Índice

DEVICE DATASTORE

1. Introduction

The DATASTORE device, is used to manage the reads and writesof the removable memory MMC (Multi Media Card) and in the non-removable memory (NAND) of the Qmove family instruments.

1.1 Installation

1.1.1 DEVICE DECLARATION IN THE CONFIGURATION UNIT

In the configuration unit, the BUS section must be declared so that you have the hardware resources required for the implementation of the DATASTORE device.

```
;--------------------------------
; Internal device declaration
;--------------------------------
INTDEVICE
<device name>    DATASTORE          TCamp          <port_type>
```

where:

<device name>	The name given to the device.
DATASTORE	Keyword that identifies the device.
Tcamp	Time sampling device (1÷10 ms).
<port type>	Connected port type (MMC on port 0)

1.1.1.1 Example

```
;--------------------------------
; Internal device declaration
;--------------------------------
INTDEVICE
DEV        DATASTORE        0001        0
```

1.2 Operation

1.2.1 INTRODUCTION

The access to MMC storage device, it is implemented through a “file_system” and not with direct access. In this way the saved data can be shared with any PC and the device is seen by it as a simple floppy disk.

File system features The file system supports the following types of FAT:
FAT12 Used on floppy disks or storage devices with limited capacity.
FAT16 Used in most storage devices. It can get to store 2 GB of data.
FAT32 Used in high-capacity storage devices, over 2 GB.

The file-system implemented, allows access to the file for reading and writing. Does not allow you to create directories, and then those files must be present in the single “root directory” . The device is also able to open/create files whose names or ASCII conversion of a number and the extension is fixed.

Supports two types of files:

- **BINARY format**: the file is written directly as if it were an array of long (“.hex” extension);
- **CSV format (Comma Separated Values)**: data is written to a table with a number of columns can be set. The data are separated by “;” or is on a maximum number of characters for any data (fixed number).

1.2.2 FILES AND DATAS FORMAT

Supported file formats are the BINART and the CSV. In our implementation are available the reading and appending functions (writing to END file), but not overwrite function on all types of files.

BINARY format The binary files have “hex” extension. In our implementation you can write and read a binary file where data values are stored in long (32bits) in hexadecimal base (HEX). To write or read such a file you must set the parameter *filetype* = 0.

1.2.2.1 Example:

If in the olongXX variables we have the following values:

```
olong01 = 0 => H'00000000
olong02 = -1 => H'FFFFFFFF
olong03 = 10000 => H'00002710
olong04 = -10000 => H'FFFFD8F0
```

We run the WRITE command by setting the parameter "*field* = 4", we get to the end of the file:

```
00000000FFFFFFFF00002710FFFFD8F0''
```

CSV format The csv file will have "csv" extension. Data written in these files are organized in a table formed from a settable number of columns, each column is called a "field". The rows of the table are called "record". The DATASTORE device does not consider the presence of the record header. The maximum number of field is set to 32 (ilong / olong).

To write some long data type, "10000", "-1", "2147483647" and "-2147483648", by setting the "*filetype* = 1" parameter, we will have:
The file system supports the following types of FAT:

```
Field1------------ +Field2--------- +Field3-------- +Field4-----+
10000              -1               2147483647      '2000000000
```

In this way the occupation of space in each record is fixed. If we set instead "*filetype* = 3", we will have:

```
10000;-1;2147483647;'2000000000
```

The major evaluation software including Microsoft Excel, OpenOffice, Lotus 123, successfully read the fixed width file (*filetype* = 1) but when you save it into file with the field delimiters (*filetype* = 3).
The main difference between a format and the other is in the search of records.
Read a particular record using the format 1, It is not difficult as the start position within the file can be calculated easily given the record length is fixed.
Read instead a record with the format 2, implies a search within the file record start position. This operation may be long.

1.2.3 ACCESS TO FILES

To use the (MMC) storage device You must use the device command that allows to recognize it and activate it. The MOUNT command allow to execute this operationn. The device will be activated when the state *st_mount* take the value 1.
Please note that after each command, the device marks is handling that command with the *st_busy* state. When the *st_busy* state is to 1 the device cannot access other commands.
To open an existing file or open a new file, set the *filetype* parameter. It allows you to specify the format of the file you intend to treat.
Below we will divide the discussion on whether the filetype parameter takes the following values:
a) 0: binary files;
b) 1-2: CSV file with fixed space for data;
c) 3-4: CSV file with data separated by ";".
Before sending the command of opening the file you have to specify which file is open with the *filenuum* parameter.
If the file is present in the storage device it is open otherwise it creates a new file with the name specified in *filenum* file.
When the *st_openfile* state is to 1 the file is opened.

a) In the binary files data is present one after the other. It's possible:

- Add a data in queued;
- overwrite a given already in the list (knowing the location).

To read one or more data to a binary file, you must set the *record* parameter what data you begin reading and in the *filed* parameter how much data to read up to 32 data with one reading. At this point you have to use the READ command.
Data read are shown in the *ilong01÷ ilong32* parameters.

1.2.3.1 Example:

Reading of 3 data in the position 11, 12, 13 in the binary file.

```
DEV:record=11
DEV:field=3
READ DEV
WAIT NOT DEV: st_busy
glVar1=DEV: ilong01
glVar2=DEV: ilong02
glVar3=DEV: ilong03
```

To write one or more data to a binary file you must set the*record* parameter what data start overwrite and in the *field* parameter how much data will overwrite.
Data values by overwriting the data in the file should be inserted into *olong01÷ olong32* parameters.

1.2.3.2 Example:

Writing of 3 datas int the position 11, 12, 13 in the binary file.

```
DEV:record=11
DEV:field=3
DEV:olong01=123        ;first data
DEV:olong02=456        ;second data
DEV:olong03=789        ;third data
WRITE DEV
WAIT NOT DEV: st_busy
```

To append new data to a binary file, you must use the method described above remembering to set the *record* parameter to -1.

B) In the CSV files with fixed datas space, is creates a table with columns, called field, and rows, called record. The field numbers with which to create the table you must specify before you open the new file in the *field* parameter.
For each entry in this table is reserved a fixed space. In this way each record, composed of a number field of data, fixed space in file.
To read a record of such a file, you must set in the *record* parameter, the number of records to read and send the READ command. In the *ilong01-ilong32* parameters, are shows the read values from the record.

To overwrite a record of such a file, set the parameter records the number of records to write, set in the *olong01-olong32* paramenters a number of values equal to the number of field in the record and then send the WRITE command.
To append a new record to the file you must set the value -1 in *record*.

1.2.3.3 Example:

Read of the record 3 (Composed of three field):

```
DEV:record = 3
READ DEV
glVar1 = DEV:ilong01
glVar2 = DEV:ilong02
glVar3 = DEV:ilong03
```

C) As in the case B, but in this case you cannot read or overwriting a particular record. You can only read sequentially around the file and write just by adding a new record.

Summary table:

Operations	filetype		
	0	1-2	3-4
OPEN/CLOSE	yes	yes	yes
DELETE	yes	yes	yes
READ sequential records*	yes	yes	yes
READ indexed records*	yes	yes	no
WRITE ad a record	yes	yes	yes
WRITE overwrite a record	yes	yes	no

* = **READ sequential records**: reading of records in file starting from the first record to the last. This kind of reading does not provide the ability to read records in random order within the file. This kind of reading are only going to read records from the first and reading always following the last read.
* * = **READ indexed records**: This kind of reading can read any record in the file even in random order.

1.3 Commands and parameters table

1.3.1 SYMBOLS USED

The name of the parameter, state or command is shows at the left of the table.

R
Indicates if the parameter or state is retentive (upon initialization of the device maintains the previously defined state), or the state assumes upon initialization of the device.
If the device does not need to initialize the “R” field indicates the value that the parameter take to power up of the card.
R = Retentive
0 = Upon initialization of the device the value is forced to zero.

1 = Upon initialization of the device the value is forced to one.
- = Upon initialization of the device is presented significant value.

D
Indicates the **paramenter size**.
F = Flag
B = Byte
W = Word
L = Long
S = Single Float

1.3.1.1 Conditions

Are describe all **conditions that must exist is considered correct or because the command is accepted**.
In some cases, limit values are specified for the acceptance of the parameter: If there are any values outside the limits set, the data is in any case accepted; ptherefore appropriate controls of the application must be provided to ensure the proper functioning.
To run a command, all the conditions must be met; otherwise the command is not sent.

A
Indicates the **access mode**.
R = Read.
W = Write.
RW,= Read / Write.

1.3.2 Parameters

The following are the parameters, variables, and commands you need to execute the device.

Name	D	R	A	Conditions	Descriptions
priority	B	5	R-W	-	**Priority** In the **Qmove** products specifies the execution priority of the device compared to the performance of the QCL and display tasks (Range 0÷10). In the **Qmove+** products specifies the media at which the device references (0 = SD/MMC, 1=NAND internal, 2= USB).
filetype	B	0	R-W	-	**File Format** Defines the features of the file (Range 0÷4): Type 0 type = BINARY record not separated field not separated by “,” single marks are not used header not present Type 1 type = CSV records separated by = carrige return and line feed (Windows) field separated by spaces single marks are not used header not present Type 2 type = CSV records separated by = line feed (Unix) field separated by spaces single marks are not used header not present Type 3 type = CSV records separated by = carrige return and line feed (Windows) field separated by = “;” (hex 3b) single marks are not used header not present Type 4 type = CSV records separated by = line feed (Unix) field separated by = “;” (hex 3b) single marks are not used header not present

Name	D	R	A	Conditions	Descriptions
field	W	0	R-W	filetype=0÷4	**Field** In the case *filetype*=0, indicates the number of records to be overwritten starting from the record indicated by the record parameter. In the case *filetype*=1÷4 indicates the number of fields that will be written for each record. The value is used only by creating a new file. (Range 0÷32)
filenum	L	0	R-W	-	**File Number** Defines the “filename” of the file to open considering the converting of the number in ASCII characters. (Range 0÷99999999)

1.3.3 Variables

Name	D	R	A	Conditions	Description
disksize	L	0	R	-	**Disk Size** Indicates the size of the memory device in bytes. The value is updated when MOUNT command is executed.
diskfree	L	0	R	-	**Disk Free** Indicates the free space on the memory device in bytes. The value is updated when MOUNT command is executed and at the closing and writing of a file.
filesize	L	0	R	-	**File Size** Indicated the file size in bytes. The value is updated upon opening and at each subsequent write operation.
numrecord	L	0	R	st_openfile	**File Record Number** Indicates the number of records in the file. The value is updated upon opening and at each subsequent write operation. If 0, it means that the file does not contain correctly formatted records (is set also the *error* variable), or that it is a new file.
numfield	W	0	R	-	**Field Number** Indicates the number of fields in the first record read. The value is updated upon first opening or at the time of first writing.
record	L	0	R-W	-	**Record Number** Indicates the number of records (*row*) that is processed by WRITE and READ commands. If '1, with the WRITE command will execute an insert.
ilong01÷32	L	0	R	-	**Input Long nr. 1 - 32**
ibyte0	B	0	R	-	**Input Byte 0 (LSB) ilong01** Indicates the byte nr. 0 (LSB) of the long ilong01
ibyte1	B	0	R	-	**Input Byte 1 ilong01** Indicates the byte nr. 1 of the long ilong01
ibyte2	B	0	R	-	**Input Byte 2 ilong01** Indicates the byte nr. 2 of the long ilong01

ibyte3	B	0	R	-	**Input Byte 3 (MSB) ilong01**
					Indicates the byte nr. 3 (MSB) of the long ilong01.
					`olong01  xxxxxxxx`
					`obyte3  xx------ (MSB)`
					`obyte2  --xx---- +`
					`obyte1  ----xx-- +`
					`obyte0  ------xx (LSB)`

olong01-32	L	0	R-W	-	**Output Long nr. 1 - 32**
obyte0	B	0	R-W	-	**Output Byte 0 (LSB) olong01** Indicates the byte nr. 0 (LSB) of the long olong01
obyte1	B	0	R-W	-	**Output Byte 1 olong01** Indicates the byte nr. 1 of the long olong01
obyte2	B	0	R-W	-	**Input Byte 2 olong01** Indicates the byte nr. 2 of the long olong01

obyte3	B	0	R-W	-	**Input Byte 3 (MSB) olong01**
					Indicates the byte nr. 3 (MSB) of the long olong01.
					`ilong01 xxxxxxxx`
					`ibyte3 xx------ (MSB)`
					`ibyte2 --xx---- +`
					`ibyte1 ----xx-- +`
					`ibyte0 ------xx (LSB)`

errcode	B	0	R	-	**Error Code** Indicates the last error that occurred in the device management commands. 0 = no error 1 = n.d. (reserved) 2 = open file error 3 = read data error 4 = write data error 5 = file system error (reserved) 6 = file system error (reserved) 7 = file system error (reserved) 8 = file system error (reserved) 9 = file system error (reserved) 10 = record format error 11 = unknow error
errvalue	B	0	R	-	**Error Value** Always 0.

Name	D	R	A	Conditions	Description
wrncode	B	0	R	-	**Warning Code** Indicates the last warning incurred in the performance of the device management commands. 0 = no warning 1 = warning data 2 = warning command 3 = device not inserted
wrnvalue	B	0	R	-	**Warning Value** Always 0.

1.3.4 States

Name	D	R	A	Conditions	Description
st_mount	F	0	R	-	**Device Mount** Activation indicates that in inserted a storage device. The device checks for a new device after the MOUNT command.
st_openfile	F	0	R	-	**Open file** Activation indicates that the file has been opened/created successfully. Failure to activate may indicate an error file or opening: the error is reported by the “error” variable.
st_busy	F	0	R	-	**Busy** Activation indicates that the system is busy in an operation and therefore you should not remove the storage media.
st_error	F	0	R	-	**Error**
st_warning	F	0	R	-	**Warning**

1.3.5 Commands

Name	D	R	A	Conditions	Description
MOUNT	-	-	-	st_openfile = 0	**Mount Device** Checks for a new device (if detected). The *st_mount* state becomes 1.
UMOUNT	-	-	-	st_openfile = 0	**Unmount Device** This command is used to “close” the device so that it can be removed. The st_mount state becomes 0.
OPENFILE	-	-	-	st_mount = 1 st_openfile = 0	**Open File** Opens the “filenum” file. If the file does not exist, is created. The st_openfile state becomes 1.
CLOSEFILE	-	-	-	st_openfile = 1	**Close File** Close the “filenum” file. The st_openfile state becomes 0.
DELFILE	-	-	-	st_mount = 1 st_openfile = 0	**Delete File** Delete the “filenum” file. To be deleted, the file must be closed.
READ	-	-	-	st_openfile = 1	**Read Record** Reads the record indicated by the “record” variable. All field reads are placed in the ilongXX from ilong01 variables.
WRITE	-	-	-	st_openfile = 1	**Append Record** Run the append of a new record, by taking values from olongXX variables starting from olong01.
RSERR	-	-	-	-	**Clear Error** Reset the st_error parameter and errcode and errvalue variables.
RSWRN	-	-	-	-	**Clear Warning** Reset the st_warning parameter and *wrncode* and *wrnvalue* variables.

1.4 Limitations

The data SINGLE type are not deal. You must convert these internal data before transferring them in MMC.

1.5 Application example

1.5.1 VERIFY A DEVICE:

```
WAIT NOT DEV:st_busy                ; is required because the device when RUN
                                    ; is in a state of busy until it ends
                                    ; initialization operations.
MOUNT DEV                           ; run the MOUNT of the device
WAIT NOT DEV:st_busy                ; waits the finish of the operation
IF DEV:st_mount                     ; if the device is present,
  <operations>                      ; the st_mount state is 1
ENDIF
```

1.5.2 FILE OPENING

```
DEV:filetype = 0                    ; sets the type of BINARY file
DEV:filenum = 123                   ; sets the name of the file as "123.hex"
WAIT NOT DEV:st_busy
OPENFILE DEV                        ; run the command to open
WAIT NOT DEV:st_busy                ; waits for the operation finish
IF DEV:st_openfile                  ; if the file is opened successfully,
```

```
  <operations>                    ; the st_openfile state is 1
ENDIF
```

1.5.3 WRITING SOME RECORDS, STARTING FROM THE PREVIOUSLY OPENED FILE:

```
DEV:field = 2                     ; sets the number of fields that
                                  ; is composed the record
                                  ; Note: if the file type is BINARY,
                                  ; this parameter indicates the number of long
                                  ; that will be written/read.
DEV:record = -1                   ; instructs the device to execute the operation
                                  ; from the end of the file (appending).
DEV:olong01 = 123456              ; sets the data to write
DEV:olong02 = 654321
WAIT NOT DEV:st_busy
WRITE DEV                         ; execute the write command
WAIT NOT DEV:st_busy              ; waits for the operation to finish
IF DEV:st_error                   ; check if there have been errors in writing
  <error>
ENDIF
```