# The EasySDK Guide

## Table of contents

# 1 Hardware Description

An EasyFlash cartridge consists of two flash memory chips, called LOROM chip and HIROM chip. In the current hardware version, each of them has a size of 512 KiB.

Only 8 KiB of memory per chip can be seen in the address space of the C64 at once. Therefore EasyFlash supports banking. When each chip has a size of 512 KiB, there are 512 / 8 = 64 banks. The EasyFlash cartridge allows software to select the active bank.

The mapping of the chips into the C64 memory is controlled by two lines of the Expansion Port: /GAME and /EXROM. The "/" means that their meaning is inverted, i.e. 0 or low means active.

The possible states of these lines and their meaning is listed in Table 1.

| /GAME | /EXROM | Comment |
|---|---|---|
| 1 | 1 | Cartridge memory invisible ("Invisible Mode") |
| 1 | 0 | 8 KiB from Chip 0 at $8000 ("8K Mode") |
| 0 | 0 | 8 KiB from Chip 0 at $8000, 8 KiB from Chip 1 at $A000 ("16K Mode") |
| 0 | 1 | 8 KiB from Chip 0 at $8000, 8 KiB from Chip 1 at $E000 ("Ultimax Mode") |

*Table 1: States of /GAME and /EXROM lines*

These lines can be controlled by software, we'll come back to it soon.

The resulting memory model is shown in Fig. 1.

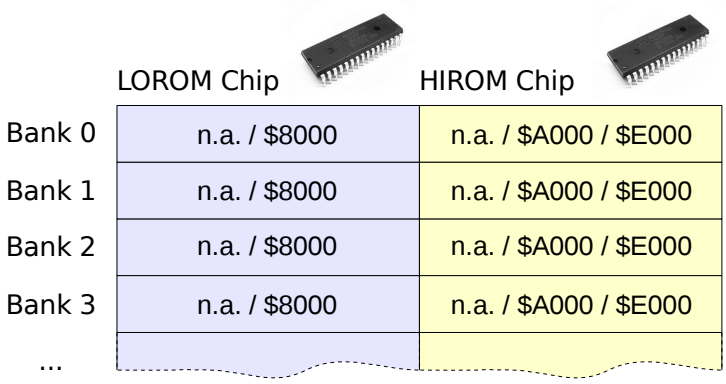


*Fig. 1: EasyFlash memory model*

If the Boot switch is in position "Boot" (or jumper open) and the computer is reset, EasyFlash is started normally: The Ultimax memory configuration is set, bank 0 selected. The CPU starts at the reset vector at $FFFC. As in Ultimax mode the HIROM chip is banked in at $E000, the flash must contain a valid reset vector and some start-up code. This is described in a separate chapter.

If the Boot jumper is in position "Disable" (or jumper closed) EasyFlash starts with cartridge memory hidden. However, software like EasyProg can write to the flash memory because it can override the jumper setting.

The software can chose the active bank and set the memory location by using two I/O registers. They are described in the following chapters.

Additionally an EasyFlash cartridge has 256 Bytes of RAM. This memory is always visible. It can be used to save small portions of code or data in a way which is unlikely to interfere with other software.

## 1.1 Register $DE00 – EasyFlash Bank

This is a write-only register located at $DE00. The active flash memory bank is chosen using this register. Basically it selects the state of the higher address lines. The bank selection can be made regardless if the flash memory is actually visible according to the memory configuration or not. The selected bank is valid for both chips, LOROM and HIROM.

| **Bit** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Meaning** | 0 | 0 | Bank number | | | | | |

*Table 2: Register $DE00*

The value after reset is $00. The bits 6 and 7 should always remain 0, unless stated otherwise in this document.

## 1.2 Register $DE02 – EasyFlash Control

This is a write-only register located at $DE02. The software can control the memory configuration of the cartridge using this register. Additionally it can be used to set the state of the status LED.

| **Bit** | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| **Meaning** | L | 0 | 0 | 0 | 0 | M | X | G |

*Table 3: Register $DE02*

The value after reset is $00.

| **Position** | **Name** | **Comment** |
|---|---|---|
| 7 | L | Status LED, 1 = on |
| 6..3 | 0 | reserved, must be 0 |
| 2 | M | GAME mode, 1 = controlled by bit G, 0 = from jumper "boot" |
| 1 | X | EXROM state, 0 = /EXROM high |
| 0 | G | GAME state if M = 1, 0 = /GAME high |

*Table 4: Bits of $DE02*

Following memory configurations are possible:

| **MXG** | **Type** |
|---|---|
| 000 | GAME from jumper, EXROM high (i.e. Ultimax or Off) |
| 001 | Reserved, don't use this |
| 010 | GAME from jumper, EXROM low  (i.e. 16K or 8K) |
| 011 | Reserved, don't use this |
| 100 | Cartridge ROM off (RAM at $DF00 still available) |
| 101 | Ultimax (Low bank at $8000, high bank at $e000) |
| 110 | 8k Cartridge (Low bank at $8000) |
| 111 | 16k cartridge (Low bank at $8000, high bank at $a000) |

*Table 5: Configuration of cartridge memory maps*

Most likely the software is only interested in setting combinations with M = 1. Of course the

usual memory configurations selected with the 6510 register $01 do also apply.

## 1.3 RAM at $DF00

From $DF00 to $DFFF there are 256 bytes of RAM. This memory is not part of the normal 64 KiB of internal C64 RAM but part of the I/O memory space.

# 2 Conventions

Conventions can make the life easier and look professional. So let's invent some.

## 2.1 Address Format Convention

Addresses of EasyFlash have a special format in this document and related tools:

BB:C:FFFF

| | |
|---|---|
| BB | Bank number, currently supported range 00..3F |
| C | Chip number, 0 for LOROM chip or 1 for HIROM chip |
| FFFF | Offset in this bank and chip, 0000..1FFF |

*Table 6: Address format convention*

The address 00:1:1FFC means bank 0, HIROM chip, offset 0x1FFC. This is the address which has to contain a reset vector to make the cartridge work.

## 2.2 Scan the Keyboard

When a cartridge is plugged into the C64, often the user has no other way to get to the BASIC prompt than unplugging the cartridge.

EasyFlash has a switch or a jumper to avoid booting. However, let's make it even easier for our users:

When you implement start-up code for EasyFlash, scan the keyboard as early as possible. If the Stop key is held down, make the cartridge invisible and call the Kernal's reset vector. You can hide (“kill”) the cartridge by writing $04 to $DE02.

# 3 Supported cartridge types

Different types of cartridge images can be written to and started from an EasyFlash cartridge. Let's have find out how these work.

## *3.1 Normal 8K Cartridges*

Normal 8K cartridges consist of up to 8 KiB of ROM visible at LOROM. This type of cartridges pull down /EXROM and leave /GAME high.

The memory is visible at $8000 and contains some magic bytes. The C64 Kernal detects these bytes in the reset code. If such a cartridge is found, it gets started with JMP($8000).

EasyFlash cartridges always start in Ultimax mode. Therefore there is a small boot code at the end of the HIROM flash chip on bank 0, i.e. at 00:1:FFxx. This code copies itself to RAM and checks whether the Stop key is pressed (refer to chapter 5). If not, it starts the 8K Cartridge.

To do this, it does some initializations which would be done by the kernel normally. Then it sets the 8K mode using $DE02. Finally it does JMP($8000).

Xxxxxxxxx

Bild

xxxxxxxxxx

# 4 Native EasyFlash Cartridges

## 4.1 Introduction

EasySDK comes with some tools and code snippets to create, write and read EasyFlash cartridges.

This chapter contains information about a recommended structure of these cartridges.

Chapter 4.2 describes the start-up code, which is executed directly after a CPU reset. This start-up code is the only mandatory part of an EasyFlash cartridge.

EasyFS is the Easy File System. This is a simple ROM file system which simplifies implementing software for EasyFlash. The structure of EasyFS is described in chapter 4.3.

Additionally there are some library functions to read EasyFS files and to write to the flash chips. Chapter Fehler: Referenz nicht gefunden specifies this interface.

Of course developers are free to implement cartridges which do not use EasyFS or EasyAPI by directly accessing the I/O registers described in this document. It is also possible to use only EasyFS without EasyAPI or the other way around.

An example memory layout of an EasySDK cartridge using EasyFS is shown in Fig. 2.

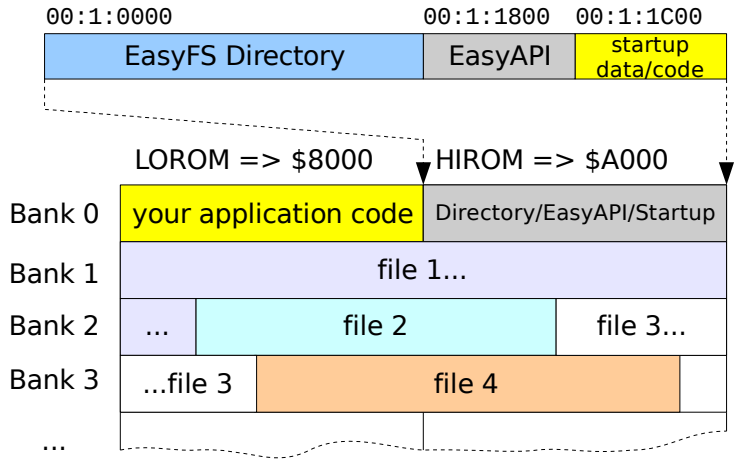


*Fig. 2: An EasyFS cartridge*

## 4.2 Start-Up Code

As already mentioned earlier, EasyFlash cartridges always start in Ultimax mode. Therefore there is a small boot code at the end of the HIROM flash chip on bank 0.

The memory area from 00:01:1800 to 00:01:1BFF is reserved for EasyAPI. So the startup code and data can occupy the area 00:01:1C00 to 00:01:1FFF.

Todo: Explain what the code has to do.

## 4.3 Easy File System (EasyFS)

Cartridges using EasyFS always use the 16 KiB cartridge mode, which is configured by the start-up code by setting /GAME and /EXROM active (low). All explanations in this chapter refer to this configuration.

If your cartridge makes use of the Easy File System (EasyFS), it contains a directory structure. This structure resides at 00:1:0000. This means it is visible at $A000 in 16 KiB mode when bank

0 is selected.

A directory shall not contain more than 255 entries (excluding end mark). We will see that each entry has a size of 24 Bytes, so the whole directory can have a size of up to $1800 Bytes. Behind the directory there is space of $0800 bytes for EasyAPI and the start-up code and data.

Remember that HIROM is banked in in Ultimax mode at $E000 after a reset. Therefore the start-up code contains the reset vector.

The format of EasyFS directory entries is as follows:

| Field | Type | Comment |
|---|---|---|
| Name | 16 bytes | File name, 16 characters PETSCII (?), 0-padded |
| Flags | 1 byte | File type and some flags |
| Bank | 1 byte | Bank where the file starts, 0..63 |
| Bank High | 1 byte | Reserved for high byte of bank, currently always 0 |
| Offset | 2 bytes | File start offset in this bank 0x0000..0x3FFF, little endian |
| Size | 3 bytes | File size, little endian |

The flags field has following structure:

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| Meaning | H | R | R | Type | | | | |

The bit *H* means hidden. If this is 1, this file is not intended to be seen by a user, a file browser should not show it.

`Reserved bits marked with R must be set to 1.`

Table 7 shows the possible values for *Type*.

| Value | Type |
|---|---|
| $00 | `Files with this type are marked as invalid, they must be skipped. Note that the flags of this file may be not 0.` |
| $01 | `Normal PRG file with 2 bytes start address` |
| ... | |
| $10 | Normal 8 kByte cartridge (0x8000..0x9FFF) |
| $11 | Normal 16 kByte cartridge (0x8000..0x9FFF, 0xA000..0xBFFF) |
| $12 | Normal Ultimax cartridge (0x8000..0x9FFF, 0xE000..0xFFFF) |
| $13 | Normal Ultimax cartridge, first bank not used (0xE000..0xFFFF) |
| ... | |
| $1F | End of directory. This entry is only a terminator. |

*Table 7: EasyFS file types*

Note that erased flash memory does have the value $FF, so all bits are *one*. This means that an entry located in erased flash has the type 0x1f naturally.

Furthermore, when a flash is written, bits can only turn from *one* to *zero*. It is not possible to change a bit from *zero* to *one* when writing to flash memory. That's why the file type $00 marks a deleted or invalid file. $00 can be written regardless from the old file type by only changing *ones* to *zeros*.

The meaning of the file types $10 to $13 will be described in chapter 11.

## *4.4 EasyAPI*

EasyFlash cartridges may contain a block of code which is called EasyAPI. This application

programming interface supports reading files from EasyFS, erasing blocks of flash memory and write data to flash.

Note that it is not supported to write real files to the flash and to the directory structure. Writing is done on a lower logical layer only, by directly addressing flash memory.

If a cartridge makes use of it, EasyAPI is always located from 00:1:1800 to 00:1:1BFF. The first four bytes of this memory area must be $65 $61 $70 $69 (“EAPI”) to indicate the presence of EasyAPI.

There is one very important feature for EasyAPI: When a CRT image which contains the “EAPI” signature explained above is written to a cartridge, the EasyAPI memory area is updated with the latest version of the EasyAPI code. The jump table of this code will remain compatible, but the new code may support new Flash chip types or bug fixes.

An imaginary example shows why this is done in this way: Let's say a programmer creates a cartridge in 2009 and adds the latest EasyAPI code directly to his CRT image. This code would only support the Am29F040 chips. In 2010 somebody builds a new version of EasyFlash which use a different type of flash memory. The software of our poor programmer wouldn't be able to save game states or whatever to the new kind of EasyFlash.

This situation is solved by the replace mechanism: When an updated version of EasyProg (the tool for writing CRT images to cartridges) is used to write the old CRT image, the EasyAPI is automatically replaced with an updated version.

To add the EasyAPI code to your self-implemented CRT images, include the binary (*todo*) at 00:01:1800 in your CRT image.

The following chapters explain the entries to the jump table. The flash address 00:1:1800 is seen at $B800 in normal C64 mode. Therefore we use this address format directly.

### 4.4.1 EasyAPI Memory Usage

The functions of EasyAPI do not pollute the C64 RAM. All data they need is stored into the EasyFlash RAM at $DF00.

Programs using EasyAPI must not modify the RAM contents at **$DF80..$DFFF** between calls being related (e.g. Between EFSETNAM, EFSETLFS and EFLOAD).

Programs which use the extended API functions EFX* must not change **$DF40..$DFFF** between calls to EFXPREP and other EFX* functions.

Outside calls to EasyAPI which are related the application may use the whole memory block $DF00..$DFFF as needed.

The remaining bytes at $DF00 to $DF3F are not used by EasyAPI and can always be used by the application.

### 4.4.2 EasyAPI Signature

| Position | Length | Comment |
|---|---|---|
| $B800 | 4 | EasyAPI signature, always $65 $61 $70 $69 (“EAPI”) |

This signature is only used to show the existence of EasyAPI. If the tool EasyProg finds this signature, it can replace this memory area 00:1:1800 to 00:1:1BFF with a newer version of EasyAPI.

### 4.4.3 EasyAPI Version

| Position | Length | Comment |
|---|---|---|
| $B804 | 2 | EasyAPI version, currently $01 $00 = 1.0 |

This version number has an informational purpose only.

### 4.4.4 EFSETNAM - $B806

This should be a drop-in replacement for KERNAL's SETNAM. It prepares the file name for EFOPEN. The file name will be copied into the EasyFlash private RAM, so the original memory doesn't need to be kept after this call.

The IRQ flag isn't touched, IRQs may be active. Do not hide the I/O area or the HIROM area before calling this.

**Parameters**

A Length of file name

X Low byte of file name address

Y High byte of file name address

**Call**

JSR $B806

**Return**

-

**Changes**

All registers are preserved

### 4.4.5 EFSETLFS - $B809

This should be a drop-in replacement for KERNAL's SETLFS. It prepares the parameters for EFLOAD.

The IRQ flag isn't touched, IRQs may be active. Do not hide the I/O area or the HIROM area before calling this.

**Parameters**

A Logical File number (use the value 1, currently ignored)

X Device number (ignored)

Y Secondary address, 0 to load to file address, > 0 to load to new address

**Call**

JSR $B809

**Return**

-

**Changes**

All registers are preserved

### 4.4.6 EFLOAD - $B80C

This should be a drop-in replacement for KERNAL's LOAD. Load a file into memory according to the parameters set with EFSETNAM and EFSETLFS.

**Parameters**

A Load flag, ignored

X Low byte of new load address (when EFSETLFS secondary address > 0)

Y High byte of new load address (when EFSETLFS secondary address > 0)

**Call**

JSR $B80C

**Return**

C set for error, A contains error: $04 file not fount, $1d load error

**Changes**

All registers are preserved

The IRQ flag isn't touched, IRQs may be active. Do not hide the I/O area or the HIROM area before calling this.

### 4.4.7 EFXPREP - $B8xx

To make it easier to patch the EasyAPI functions into existing software, all functions have a second version which can be called independently from the memory configuration. The only precondition is that the I/O area is visible.

These extended versions of the functions are named EFX*. These functions do the same as their original version EF*, but do some additional preparations:

1. SEI
2. Save the current contents of $01
3. Activate cartridge ROM using $01 and $DE02
4. Call the original EF* function
5. Hide cartridge ROM using $DE02
6. Restore old $01
7. CLI
8. RTS

These functions are located in the EasyFlash RAM, they are copied there by the function EFXPREP. Es long as you don't touch this memory area, you can call them there. If you overwrite the RAM area reserved for EasyFlash, you can restore the contents by calling EFXPREP again. When calling EXPREP, the IRQ flag isn't touched, IRQs may be active. Do not hide the I/O area or the HIROM area before calling this.

**Parameters**

-

**Call**

JSR $B8xx

**Return**

-

**Changes**

A, X, Y

These are the addresses for the EFX* functions:

| Original Function | Original Address | Extended Function | Address |
|---|---|---|---|
| EFSETNAM | $B806 | EFXSETNAM | $DF46 |
| EFSETLFS | $B809 | EFXSETLFS | $DF49 |
| EFLOAD | $B80C | EFXLOAD | $DF4C |

## *4.5 Hybrid cartridges*

An image may also contain one or more normal 8 kByte, 16 kByte or Ultimax sub-modules. These have to be placed bank-aligned, because they are not loaded, but banked in and started.

This special feature may be useful for something like CRT collections.

That is the reason a directory may also have entries which refer to sub-cartridges.

Fig. 3 shows an example hybrid cartridge which contains a 16K sub-cartridge.

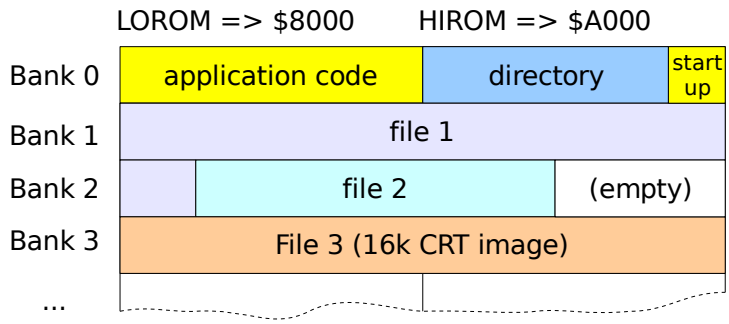


*Fig. 3: Hybrid cartridge containing a normal 16K cartridge*