halec
Herrnröther Str. 54
63303 Dreieich
Germany

www.halec.de

Manual

roloFlash AVR
Base Version

Document version 1.0.1 as of 2012-02-24
(Software version: 1.7)

Table of Contents

I PREFACE

- roloFlash allows for mobile and PC-independent flashing of your Atmel AVR microcontroller based products.
- Let untrained personnel or customers flash your products, as operating errors are conceptually impossible.
- No PC and specific tool chains (e.g. from Atmel) are necessary.
- Use roloFlash in field, at your customers' sites and in large- and small-batch production.
- Gain more freedom by employing a uniform process for all supported microcontroller families*.

Term "Target"

The term "target" is used to mean your products to be flashed. The products contain the microcontroller to be flashed. From now on, this term is used regularly throughout this document.

* At the moment Atmel AVR series, more microcontroller families on request.

II Package Contents

Carefully check the scope of supply:

- roloFlash AVR
- microSD card
  - prepared for use in your roloFlash, containing documentation, examples, firmware and roloBasic compiler
  - for insertion into roloFlash's card slot
- Adapter for microSD card
  - for use in PCs with a card slot only fitted for SD cards

Note: The microSD card is either inserted into roloFlash or into the adapter or is enclosed separately.

III DESCRIPTION

ISP Jack:

This connector is plugged on the ISP plug of the target to be programmed. The specification of the ISP plug can be found in the Atmel documentation (e. g. description for STK500).

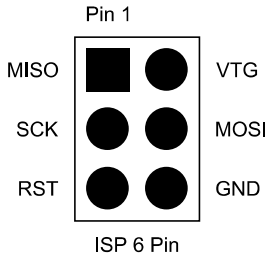

Fig.: *Top view pinout* ***ISP plug*** *on the target PCB*

LEDs:

Five programmable bi-color (red and green) LEDs. Using the LEDs, you can e.g.

- show a running light visualizing the flash process
- output errors in red

microSD Card Slot:

For a microSD or microSDHC card comprising the compiled script to be run (RUN.BIN) as well as the images to be flashed.

WARNING: Please exclusively employ microSD cards being provided with an SD card adapter by the manufacturer. Only these microSD cards have mandatoy support for the necessary protocol. Furthermore, we recommend using microSD cards from Kingston or SanDisk exclusively.

IV USAGE

The process is divided into two parts:

- Preparation of the microSD card at the PC (e.g. in the development department)
- Flashing the targets (e.g. untrained personnel in the production plant, customer or field engineer in field)

1 Preparing the microSD card at the PC (e.g. in the development department)

The relevant file is "RUN.BIN", which is being processed by roloFlash for flashing.

- In case you want to format a microSD card, do so using Windows 7 (Windows XP is not suitable).
- Model the desired process in roloBasic. For this, you either use or adapt an example script. The approprate list of procedures and functions provided by roloFlash can be found in chapter VII "List of Procedures and Functions". The file you create should have the file extension ".BAS".
- Your script can point to standard ".HEX" files (Intel Hex format: "INTEL 16" / "I16HEX") which are to be flashed to the target.
- Run the compiler "`rbc.exe`". This creates a compiled file of the same name with the extension ".BIN".
- Rename the file to "RUN.BIN" or instead of running "`rbc.exe`" run the batch file `compile.bat`, which creates "RUN.BIN" from "RUN.BAS". After that, copy the file "RUN.BIN" and the files needed by the script (usually a ".HEX" file) to the microSD card.

You can store the script files (".BAS"), the compiled files (".BIN") and the compiler at your own discretion on the PC and/or on the microSD card.

roloFlash only evaluates the file "RUN.BIN" (as well as the files being referenced by the code).

2 Flashing the target (e.g. untrained personnel in production plant)

Here, the process is straightforward:

- Supply target with power.
- Plug roloFlash on the target's 6-pin ISP plug.
- roloFlash gets powered by the target and automatically starts processing the file "RUN.BIN", where usually the actual flashing is carried out. Meanwhile, for instance, a green running light can visualize the flashing process.
- Remove roloFlash – done.

v UPDATING ROLOFLASH

roloFlash has its own firmware, which can be updated.

Starting the update

- For updating, the firmware file "RFAVRBAS.HMP" must be present in the microSD card's root directory.
- The update process gets triggered if
  - roloFlash gets plugged on an arbitrary target **without** a microSD card in its card slot, or
  - a previous update failed. In this case, the order of plugging roloFlash on a target and inserting a microSD card does not matter.
- There is no check if the firmware on the microSD card is newer or older. Thus, you can return to an older version, if you ever need to.

The update process

- The target only serves as power supply.
- The process is being displayed by means of the LEDs, see chapter VIII, 2. "Updating roloFlash".
- As long as the microSD card is not inserted, LED 1 is lit in red.
- During the update, LED 2 and 3 additionally flash quickly. roloFlash should not yet be removed.
- In case of success, LED 1 and 2 light up in green afterwards.
- roloFlash remains in this state until removed.
- The updated firmware is now available.

In case the update failed, please use a microSD that has been freshly formated with FAT32 under Windows 7, and that only contains the file "RFAVRBAS.HMP".

For a production plant, it is recommended not to leave the file "RFAVRBAS.HMP" on the microSD card.

VI LIST OF SUPPLIED ROLOBASIC SCRIPTS

NORMAL.BAS

- Checks if the target controller has the right signature
- Checks the voltage range
- Erases the chip
- Sets the fuses
- Writes and verifies the flash memory with `IMAGE.HEX`
- Meanwhile green running light, in the end, LED 5 stays lit if successful
- Writes result to the log file

VERSIONS.BAS

- Writes all version numbers to the log file:

```
companyName
deviceName
softwareVersion
hardwareVersion
bootloaderVersion
imageVersion
```

VII List of Procedures and Functions

Procedures:

Procedures have no return value. The parameters to be passed must be specified without parentheses.

Example:

```
setProgrammingSpeed 1000
```

Functions:

Functions do have a return value. The parameters to be passed must be put in parentheses.

Example:

```
f = readFuses(fuses_low)
```

For functions without parameters, the parentheses can be left out.

Example:

```
value = getTargetPresent
```

or

```
value = getTargetPresent()
```

1 Target in general

The target can be in one of the following modes:

RunMode

Target runs normally, as if roloFlash wasn't connected.

ProgramMode

Target can be programmed.

Some procedures change the mode (`programTarget` and `runTarget`).

Other procedures and functions depend on a certain mode. In these cases it is mentioned in their respective description.

1.1 getTargetPresent

```
value = getTargetPresent
```

Determines if a target is connected. The mode stays unaltered.

If the target is in RunMode, the target temporarily gets switched to ProgramMode. A program which might be running on the target will thereby be restarted.

Note:

roloFlash should always be connected to a target, because there is no energy supply otherwise. This function is meant to be used mainly for programming devices having their own power supply.

Furthermore it is conceivable to plug roloFlash onto something different than a target. Hence, this function actually establishes a communication to the target.

Preconditions:

- none

Parameters:

- none

Return value:

0 = no target found

1 = target found

Exceptions:

- none

1.2 programTarget

`programTarget`

- Stops target.
- Puts target and roloFlash in ProgramMode.

Preconditions:

- none

Parameters:

- none

Return value:

- none (procedure)

Exceptions:

`targetCommunication`	Communication with target does not work.

1.3 runTarget

`runTarget`

- Starts target.
- Puts roloFlash in RunMode.

Preconditions:

- none

Parameters:

- none

Return value:

- none (procedure)

Exceptions:

- none

1.4 restartTarget

`restartTarget`

The target gets restarted and runs through the reset-cycle. If the target was in ProgramMode, this mode gets restored. A firmware possibly existing on the target might already have run for a short time after the end of the reset-cycle.

It is advised to run this command only if this behavior is either not critical or no firmware is present on the target.

This procedure is needed when changing fuses where the change should become active immediately. This is especially true for activating a quartz crystal on the target PCB, which afterwards enables higher programming speeds.

A common sequence:

```
writeFuses(f)   ! for activating the quartz crystal
restartTarget
setSpeed 1000   ! 1 MHz
```

```
writeFileToFlash 0, "IMAGE.HEX"
```

Preconditions:

- none

Parameters:

- none

Return value:

- none (procedure)

Exceptions:

`targetCommunication`	Communication with the target does not work.

1.5 setProgrammingSpeed

```
setProgrammingSpeed <speed>
```

Sets the programming speed for the target.

Preconditions:

The target must be in ProgramMode.

Parameters:

speed

Specification in kHz. Supported values are:
4000, 2000, 1000, 500, 250, 125 kHz
`speed_min`: Sets the minimum speed (i.e. 125 kHz).
If the given frequency is not in this list, it gets rounded to the next possible value.

Return value:

- none (procedure)

Exceptions:

`targetWrongMode`	Target is not in "ProgramMode"
`rangeValue`	Invalid value for speed.
`typeFault`	Invalid type for speed.

1.6 getTargetVoltage

```
u = getTargetVoltage
```

Determines the voltage in mV delivered by the target.

Preconditions:

- none

Parameters:

- none

Return value:

Read out voltage in mV.

Exceptions:

- none

2 Target Fuses & Lock Bits

2.1 readBits

```
values = readBits(index)
```

Reads out the given fuse or lock bits.

Preconditions:

The target must be in ProgramMode.

Parameters:

index

Determines which fuse or lock bits are to be read. For this purpose, the constants `fuses_low`, `fuses_high`, `fuses_ext` and `lock_bits` exist.

Controllers without extended fuse return an undefined value for `fuses_ext` (no exception is generated).

Return value:

Read out fuses or lock bits.

Exceptions:

`targetWrongMode`	Target is not in "ProgramMode".
`targetCommunication`	Communication with target does not work.
`rangeValue`	Invalid value for index.
`typeFault`	Invalid type for index.

2.2 writeBits

`writeBits index, values`

Writes the specified fuse or lock bits.

Warning:

- Set the lock bits last, after all other accesses to the chip.
- If you want to work on a chip locked by lock bits, first issue an `eraseFlash`. This also resets the lock bits.

Preconditions:

The target must be in ProgramMode.

Parameters:

index

Determines which fuse or lock bits are to be written. For this purpose, the constants `fuses_low, fuses_high, fuses_ext and lock_bits` exist.

For controllers without extended fuse, writing to `fuses_ext` has no effect (no exception is generated).

values

Values to be written.

Return value:

- none (procedure)

Exceptions:

`targetWrongMode`	Target is not in "ProgramMode".
`targetCommunication`	Communication with target does not work.
`rangeValue`	Invalid value for index or value.
`typeFault`	Invalid type for index or value.

3 Target Signature and Memory Layout

3.1 getSignature

```
s = getSignature()
```

Reads out the target's signature. By means of the signature you can distinguish the various controllers.

If the utilized controller is known, the layout of flash and EEPROM get configured automatically.

Otherwise, the layout of flash and EEPROM must be configured manually by using `setFlashLayout and setEepromLayout`.

You can use `getFlashLayout` to determine if the utilized controller is known to roloFlash or not. In case it's not, `getFlashLayout` throws the exception "unknownMemoryLayout".

Preconditions:

The target must be in ProgramMode.

Parameters:

- none

Return value:

Read out signature. The signature is returned as an array of 3 bytes.
There is a constant for each known target, so you can determine if the target is of the expected type with the following line:

```
if getSignature() = sig_atmega32 ! Test auf ATmega32
```

Exceptions:

`targetWrongMode`	Target is not in "ProgramMode".
`targetCommunication`	Communication with the target does not work.

3.2 getFlashLayout

```
a = getFlashLayout()
```

Returns the size of the flash and the size of a flash page. These values have been set either manually by `setFlashLayout` or, in case of a known controller, automatically by `getSignature()`.

Preconditions:

The target must be in ProgramMode.

Parameters:

- none

Return value:

Array with 2 elements:

a[0] = Size of flash in bytes

a[1] = Size of a flash page in bytes

Exceptions:

`targetWrongMode`	Target is not in "ProgramMode".

3.3 setFlashLayout

```
setFlashLayout <size, pagesize>
```

Sets size of flash and flash page. For the values matching the target controller, consult the Atmel documentation.

If the utilized controller is known, the matching values are determined automatically upon calling `getSignature()`. In that case, calling `setFlashLayout` is unnecessary und not recommended.

Preconditions:

The target must be in ProgramMode.

Parameters:

size

Size of flash in bytes.

pagesize

Size of flash page in bytes.

Return value:

- none (procedure)

Exceptions:

`targetWrongMode`	Target is not in "ProgramMode".

3.4 getEepromLayout

```
a = getEepromLayout()
```

Returns the size of the EEPROM and the size of an EEPROM page. These values have been set either manually by `setEepromLayout` or, in case of a known controller, automatically by `getSignature()`.

Preconditions:

The target must be in ProgramMode.

Parameters:

- none

Return value:

Array with 2 elements:

a[0] = Size of EEPROM in bytes

a[1] = Size of an EEPROM page in bytes

Exceptions:

`targetWrongMode`	Target is not in "ProgramMode".

3.5 setEepromLayout

`setEepromLayout <size, pagesize>`

Sets size of EEPROM and EEPROM page. For the values matching the target controller, consult the Atmel documentation.

If the utilized controller is known, the matching values are determined automatically upon calling `getSignature()`. In that case, calling `setFlashLayout` is unnecessary und not recommended.

Preconditions:

The target must be in ProgramMode.

Parameters:

size

Size of EEPROM in bytes.

pagesize

Size of EEPROM page in Bytes.

Return value:

- none (procedure)

Exceptions:

`targetWrongMode`	Target is not in "ProgramMode".

3.6 setExtendedAddressMode

`setExtendedAddressMode <value>`

For controllers with 256 kB flash, the normal command set for programming via the ISP interface is not sufficient. An extended address mode is needed.

If the utilized controller is known, the matching value is determined automatically upon calling `getSignature()`. In that case, calling `setFlashLayout` is unnecessary.

Preconditions:

The target must be in ProgramMode.

Parameters:

value

value = 0: Do not use extended address mode

value = 1: Use extended address mode

Return value:

- none (procedure)

Exceptions:

`targetWrongMode`	Target is not in "ProgramMode".
`rangeValue`	Invalid value for value.

`typeFault`	Invalid type for value.

3.7 clearMemoryLayout

`clearMemoryLayout`

Clears an already existing memory layout (flash and EEPROM layout).

Preconditions:

- none

Parameters:

- none

Return value:

- none (procedure)

Exceptions:

- none

4 Write and Verify Target with Hex Files

4.1 EraseFlash

`eraseFlash`

Erases the complete flash of the target. Depending on the fuse settings (EESAVE) the EEPROM gets also erased (default) or not.

Preconditions:

The target must be in ProgramMode.

Parameters:

- none

Return value:

- none (procedure)

Exceptions:

`targetWrongMode`	Target is not in "ProgramMode".
`targetCommunication`	Communication with the target does not work.

4.2 writeFileToFlash

```
writeFileToFlash <filesystem, filename>
```

Writes a HEX file into the target's flash.

Preconditions:

The target must be in ProgramMode.

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for file names apply, see chapter 5 "Files".

Return value:

- none (procedure)

Exceptions:

`targetMemoryLayout,` `targetExtendedAddress,` `hexFileSize,` `hexFileCRC`	See chapter "Exceptions of roloFlash".
`targetWrongMode`	Target is not in "ProgramMode".
`targetCommunication`	Communication with the target does not work.
`typeFault`	Invalid type for filename.
`<various file system exceptions>`	See chapter "Exceptions of the file system".

4.3 verifyFileToFlash

`verifyFileToFlash <filesystem, filename>`

Compares a HEX file with the data in flash.

Preconditions:

The target must be in ProgramMode.

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for file names apply, see chapter 5 "Files".

Return value:

- none (procedure). If data differs, an exception is thrown.

Exceptions:

`targetVerify`	Data read during verify differed.
`targetMemoryLayout`, `targetExtendedAddress`, `hexFileSize`, `hexFileCRC`	See chapter "Exceptions of roloFlash".
`targetWrongMode`	Target is not in "ProgramMode".
`targetCommunication`	Communication with the target does not work.
`typeFault`	Invalid type for filename.
`<various file system exceptions>`	See chapter "Exceptions of the file system".

4.4 writeVerifyFileToFlash

`writeVerifyFileToFlash <filesystem, filename>`

Writes a HEX file into the target's flash and compares the flashed data with the HEX file.

The behavior is the same as calling `writeFileToFlash` first and then `verifyFileToFlash`, but can be faster.

Preconditions:

The target must be in ProgramMode.

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for file names apply, see chapter 5 "Files".

Return value:

- none (procedure). If data differs, an exception is thrown.

Exceptions:

`targetVerify`	Date read during verify differed. See chapter "Exceptions of roloFlash".
`targetMemoryLayout`, `targetExtendedAddress`, `hexFileSize`, `hexFileCRC`	
`targetWrongMode`	Target is not in "ProgramMode".
`targetCommunication`	Communication with the target does not work.
`typeFault`	Invalid type for filename.
`<various file system exceptions>`	See chapter "Exceptions of the file system".

4.5 writeFileToEeprom

`writeFileToEeprom <filesystem,` filename>

Writes a HEX file into the target's EEPROM.

Preconditions:

The target must be in ProgramMode.

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for file names apply, see chapter 5 "Files".

Return value:

- none (procedure)

Exceptions:

`targetMemoryLayout`, `targetExtendedAddress`, `hexFileSize`, `hexFileCRC`	See chapter "Exceptions of roloFlash".
`targetWrongMode`	Target is not in "ProgramMode".

`targetCommunication`	Communication with the target does not work.
`typeFault`	Invalid type for filename.
`<various file system exceptions>`	See chapter "Exceptions of the file system".

4.6 verifyFileToEeprom

`verifyFileToEeprom <filesystem, filename>`

Compares a HEX file with the data in EEPROM.

Preconditions:

The target must be in ProgramMode.

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for file names apply, see chapter 5 "Files".

Return value:

- none (procedure). If data differs, an exception is thrown.

Exceptions:

`targetVerify`	Data read during verify differed.
`targetMemoryLayout, targetExtendedAddress, hexFileSize, hexFileCRC`	See chapter "Exceptions of roloFlash".
`targetWrongMode`	Target is not in "ProgramMode".
`targetCommunication`	Communication with the target does not work.
`typeFault`	Invalid type for filename.
`<various file system`	See chapter "Exceptions of the file system".

exceptions>

4.7 writeVerifyFileToEeprom

`writeVerifyFileToEeprom <filesystem, filename>`

Writes a HEX file into the target's EEPROM and compares the flashed data with the HEX file.

The behavior is the same as calling `writeFileToEeprom` first and then `verifyFileToEeprom`, but can be faster.

Preconditions:

The target must be in ProgramMode.

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for file names apply, see chapter 5 "Files".

Return value:

- none (procedure). If data differs, an exception is thrown.

Exceptions:

`targetVerify`	Data read during verify differed. See chapter "Exceptions of roloFlash".
`targetMemoryLayout, targetExtendedAddress, hexFileSize, hexFileCRC`	
`targetWrongMode`	Target is not in "ProgramMode".
`targetCommunication`	Communication with the target does not work.
`typeFault`	Invalid type for filename.
`<various file system`	See chapter "Exceptions of the file system".

exceptions>

5 Files

Filenames:

- File names must follow the 8.3 rule: "XXXXXXXX.YYY".
- Only characters "A" - "Z", "0" - "9", as well as "_" and "-" are permissible.
- Only capital letters are allowed.

Directory names:

- Directory names may consist of 8 characters at maximum: "XXXXXXXX".
- Otherwise, the same conventions apply as for file names.

Current directory is always the root directory:

- There is no "change directory". The current path always remains the root directory. Hence, a filename must always include the complete path.
- The supported separators between directory parts and filenames are "/" and "\".

5.1 fsCreate

```
fsCreate <filesystem, filename>
```

Creates the file specified. After that, the file is still closed. If the file already exists, this procedure has no effect.

If a file is to be created and written to, it needs to be opened additionally:

```
fsCreate 0, "TEST.TXT"
handle = fsOpen(0, "TEST.TXT")
```

Preconditions:

- none

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for file names apply, see chapter 5 "Files".

Return value:

- none (procedure)

Exceptions:

`typeFault`	Invalid type for filename.
`<various file system exceptions>`	See chapter "Exceptions of the file system".

5.2 fsRemove

```
fsRemove <filesystem, filename>
```

Removes the specified file or directory, if it exists.

Preconditions:

- none

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for file names apply, see chapter 5 "Files".

Return value:

- none (procedure)

Exceptions:

`fileNotFound`	The specified file does not exist.
`typeFault`	Invalid type for filename.
`<various file system exception>`	See chapter "Exceptions of the file system".

5.3 fsMkDir

```
fsMkDir <filesystem, dirname>
```

Creates the specified directory. If it already exists, this procedure has no effect.

Preconditions:

- none

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

dirname

The requirements for file names apply, see chapter 5 "Files".

Return value:

- none (procedure)

Exceptions:

`typeFault`	Invalid type for dirname.
`<various file system exceptions>`	See chapter "Exceptions of the file system".

5.4 fsFileExists

```
bool fsFileExists(filesystem, filename)
```

Checks if the specified file exists.

Preconditions:

- none

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for file names apply, see chapter 5 "Files".

Return value:

0 = Datei existiert nicht

1 = Datei existiert

Exceptions:

`typeFault`	Invalid type for filename.
`<various file system exceptions>`	See chapter "Exceptions of the file system".

5.5 fsFilesize

```
size = fsFilesize(filesystem, filename)
```

Determines the size of the specified file.

Preconditions:

- File exists.

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for file names apply, see chapter 5 "Files".

Return value:

Size of the file.

Exceptions:

`typeFault`	Invalid type for filename.
`<various file system exceptions>`	See chapter "Exceptions of the file system".

5.6 fsOpen

```
filehandle = fsOpen(filesystem, filename)
```

Opens the specified file.

Preconditions:

The file must already exist. If a new file is to be opened, `fsCreate` has to be used before.

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

filename

The requirements for file names apply, see chapter 5 "Files".

Return value:

A file handle for accessing the file (e.g. for `fsRead` and `fsWrite`) is returned. In addition, the file handle is needed to close the file (`fsClose`).

Exceptions:

`typeFault`	Invalid type for filename.
`<various file system exceptions>`	See chapter "Exceptions of the file system".

5.7 fsRead

```
a = fsRead(filehandle, position, count)
```

Reads the specified number of bytes from the file.

Preconditions:

- Valid `Filehandl returned by fsOpen.`

Parameters:

filehandle

File handle returned by `fsOpen`.

position

Byte position from where the data is read.

count

Number of bytes to read.

Return value:

Array of bytes with read data. The array's size is `count`. If not enough data could be read, the array is smaller by the approprate amount. If you try to read at or after the end of the file, an empty array of size 0 is returned.

Exceptions:

`rangeValue`	Invalid value for filehandle, position or count.
`typeFault`	Invalid type for filehandle, position or count.
`<various file system exceptions>`	See chapter "Exceptions of the file system".

5.8 fsWrite

`fsWrite <filehandle, position, array>`

Writes the passed data to the file.

If the position is greater than the current file size, the file gets filled with random data up to this position.

Preconditions:

- Valid file handle returned by `fsOpen`.

Parameters:

filehandle

Filehandle returned by `fsOpen`.

position

Byte position where the data gets written to.

array

Array of bytes with the data to be written.

Return value:

- none (procedure)

Exceptions:

`rangeValue`	Invalid value for filehandle, position or count.
`typeFault`	Invalid type for filehandle, position or count.
`<various file system exceptions>`	See chapter "Exceptions of the file system".

5.9 fsTruncate

`fsTruncate <filehandle, len>`

Truncates file to the specified length. If the file is already smaller, then this procedure has no effect.

Preconditions:

- Valid file handle returned by `fsOpen`.

Parameters:

filehandle

File handle returned by `fsOpen`.

len

Length the file gets truncated to.

Return value:

- none (procedure)

Exceptions:

`rangeValue`	Invalid value for filehandle.
`typeFault`	Invalid type for filehandle or len.
`<various file system exceptions>`	See chapter "Exceptions of the file system".

5.10 fsClose

`fsClose <filehandle>`

Closes the file. This makes the specified file handle invalid, which thus may not be used anymore.

Preconditions:

- Valid file handle returned by `fsOpen`.

Parameters:

filehandle

File handle returned by `fsOpen`.

Return value:

- none (procedure)

Exceptions:

`rangeValue`	Invalid value for filehandle.
`typeFault`	Invalid type for filehandle.
`<various file system exceptions>`	See chapter "Exceptions of the file system".

5.11 fsSync

```
fsSync <filesystem>
```

Makes sure all data not yet written to the card gets written now. It is recommended to call this procedure after performing write accesses to the card.

Preconditions:

- none

Parameters:

filesystem

This parameter is ignored and should be specified as 0.

Return value:

- none (procedure)

Exceptions:

`<various file system exceptions>`	See chapter "Exceptions of the file system".

6 LEDs

Only one LED lit:

- You cannot turn on more than one LED at a time in roloBasic. This way, system error messages are easier to recognize, as they always use more than one LED.

Numbering and colors:

- The numbering of LEDs in roloBasic is the same as printed on the case: from 1 to 5.
- The LEDs can light up green or red. For this, the constants `color_green` and `color_red` are available.

Non-blocking:

- All procedures of this chapter are non-blocking. This means e.g. that a running light activated by `ledRunningLight` runs in parallel to other roloBasic activities.

6.1 ledOn

`ledOn <index, color>`

Switches the LED to the specified color.

Preconditions:

- none

Parameters:

index

Number of LED

color

`color_green` or `color_red`

Return value:

- none (procedure)

Exceptions:

`rangeValue`	Invalid value for index or color.
`typeFault`	Invalid type for index or color.

6.2 ledOff

`ledOff`

Switches all LEDs off.

Preconditions:

- none

Parameters:

- none

Return value:

- none (procedure)

Exceptions:

- none

6.3 ledBlink

`ledBlink <index, color, speed>`

Makes LED flash with the specified speed.

Preconditions:

- none

Parameters:

index

Number of LED

color

`color_green` or `color_red`

speed

Speed of flashing in ms

Return value:

- none (procedure)

Exceptions:

`rangeValue`	Invalid value for index, color or speed.
`typeFault`	Invalid type for index, color or speed.

6.4 ledRunningLight

`ledRunningLight <from, to, color, speed>`

Activates running light.

Preconditions:

- none

Parameters:

from, to

The light runs from LED '`from`' to LED '`to`'.

If '`from`' is smaller than '`to`', the light runs in reverse.

If '`from`' equals '`to`', this LED is lit constantly.

color

`color_green` or `color_red`

speed

Speed of running light in ms

Return value:

- none (procedure)

Exceptions:

`rangeValue`	Invalid value for from, to, color or speed.
`typeFault`	Invalid type for from, to, color or speed.

6.5 ledRunningLightOutstanding

```
ledRunningLightOutstanding <from, to, color, speed,
outstandingLedNumber>
```

Activates running light with the specified LED having a different color.

Preconditions:

- none

Parameters:

from, to

The light runs from LED '`from`' to LED '`to`'.

If '`from`' is smaller than '`to`', the light runs in reverse.

If '`from`' equals '`to`', this LED is lit constantly.

color

`color_green` or `color_red`

speed

Speed of running light in ms

outstandingLedNumber

Number of LED lighting in different color

Return value:

- none (procedure)

Exceptions:

`rangeValue`	Invalid value for from, to, color, speed or outstandingLedNumber.
`typeFault`	Invalid type for from, to, color, speed or outstandingLedNumber.

7 Miscellaneous

7.1 print

```
print a, b, ...
```

Prints parameters `a, b` etc. The number of parameters is unlimited.

The output gets appended to the file "LOG.TXT". If the file does not exist yet, it gets created.

Preconditions:

- none

Parameters:

a, b, ...

You can output numbers and arrays. Example:

```
value = 42
print "The value is: ", value
```

If a given Parameter is neither a number nor a char-array, nothing is printed.

Return value:

- none (procedure)

Exceptions:

<various file system exceptions>	See chapter "Exceptions of the file system".

7.2 delay

```
delay <duration>
```

Waits for the given duration in ms. Only afterwards the function returns.

Preconditions:

- none

Parameters:

duration

Waiting duration in ms.

Return value:

- none (procedure)

Exceptions:

`rangeValue`	Invalid value for duration.
`typeFault`	Invalid type for duration.

VIII CONSTANTS

1 Version numbers etc.

Name	Value / Meaning
`companyName`	"halec <http://halec.de>"
`deviceName`	"roloFlash AVR, base version"
`softwareVersion`	Version number of firmware
`hardwareVersion`	Version number of hardware
`bootloaderVersion`	Version number of bootloader
`imageVersion`	roloFlash expects the compiler generated image in this version. Therefore, please use a compiler matching the roloFlash firmware.

2 Colors for LEDs

Name	Value / Meaning
`color_green`	1
`color_red`	2

3 Signatures of various controllers

Name	Value / Meaning
Example: `sig_atmega32`	Array with [0x1E, 0x95, 0x02]

The following signature constants are defined:

```
sig_at89s51
sig_at89s52
sig_at90can128
sig_at90can32
```

```
sig_at90can64
sig_at90pwm2
sig_at90pwm216
sig_at90pwm2b
sig_at90pwm3
sig_at90pwm316
sig_at90pwm3b
sig_at90pwm81
sig_at90scr100h
sig_at90usb1286
sig_at90usb1287
sig_at90usb162
sig_at90usb646
sig_at90usb647
sig_at90usb82
sig_atmega128
sig_atmega1280
sig_atmega1281
sig_atmega1284p
sig_atmega128a
sig_atmega128rfa1
sig_atmega16
sig_atmega162
sig_atmega164p
sig_atmega164pa
sig_atmega165
sig_atmega165p
sig_atmega168
sig_atmega168p
sig_atmega168pa
sig_atmega169
sig_atmega169p
sig_atmega16a
sig_atmega16hva
sig_atmega16hvb
sig_atmega16m1
sig_atmega16u2
sig_atmega16u4
sig_atmega2560
sig_atmega2561
sig_atmega32
sig_atmega324p
sig_atmega324pa
sig_atmega325
```

```
sig_atmega3250
sig_atmega3250p
sig_atmega325p
sig_atmega328p
sig_atmega329
sig_atmega3290
sig_atmega3290p
sig_atmega329p
sig_atmega32a
sig_atmega32c1
sig_atmega32hvb
sig_atmega32m1
sig_atmega32u2
sig_atmega32u4
sig_atmega32u6
sig_atmega48
sig_atmega48p
sig_atmega48pa
sig_atmega64
sig_atmega640
sig_atmega644
sig_atmega644p
sig_atmega644pa
sig_atmega645
sig_atmega6450
sig_atmega649
sig_atmega6490
sig_atmega64a
sig_atmega64c1
sig_atmega64m1
sig_atmega8
sig_atmega8515
sig_atmega8535
sig_atmega88
sig_atmega88p
sig_atmega88pa
sig_atmega8a
sig_atmega8hva
sig_atmega8u2
sig_attiny13
sig_attiny13a
sig_attiny167
sig_attiny2313
sig_attiny2313a
```

```
sig_attiny24
sig_attiny24a
sig_attiny25
sig_attiny26
sig_attiny261
sig_attiny261a
sig_attiny4313
sig_attiny43u
sig_attiny44
sig_attiny44a
sig_attiny45
sig_attiny461
sig_attiny48
sig_attiny84
sig_attiny85
sig_attiny861
sig_attiny861a
sig_attiny87
sig_attiny88
```

IX EXCEPTIONS

The exact description how exceptions can be thrown and caught again can be found in the roloBasic manual. If an exception does not get caught, it gets output by means of the LEDs.

Here, if the exception is not a number, the exception "exceptionNotANumber" is output. This is detailled in chapter X.1.3 "Exception occurred". Only user-generated exceptions can be non-numeric.

There are different kinds of exceptions which are all treated equally:

- Exceptions of roloBasic
- Exceptions of the file system
- Exceptions of roloFlashs
- User-generated exceptions

1 roloBasic exceptions

These exceptions occur with errors not related to roloFlash, but with processing of roloBasic. A typical example is an outofRange exception.

A list of these exceptions can be found in the roloBasic manual.

2 File system exceptions

These exceptions occur with errors related to the file system or the microSD card.

Name	Number	Meaning
deviceError	101	The microSD card could not be read from / written to.
badCluster	102	Problems within file system. The file system should

		be checked on a PC for consistency.
notMounted	103	It was tried accessing the microSD card, although it is not registered. This points to a problem with the microSD card.
removeError	104	The microSD card has been removed.
createError	105	Creation of a file or directory failed.
fileNotOpen	106	File is not open.
fileNotFound	107	The given file or directory could not be found.
diskFull	108	The microSD card is full.
truncateError	109	Shortening a file with fsTruncate failed.
illegalCluster	110	Problems within file system. The file system should be checked on a PC for consistency.
fileLocked	111	It was tried to open an already open file a second time. Maybe an fsClose was forgotten.
outOfFileHandles	112	The maximum number of open files is limited to 3. It was tried to open more files.

3 roloFlash exceptions

Name	Number	Meaning
exceptionIsNotANumber	200	An exception that is not a number was thrown and not caught within Basic. In this case, the original exception is discarded and replaced by this exception. This can only occur with user generated exceptions, since all other functions solely use the numeric exceptions described here. Example: `throw "Error"`
imageTooLarge	201	The Basic script is too large. About 1024 bytes can be loaded at the most. Please check the size of the compiler generated file.

imageWrongVersion	202	The compiler employed does not match the firmware running on roloFlash. It is recommended to use the respectively latest compiler and firmware for roloFlash.
productWrongVersion	203	It was tried to load an image of a different product onto roloFlash. E.g. it was tried to load an image for the industry version of roloFlash onto the base version of roloFlash.
imageNotFound	204	The microSD card could be mounted, however, the file RUN.BIN could not be found.
targetWrongMode	210	The called procedure or function requires a certain mode of the target. For instance, the procedure `setProgrammingSpeed` requires ProgramMode.
targetCommunication	211	Communication error with the target.
targetMemoryLayout	212	The controller's memory layout has not been determined. With known controllers this happens automatically upon calling `getSignature()`. Otherwise, the layout can be set manually using `setFlashLayout()` and `setEepromLayout()`.
targetExtendedAddress	213	It has not been determined if the controller has an ExtendedAddressMode. With known controllers this happens automatically upon calling `getSignature()`. Otherwise, this can be done manually using `setExtendedAddressMode()`.
targetVerify	214	While reading back the data, differences were detected.
hexFileSize	230	Unplausible size of the given hex file. Maybe the hex file is corrupt.
hexFileCRC	231	Checksum error while parsing hex file. Maybe the hex file is corrupt.

4 User generated exceptions

- Users can generate exceptions himself using `throw`. These can be numeric and use existing values, e.g.:
  `throw rangeError`

- To make it easier to tell user generated exceptions from other exceptions, different exceptions numbers can be used. For this purpose, the constant `userException` with value 1000 is provided. The advantage of this value is that it has an especially well noticable flashing code when the exception is not caught. The constant is usable as offset for own exceptions:
- The constant is can be used as offset for own exceptions:
  `throw userException + 1`
- Non-numeric exceptions can be thrown, too. If such an exception does not get caught, it is finally converted to the exception `exceptionIsNotANumber`, and its code is blinked out, e.g.:

  ```
  throw "error"
  ```

x Meaning of LED Codes

1 Normal operation

1.1 No microSD card found

LEDs:

1: red

2:

3:

4:

5:

Meaning:

No microSD card found, or not formatted with FAT32.

1.2 Exception occurred

If an exception ocurred and it does not get resolved (caught), the number of the exception is output by a blink code.

LEDs:

1: red: goes out at the beginning of the blink code for a short time, and then goes on again

2: red: flashing, number corresponds to 1000s of the exception

3: red: flashing, number corresponds to 100s of the exception

4: red: flashing, number corresponds to 10s of the exception

5: red: flashing, number corresponds to 1s of the exception

Meaning:

Count how often the LEDs light up, and you'll get the code of the exception.

This code could originate from:

- an approprate "`throw`" instruction. Example:
```
if getSignature() <> sig_atmega32 !atMega32?
  throw 1234 !create exception 1234
endif
```
- a function / procedure unable to fulfill its task and creating an exception.

2 Updating roloFlash

Updating roloFlash is described in chapter V "Updating roloFlash".

2.1 Update in process

LEDs:

1: red

2: green \ quickly flashing alternately

3: green /

4:

5:

Meaning:

File RUN.BIN not found. Please follow the instructions in IV.1 "Preparing the microSD card at the PC (e.g. in the development department)".

2.2 Update finished successfully

LEDs:

1: green

2: green

3:

4:

5:

Meaning:

The update succeeded. After pulling roloFlash off the new firmware is available the next time you use roloFlash.

2.3 Update erroneous

LEDs:

1: red

2:

3: red

4:

5:

Meaning:

The update failed. The old firmware might still be available.

Possible remedy:

- Try update again.
- Update with a different firmware.

XI SPECIFICATIONS

Technical Data

- Supports Atmel AVR series controllers with ISP interface:
  - AT89
  - AT90
  - ATtiny (not ATtiny4/5/9/10)
  - ATmega
- Programming of the microcontroller via 6-pin ISP plug
- Power supply via microcontroller to be flashed (2,0 - 5,5 volts)
- Programming of:
  - Flash
  - EEPROM
  - Fuse bits (LO, HI, EXT)
  - Lock bits
- Supported file system: FAT32
- Supported file format: Intel HEX (".HEX")
- Supported memory card formats: microSD, microSDHC