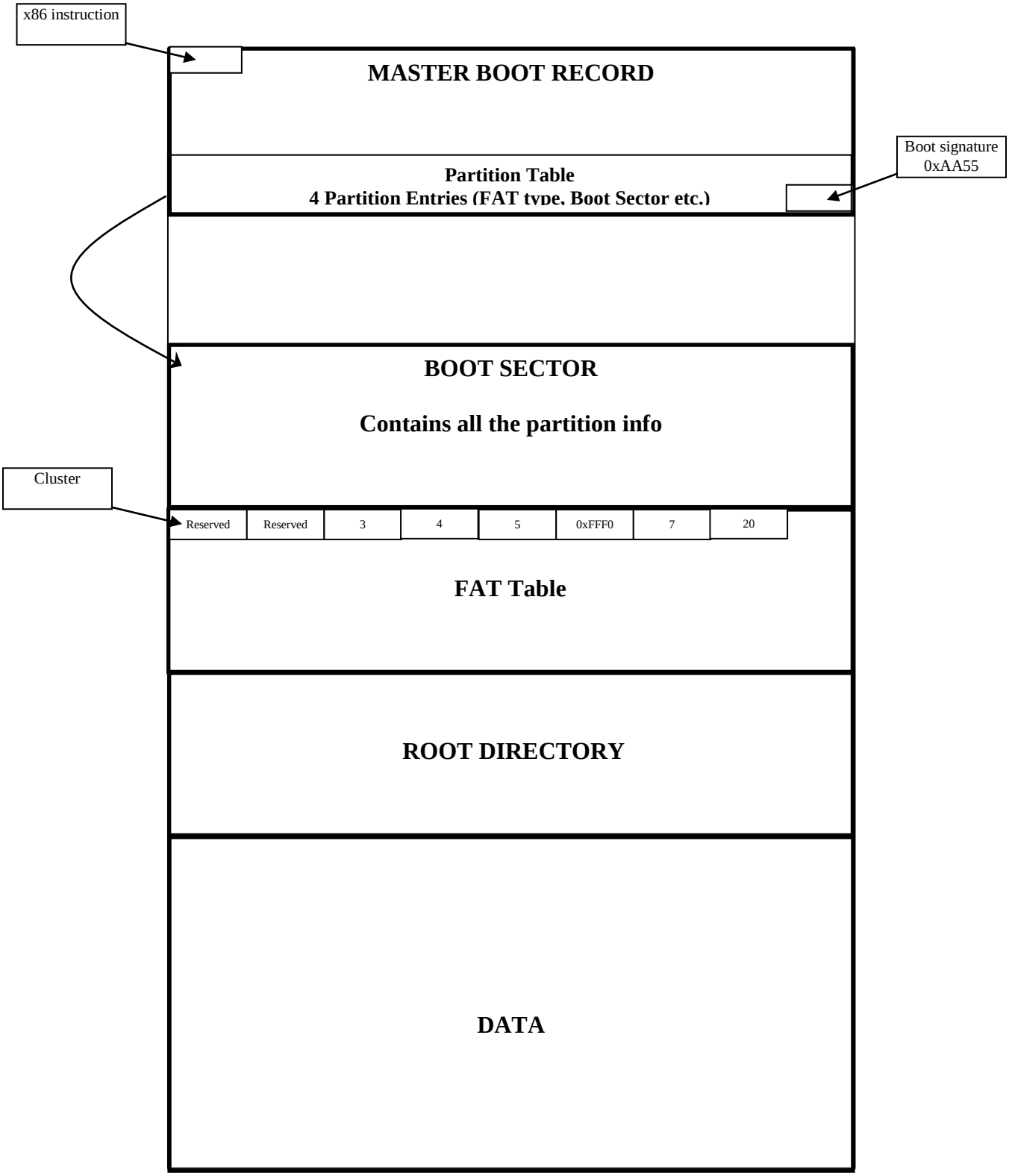
- For Error codes, please consult Appendix A.
- File Attributes: -

```
#define ATA_ATTR_READ_ONLY      0x01
#define ATA_ATTR_HIDDEN         0x02
#define ATA_ATTR_SYSTEM         0x04
#define ATA_ATTR_VOLUME         0x08
#define ATA_ATTR_DIRECTORY      0x10
#define ATA_ATTR_ARCHIVE        0x20
```

- Fat Types :-

```
#define ATA_FAT_TYPE_12     0x01
#define ATA_FAT_TYPE_16     0x04/* 16-bit FAT. Partitions smaller than 32MB */
#define ATA_FAT_TYPE_32     0x0B/* 32-bit FAT. Partitions up to 2047GB */
#define ATA_FAT_TYPE_EXT  0x05/* Extended MS-DOS Partition */
#define ATA_FAT_TYPE_16_DOS4  0x06/* 16-bit FAT. Partitions >= 32MB */
#define ATA_FAT_TYPE_32  0x0B /* 32-bit FAT. Partitions up to 2047GB */
#define ATA_FAT_TYPE_32X 0x0C/* Same as 32(0Bh),but uses LBA Int 13h ext*/
#define ATA_FAT_TYPE_32X13 0x0E /* Same as 16_DOS4(06h), but uses Logical Block Address Int 13h extensions. */
#define ATA_FAT_TYPE_32X13X  0x0F/* Same as EXT(05h), but uses Logical Block Address Int 13h extensions. */
```

8. APPENDIX C – DIAGRAM of FAT MEDIA



First Byte of entry	Indicates
0xE5	Directory entry is free
0x05	Same as before, but indicates a kanji value being used. Usage rules similar to entry of 0xE5
0x20, values less than 0x20, 0x22, 0x2A, 0x2B, 0x2C, 0x2E, 0x2F, 0x3A, 0x3B, 0x3C, 0x3D, 0x3E, 0x3F, 0x5B, 0x5C, 0x5D, 0x7C	Invalid entries

Table C.1 – Filename entry codes

0x0F	Entry is part of longname entry for some other file

Table C.2 – File Attribute Codes

Cluster Count	FAT type
<4085	12
<65525	16
>=65525	32

Table C.3 - Determination of FAT type in AtaSystemInit()

9. Memory Numbers for different scenarios: -

Total memory consumption is around 6KW (.text) and 1KW(.data+. const). This number includes all the features. Memory footprint comes down depending on the APIs the application wants to use. Given below is a breakdown of memory consumption for different types of file system operations.

1. Initialization sequence – 350W.
2. Browsing sequence – 800W.
3. Read/Seek sequence – 700W.
4. File create/write/close/rename/delete sequence – 1.5KW.
5. Disk size/free space/used space calculations – 150W.
6. Long File/Directory browsing – 400W+800W.
7. Long File/Directory Creation/Renaming/Deletion – 1.8KW.

10. APPENDIX D – REFERENCES

- Advanced MS-DOS Programming, Ray Duncan, MS Press
- Inside the Windows 95 File System, Stan Mitchell, O'Reilly
- FAT: General Overview of On-Disk Format, fatgen102.pdf, msdn.microsoft.com

- Long Filename Specifications, lfnspec.pdf, msdn.microsoft.com
- AT Attachment with Packet Interface – 6 (T13-1410D), ATA-6(d1410r0a).pdf, www.t13.org

11. Additional Notes

- Method of Profiling :- The ATA code uses MMC low level routines & the MMC filesystem contains one directory & one file in that directory & root directory. The profiling is done after copying the folder & file onto the MMC after formatting it. While profiling the system clock was 120MHz & MMC module was working with function divisor & clock divisor at respectively 1 & 1.
- The examples in this doc are capable of running by themselves, you just need to cut-paste the code. One sample test project is supplied with the release to facilitate your testing. You just need to modify the test.c in atatest project to cut paste example codes. Just put in your code in place of the #if 1 #endif code area, & your code will run.
- Testbench is supplied along with the release. Details about test bench will be included in the next release of this doc. You have to run ATA_TEST_MMC.out & it should say "All Tests Passed".

12. Bugs Fixed & feature additions in this Version

0001499	File System (FAT)	minor	(susmit)	**09-21**	ATA_format fails for FAT32
0001496	File System (FAT)	feature	(susmit)	**09-21**	Should have a seek function which can do relative seek.
0001466	File System (FAT)	**major**	(susmit)	09-16	File seek does not work.
0001462	File System (FAT)	minor	(susmit)	09-14	ATA_renameLong & ATA_deleteLong leaves invalid LFN entries when applied over some LFNs.
0001451	File System (FAT)	minor	(susmit)	09-11	ATA_read & ATA_seek doesn't return ATA_ERROR_EOF for some specific conditions.
0001450	File System (FAT)	**major**	(susmit)	09-11	Internal API _AtaReadWord doesn't return ERR_STATUS properly.
0001448	File System (FAT)	**major**	(susmit)	09-10	Unable to create longfilenames when LFN entries are spread on multiple non-contiguous clusters.

0001447	File System (FAT)	minor	(susmit)	09-10	If we try to create more than 512 files in a root directory, an lost cluster comes on disk.
0001438	File System (FAT)	minor	(susmit)	09-09	ATA_createDirectoryLong fails to create case sensitive short directories
0001437	File System (FAT)	**major**	(susmit)	09-08	ATA_renameLong sometimes overwrite other LFN entries.
0001427	File System (FAT)	**block**	(susmit)	09-06	Unable to create more than 62 files in a subdirectory on a FAT16 32MB MMC.
0001421	File System (FAT)	minor	(susmit)	09-06	Need API for setting & creating Long Directory Name.
0001399	File System (FAT)	minor	(susmit)	09-05	ATA_getLongName return wrong longfilenames for LFNs of zero size.
0001395	File System (FAT)	minor	(susmit)	09-05	Deletion of directory using ATA_delete or ATA_deleteLong does not free up the cluster.
0001367	File System (FAT)	**major**	(susmit)	09-03	Issue with use of 2 Atastates.
0001366	File System (FAT)	minor	(susmit)	09-03	ATA-FS does not report error when more than 512 file create calls are placed on a FAT16 root directory.
0001363	File System (FAT)	minor	(susmit)	09-02	Attribute setting is not absolute
0001360	File System (FAT)	**major**	(susmit)	08-31	AtaState._AtaWriteCurrentPhySector is not initialized properly.
0001359	File System (FAT)	minor	(susmit)	08-31	Currentbyte field is not updated properly.
0001354	File System (FAT)	**major**	(susmit)	08-31	Overwriting using ATA_write causes a FAT corruption problem
0001353	File System (FAT)	minor	(susmit)	08-30	ATA_isDir return values doesn't match with documentation.
0001322	File System (FAT)	minor	(susmit)	08-20	ATA_vol doesn't return correct volume label for FAT32 disk.
0001308	File System	**major**	(susmit)	08-	ATA 2.21ROM makes a

	(FAT)			20	reference to a global variable
0001309	File System (FAT)	feature	(susmit)	08-18	Internal function references are resolved directly

0001291	File System (FAT)	**major**	(susmit)	**08-15**	ATA_label fails to set Label properly if there is a LFN in the root directory.
0001231	File System (FAT)	minor	(susmit)	08-14	ATA_write() does not report error when Disk Full
0001230	File System (FAT)	minor	(susmit)	08-13	ATA_delete() has some bugs with empty file

0001170	File System (FAT)	**major**	(susmit)	08-05	ATA_diskSize & ATA_diskFree fails for large FAT32 Disks.
0001134	File System (FAT)	**major**	(susmit)	07-29	LFN create fails when LFN ends just before the checksum byte.
0001130	File System (FAT)	minor	(susmit)	07-29	DOS Filename indexes of LFNs are not proper.
0001108	File System (FAT)	tweak	(susmit)	07-23	ATA dependencies packaging
0001109	MMC	tweak	(susmit)	07-23	MMC v2.2 release with ATA?
0001118	File System (FAT)	minor	(susmit)	07-23	Bug when 2 AtaState structures are used and they use the same MMC state.

0001029	File System (FAT)	major	(susmit)	06-28	More than 16 files in the root directory of a FAT32 formatted disk can't be browsed.
0000998	File System (FAT)	**major**	(susmit)	06-28	ATA-FS 2.1 slower than 1.3
0001020	File System (FAT)	**major**	(susmit)	06-27	ATA_createDirectory API creates invalid directory entries
0000461	File System (FAT)	feature	(susmit)	06-25	Long filename support needed for creating and renaming files.
0000778	File System (FAT)	feature	(susmit)	06-25	Additional ATA - API

0000932	File System (FAT)	major	(susmit)	06-25	ATA_cdRoot always returns ATA_ERROR_NONE
0000933	File System (FAT)	major	(susmit)	06-25	Filename is not NULL terminated when set using ATA_setFileName
0000981	File System (FAT)	major	(susmit)	06-22	Creation of more than 16 zero size files in a newly created subdirectory gives scandisk errors.
0000934	File System (FAT)	feature	(susmit)	06-18	Need support for AtaWriteSector + AtaReadSector+Mediapresent API's with return of appropriate status
0000838	File System (FAT)	tweak	(susmit)	06-06	ATA_systemInit needs to return more error codes

13. Appendix E

- get_time & get_mod_time returns an int where first 5 bits means hour (upto 24hours), next 6 bits minutes, lowest 5 bits seconds.
- get_date & get_mod_date returns an int where first 7 bits are for year (offset from 1980), next 4 bits for month, lowest 5 bits for day.

These two functions need to be defined by the application to pass the current time & date to the ATA module.

14. Note About Long Filename Support

- *Long Filename/Directory Creation* : You need to first browse & go to the directory where you want to create the file. Then you need to call ATA_setLongFileName/ATA_setLongDirectoryName API with the AtaFile handle & the long name. This will compute the corresponding short name entry from the long name & update your AtaFile handle with that. Then you need to call ATA_createLong/ATA_crateDirectorLong with the AtaFile handle & the long name. After ATA_createLong/ATA_crateDirectorLong exits, you must call ATA_close of the AtaFile handle.
- *Long Filename/Directory Deletion* : You need to first browse & go to the directory where your file/Directory is located & find the file/Directory inside that directory. This can be done by using ATA_findFirst & then by subsequent ATA_findNext calls. As you are looking to delete a long filename/Directory, after calling ATA_findNext, you need to call ATA_getLongName to find out whether you have reached the correct file/Directory or not . Once the long filaname returned by

ATA_getLongName matches the long filename/Directory you want to delete, you need to call ATA_deleteLong with the current AtaFile handle. The current AtaFile handle will be automatically pointing to the 8.3 entry of the long filename you want to delete during the course of browsing. If you call ATA_delete instead of ATA_deleteLong, then only the 8.3 entry will be deleted & unused LFN entries will be there, so subsequent scandisk will return an error.

- *Long FileName/Directory Renaming* : You need to first browse & go to the directory where your file/Directory is located & find the file/Directory inside that directory. This can be done by using ATA_findFirst & then by subsequent ATA_findNext calls. As you are looking to delete a long filename/Directory, after calling ATA_findNext, you need to call ATA_getLongName to find out whether you have reached the correct file or not . Once the long filaname returned by ATA_getLongName matches the long filename/Directory you want to rename, you need to call ATA_renameLong with the current AtaFile handle & with the old & new long filename. The current AtaFile handle will be automatically pointing to the 8.3 entry of the long filename you want to rename during the course of browsing. If you call ATA_rename instead of ATA_renameLong, then only the 8.3 entry will be renamed & unused LFN entries will be there & new LFN entries won't be created, so subsequent scandisk will return an error.
- *Long Filename Writing* : You need to first browse & go to the directory where your file is located & find the file inside that directory. This can be done by using ATA_findFirst & then by subsequent ATA_findNext calls. As you are looking to write to a long filename, after calling ATA_findNext, you need to call ATA_getLongName to find out whether you have reached the correct file or not . Once the long filaname returned by ATA_getLongName matches the long filename you want to write to, you need to call ATA_write with the current AtaFile handle. The current AtaFile handle will be automatically pointing to the 8.3 entry of the long filename you want to write to during the course of browsing. Once you finish writing on that file, you need to call ATA_close.

15. Revision Information

- First Document – 30NOV 2001 – Susmit
- Updation of Data Structures for fragmentation fix– 10DEC 2001 -- Susmit
- Updation for added APIs – 18DEC 2001 -- Susmit
- STDIO API DOC updated. – 19DEC 2001 – Mihir
- Final DOC updation done for 21DEC release – 20DEC 2001 – Susmit
- Document Updation for 31JAN release – 31JAN2002 – Susmit
- Document Updation for 12FEB release – 12FEB2002 – Susmit
- Document Updation for 28FEB release – 27FEB2002 – Susmit

- Document Updation for 28FEB release – 28FEB2002 – Susmit
- Document Updation for ATA1.4 release – 15MAR2002 – Susmit
- Document Updation for ATA2.1 release -- 16APR2002 – Susmit
- Document Updation of ATA2.2 Release – 1JUL 2002 – Susmit
- Cycle count updation for ALL APIs for ATA2.2 – 2JUL2002 – Susmit
- Document Updation for release for ROM Code – 9AUG2002 – Susmit
- Document Updation for release 2.21 – 15AUG2002 – Susmit
- Document Updation for 14SEP release – 14 SEP 2002 – Susmit
- Document Updation for 23SEP release – 21 SEP 2002 – Susmit
- Removed memory numbers & put in more effective number count. – Susmit
- Removed cycle counts as they should come from profile data – Susmit
- Final Document Updation for 23SEP release – 21 SEP 2002 – Susmit