

**halec**  
Herrnröther Str. 54  
63303 Dreieich  
Germany

[www.halec.de](http://www.halec.de)



# Handbuch

## roloFlash 2 AVR



Dokumentenversion 1.7.4 vom 2025-09-17  
(Stand der Firmware: ab v07.AA)

Copyright © 2009-2025 halec. Alle Marken, Logos und Bilder sind Eigentum der jeweiligen Hersteller bzw. Urheber. Änderungen und Irrtümer vorbehalten.

# Inhaltsverzeichnis

I	Vorwort.....	1
II	Verpackungsinhalt.....	3
III	Beschreibung.....	4
1	Universal-Connector (Programmier-Buchse).....	4
1.1	Pin-Belegungen (Überblick).....	4
1.2	Pin-Belegung Atmel ISP-Interface.....	5
1.3	Pin-Belegung Atmel TPI-Interface.....	5
1.4	Pin-Belegung Atmel PDI-Interface.....	6
1.5	Pin-Belegung Atmel UPDI-/aWire-Interface.....	7
2	Pullup- / Pulldown-Widerstände.....	7
3	Spannungsbereich.....	8
4	Elektrische Schutzmaßnahmen.....	8
5	LEDs.....	9
6	microSD-Kartenslot.....	9
7	Typischer Ablauf / Verwendung.....	9
7.1	Vorbereitung der microSD-Karte am PC.....	9
7.2	Flashen der Targetboards.....	11
IV	Aktualisieren von roloFlash.....	12
1	Aktualisieren des Bootloaders.....	12
2	Aktualisieren der Firmware.....	14
V	Liste der mitgelieferten roloBasic-Skripte.....	17
VI	roloFlash-API (Liste der Prozeduren und Funktionen).....	22
1	Interne Datenbank.....	23
1.1	db_getHandle.....	23
1.2	db_get.....	24
2	Busse.....	25
2.1	bus_open.....	26
2.2	bus_close.....	27
2.3	bus_setSpeed.....	28
2.4	bus_getSpeed.....	29
2.5	Atmel ISP-Bus.....	30
2.5.1	bus_open(ISP, ...) und verfügbare Geschwindigkeiten.....	30
2.5.2	Reset-Mode einstellen.....	33
2.6	Atmel TPI-Bus.....	35
2.6.1	bus_open(TPI, ...) und verfügbare Geschwindigkeiten.....	35
2.6.2	Reset-Mode einstellen.....	38
2.7	Atmel PDI-Bus.....	40
2.7.1	bus_open(PDI, ...) und verfügbare Geschwindigkeiten.....	40
2.8	Atmel UPDI-Bus / aWire-Bus.....	42
2.8.1	bus_open(UPDI / AWIRE, ...) und verfügbare Geschwindigkeiten.....	42
3	Target allgemein.....	45

3.1	target_open.....	45
3.2	target_close.....	46
3.3	target_getPresent.....	48
3.4	target_setMode.....	49
3.5	target_restart.....	52
3.6	Target-Memorymap lesen/schreiben.....	54
3.6.1	target_setMemoryMap.....	54
3.6.2	target_getMemoryMap.....	56
3.6.3	target_clearMemoryLayout.....	57
3.7	Target löschen, schreiben, lesen und verifizieren.....	58
3.7.1	target_erase.....	58
3.7.2	target_writeFromFile.....	60
3.7.3	target_readToFile.....	63
3.7.4	target_write.....	66
3.7.5	target_read.....	67
3.8	Target Atmel AVR (ISP-Interface).....	69
3.8.1	target_getDeviceId.....	69
3.8.2	target_readBits.....	70
3.8.3	target_writeBits.....	72
3.8.4	target_setExtendedAddressMode.....	74
3.9	Atmel TPI (TPI-Interface).....	75
3.9.1	target_getDeviceId.....	75
3.9.2	target_readBits.....	76
3.9.3	target_writeBits.....	77
3.10	Target Atmel PDI (PDI-Interface).....	78
3.10.1	target_getDeviceId.....	79
3.10.2	target_readBits.....	80
3.10.3	target_writeBits.....	81
3.11	Target Atmel UPDI (UPDI-Interface).....	82
3.11.1	target_getDeviceId.....	83
3.11.2	target_readBits.....	84
3.11.3	target_writeBits.....	85
3.12	Target Atmel AVR32 (aWire-Interface).....	87
3.12.1	target_getDeviceId.....	87
4	Flash-Data.....	88
4.1	fd_write.....	92
4.2	fd_createArray.....	94
4.3	fd_writeArrayElem.....	96
4.4	fd_writeSubArray.....	97
4.5	fd_read.....	99
4.6	fd_readArrayElem.....	100
4.7	fd_readSubArray.....	102
4.8	fd_remove.....	104
4.9	fd_getItemCount.....	105
4.10	fd_getId.....	105
4.11	fd_idExists.....	107
4.12	fd_isArray.....	108

4.13	fd_getArraySize.....	109
4.14	fd_getType.....	110
4.15	fd_getCountingBytes.....	111
4.16	fd_setCrc.....	112
4.17	fd_getCrc.....	113
4.18	fd_calcCrc.....	114
4.19	fd_hasCrc.....	115
4.20	fd_getFreeMem.....	117
4.21	fd_getBytesWritten.....	118
4.22	fd_setSingleBufferMode.....	118
4.23	fd_getSingleBufferMode.....	120
4.24	fd_cleanup.....	121
4.25	fd_format.....	121
5	Dateien.....	122
5.1	fs_mediaExists.....	124
5.2	fs_create.....	125
5.3	fs_rename.....	126
5.4	fs_remove.....	127
5.5	fs_mkdir.....	128
5.6	fs_fileExists.....	129
5.7	fs_fileSize.....	130
5.8	fs_open.....	131
5.9	fs_read.....	132
5.10	fs_write.....	133
5.11	fs_truncate.....	134
5.12	fs_close.....	135
5.13	fs_sync.....	136
6	LEDs.....	137
6.1	led_on.....	137
6.2	led_off.....	138
6.3	led_blink.....	139
6.4	led_runningLight.....	140
6.5	led_runningLightOutstanding.....	140
7	SecureApi.....	141
7.1	sec_crc.....	144
7.2	sec_hash.....	145
7.3	sec_encrypt.....	147
7.4	sec_decrypt.....	149
8	Abfrage von roloFlash-Eigenschaften.....	150
8.1	Versionsnummern etc. ....	150
8.2	sys_serialNumber.....	151
8.3	sys_uniqueld.....	151
9	Sonstige.....	152
9.1	sys_setLogMode.....	152
9.2	print.....	154
9.3	sprint.....	154
9.4	delay.....	155

9.5	sys_getSystemTime.....	156
9.6	getTargetBoardVoltage.....	157
9.7	sys_setCpuClock.....	157
9.8	sys_getCpuClock.....	158
9.9	sys_getEraseCounters.....	159
9.10	setBitBlock.....	161
9.11	getBitBlock.....	162
9.12	chain.....	163
VII	Exceptions.....	166
1	Exceptions des roloBasic.....	166
2	Exceptions des Dateisystems.....	167
3	Vom Benutzer ausgelöste Exceptions.....	168
4	Exceptions des roloFlash.....	169
VIII	Bedeutungen von LED-Codes.....	174
1	Normaler Betrieb.....	174
1.1	Keine microSD-Karte gefunden.....	174
1.2	Exception aufgetreten.....	174
2	roloFlash-Aktualisierung.....	175
2.1	Warten auf microSD-Karte für Aktualisierung.....	175
2.2	Aktualisierung läuft.....	176
2.3	Aktualisierung mit Erfolg abgeschlossen.....	176
2.4	Aktualisierung fehlerhaft: Dateifehler.....	176
2.5	Aktualisierung fehlerhaft: Datei nicht gefunden.....	177
2.6	Aktualisierung fehlerhaft: Mehrere Dateien gefunden.....	177
2.7	Aktualisierung fehlerhaft: Sonstiges.....	178
IX	Spezifikationen.....	179
1	Unterstützte Controller von Atmel.....	179
1.1	AVR (ISP-Interface).....	179
1.2	AVR (TPI-Interface).....	181
1.3	AVR (PDI-Interface).....	181
1.4	AVR (UPDI-Interface).....	182
1.5	AVR32 (aWire-Interface).....	182
2	Technische Daten.....	183



# I Vorwort

- Mit roloFlash können Sie mobil und unabhängig vom PC Ihre Produkte mit verschiedenen Mikrocontrollern flashen. Unter bestimmten Bedingungen können auch mehrere Mikrocontroller in Ihrem Produkt geflasht werden. Eine Liste der aktuell unterstützten Mikrocontroller finden Sie im Kapitel „Spezifikationen“.
- Anwenderfehler werden vermieden, da es keine Bedienelemente gibt. Dadurch ist es möglich, daß auch Kunden vor Ort ohne besondere Einweisung Software-Updates vornehmen können.
- Dazu sind kein PC und keine spezifischen Tool-Chains (z. B. von Mikrocontroller-Herstellern) nötig.
- Nutzen Sie roloFlash im Feldeinsatz, in Ihrem Kundenumfeld bzw. zur Serien- und Kleinserienfertigung.
- Gewinnen Sie Freiräume, indem Sie auf einen einheitlichen Prozeß für alle unterstützten Mikrocontroller-Familien zurückgreifen.

## **Begriff „Atmel“**

Die Firma Atmel wurde von Microchip übernommen. Es wird jedoch weiterhin der Name „Atmel“ verwendet (in Dokumentation und Software), um Verwechslungen mit anderen Controllern von Microchip (z.B. PIC-Familien) zu vermeiden.

## **Begriff „Targetboard“**

Unter „Targetboard“ verstehen wir Ihre zu flashenden Produkte. Die Produkte enthalten den bzw. die zu flashenden Mikrocontroller. Diesen Begriff verwenden wir von nun an regelmäßig in diesem Dokument.

## **Begriff „Target“**

Unter „Target“ verstehen wir den bzw. die zu flashenden Mikrocontroller (falls mehrere vorhanden sind, z.B. JTAG-Chain).

## **Begriff „zu flashende Mikrocontroller“**

Außer „Flashen“ können Sie Ihre Mikrocontroller (Target) auch auslesen (und z. B. als HEX-Datei speichern), verifizieren (z. B. gegen eine HEX-Datei), löschen oder modifizieren. Aus Gründen der Verständlichkeit wird oft nur das „Flashen“ erwähnt, ohne die anderen Möglichkeiten jedesmal zu wiederholen.

### **Zeichen „<“ und „>“**

Bei den Beschreibungen der Funktionen und Prozeduren werden die Parameter oft mit „<“ und „>“ eingerahmt. Dies soll symbolisieren, daß an dieser Stelle ein sinnvoller Wert (ohne die spitzen Klammern) verwendet werden soll:

Beispiel:

```
delay <duration>
```

Hier können Sie z. B.

```
delay 1000
```

schreiben.



## II Verpackungsinhalt

Bitte überprüfen Sie sorgfältig den Lieferumfang:

- roloFlash 2 AVR
- microSD-Karte
  - vorbereitet für den Einsatz in Ihrem roloFlash, mit Dokumentation, Beispielen, Firmware und roloBasic-Compiler
  - zum Einlegen in roloFlash

Hinweis: Die microSD-Karte befindet sich entweder im roloFlash eingesteckt oder liegt bei.

## III Beschreibung

### 1 Universal-Connector (Programmier-Buchse)

Der 6-polige Universal-Connector wird entweder auf einen passenden Stecker des zu programmierenden Targetboards gesteckt oder über einen passenden Adapter mit dem zu programmierenden Targetboard verbunden.

Sie finden auf der Vorderseite des roloFlash direkt über der Buchse eine Pin-1-Markierung.

Das Rastermaß der Buchse ist 2,54 mm (0,1 Zoll).

#### 1.1 Pin-Belegungen (Überblick)

Die Anschlüsse des roloFlash können je nach verwendetem Bus verschiedene Bedeutungen haben.

Die Belegung ist direkt passend für Atmel ISP, Atmel TPI, Atmel PDI, Atmel UPDI und Atmel aWire.

TPI	ISP	PDI	UPDI aWire			UPDI aWire	PDI	ISP	TPI
Signal				Pins		Signal			
DATA	MISO	DATA	DATA	1 ●	● 2	Vtgt			
CLK	SCK			3 ●	● 4			MOSI	
RST	RST	CLK		5 ●	● 6	GND			

Abbildung 1: Überblick über Targetboard-Steckerbelegungen in Draufsicht

#### Hinweis:

Es gibt Adapter, um die Pin-Belegung des roloFlash auf verschiedene übliche Programmierstecker-Belegungen anzupassen; diese Adapter werden bei den entsprechenden Bussen aufgelistet.

## 1.2 Pin-Belegung Atmel ISP-Interface

Wenn Sie das ISP-Interface benutzen, dann werden folgende Anschlüsse für den Bus verwendet:

Signal	Pin	Pin	Signal
MISO	1 ●	● 2	$V_{tgt}$
SCK	3 ●	● 4	MOSI
RST	5 ●	● 6	GND

Abbildung 2: Draufsicht auf ISP-Stecker eines Targetboards

Die Pin-Belegung ist direkt für den von Atmel verwendeten ISP-Programmierstecker passend, sie können roloFlash direkt und ohne Adapter aufstecken.

### Hinweis:

Es gibt folgende Adapter, um auf bestimmte übliche Programmierstecker-Belegungen zu adaptieren:

Bezeichnung	Pins	Reihen	Rastermaß [mm]
roloFlash-2-AVR-Target-Adapter 1:1 6p	6	2	2,54
roloFlash-2-AVR-Target-Adapter Atmel ISP/TPI 10p	10	2	2,54

## 1.3 Pin-Belegung Atmel TPI-Interface

Wenn Sie das TPI-Interface benutzen, dann werden folgende Anschlüsse für den Bus verwendet:

Signal	Pin	Pin	Signal
DATA	1 ●	● 2	$V_{tgt}$
CLK	3 ●	● 4	
RST	5 ●	● 6	GND

Abbildung 3: Draufsicht auf TPI-Stecker eines Targetboards

Die Pin-Belegung ist direkt für den von Atmel verwendeten TPI-Programmierstecker passend, sie können roloFlash direkt und ohne Adapter aufstecken.

#### Hinweis:

Es gibt folgende Adapter, um auf bestimmte übliche Programmierstecker-Belegungen zu adaptieren:

Bezeichnung	Pins	Reihen	Rastermaß [mm]
roloFlash-2-AVR-Target-Adapter 1:1 6p	6	2	2,54

## 1.4 Pin-Belegung Atmel PDI-Interface

Wenn Sie das PDI-Interface benutzen, dann werden folgende Anschlüsse für den Bus verwendet:

Signal	Pin	Pin	Signal
DATA	1 ●	● 2	V <sub>tgt</sub>
	3 ●	● 4	
CLK	5 ●	● 6	GND

Abbildung 4: Draufsicht auf PDI-Stecker eines Targetboards

Die Pin-Belegung ist direkt für den von Atmel verwendeten PDI-Programmierstecker passend, sie können roloFlash direkt und ohne Adapter aufstecken.

#### Hinweis:

Es gibt folgende Adapter, um auf bestimmte übliche Programmierstecker-Belegungen zu adaptieren:

Bezeichnung	Pins	Reihen	Rastermaß [mm]
roloFlash-2-AVR-Target-Adapter 1:1 6p	6	2	2,54

## 1.5 Pin-Belegung Atmel UPDI-/aWire-Interface

Wenn Sie das UPDI-/aWire-Interface benutzen, dann werden folgende Anschlüsse für den Bus verwendet:

Signal	Pin	Pin	Signal
DATA	1 ●	● 2	V <sub>tgt</sub>
	3 ●	● 4	
	5 ●	● 6	GND

Abbildung 5: Draufsicht auf UPDI-/aWire-Stecker eines Targetboards

Die Pin-Belegung ist direkt für den von Atmel verwendeten UPDI- bzw. aWire-Programmierstecker passend; sie können roloFlash direkt und ohne Adapter aufstecken.

#### Hinweis:

Es gibt folgende Adapter, um auf bestimmte übliche Programmierstecker-Belegungen zu adaptieren:

Bezeichnung	Pins	Reihen	Rastermaß [mm]
roloFlash-2-AVR-Target-Adapter 1:1 6p	6	2	2,54

## 2 Pullup- / Pulldown-Widerstände

Um auf allen Leitungen einen definierten Spannungspegel zu haben, besitzt roloFlash interne Pullup- und Pulldown-Widerstände:

Widerstand	Signal	Pin	Pin	Signal	Widerstand
Pullup 1 MΩ	MISO	1 ●	● 2	V <sub>tgt</sub>	-
Pulldown 1 MΩ	SCK	3 ●	● 4	MOSI	Pullup 1 MΩ
Pullup 1 MΩ	RST	5 ●	● 6	GND	-

Abbildung 6: Draufsicht auf passenden Stecker eines Targetboards

### 3 Spannungsbereich

roloFlash wird vom Targetboard aus über Pin 2 ( $V_{tgt}$ ) versorgt. Dabei paßt roloFlash alle Datenleitungen auf diese Spannung an.

*Spannungsbereich: 2,0 Volt - 5,5 Volt*

---

### 4 Elektrische Schutzmaßnahmen

roloFlash bietet folgende Schutzmaßnahmen:

- Bei Verpolung der Versorgungsspannung wird die Stromversorgungsverbindung über einen Transistor reversibel aufgetrennt.
- Bei Überspannung auf der Versorgungsspannung höher als 5,7 V schaltet eine Schutzschaltung über einen Transistor reversibel ab.
- Alle Datenleitungen sind mit Polyswitches reversibel gegen Überstrom abgesichert.
- Bei nicht standardkonformer Belegung von Pin 5 (GND) wird ein etwaiger Kurzschluß zum Pin 3 (GND) durch einen Polyswitch reversibel begrenzt.
- Alle Leitungen sind mit ESD-Schutzbauteilen ausgestattet, die IEC 61000-4-2 Level 4 (15 kV (air discharge) , 8 kV (contact discharge)) erfüllen.

Diese Maßnahmen bieten einen weitgehenden Schutz bei Fehlbedienungen wie verpoltes Aufstecken etc. Trotzdem ist nicht ausgeschlossen, daß bei Fehlbedienungen Schäden am Targetboard oder an roloFlash entstehen können.

---

### 5 LEDs

Fünf programmierbare zweifarbige (rote und grüne) LEDs. Mit den LEDs können Sie z. B.

- ein grünes Lauflicht laufen lassen, das den Flashvorgang darstellt.
- mit Rot Fehlermeldungen ausgeben.

## 6 microSD-Kartenslot

Für eine microSD- oder microSDHC-Karte, die das abzuarbeitende Skript (RUN\_V07.BIN) sowie die zu flashenden Dateien enthält.

Hinweis: Die Datei RUN\_V07.BIN kann auch in der Flash-Disk abgelegt sein. Falls die Datei dort vorhanden ist, dann ist sie gegenüber einer etwaigen Datei gleichen Namens auf der SD-Karte priorisiert.

---

## 7 Typischer Ablauf / Verwendung

Der übliche Ablauf bei der Verwendung von roloFlash gliedert sich in zwei Teile:

- Vorbereitung der microSD-Karte am PC (z. B. in der Entwicklung)
- Flashen der Targetboards (z. B. ungeschultes Personal in der Produktion, Kunde oder Techniker im Feldeinsatz)

### 7.1 Vorbereitung der microSD-Karte am PC

#### Z. B. in der Entwicklung

Maßgeblich ist immer die Datei „RUN\_V07.BIN“, die von roloFlash ausgewertet wird, um den darin kodierten Programmablauf abzuarbeiten. Der Zusatz „V07“ entspricht dem Major-Teil der Softwareversion des roloFlash.

Falls Sie eine microSD-Karte formatieren wollen, benutzen Sie dazu Windows 7 oder höher (Windows XP ist nicht geeignet).

- Sie erstellen den gewünschten Ablauf in roloBasic in der Datei „RUN\_V07.BAS“. Dazu können Sie ein Beispielskript verwenden oder anpassen. Im Kapitel „Spezifikationen“ finden Sie eine Liste der exakten Namen der bekannten Controller, die Ihnen roloFlash dazu zur Verfügung stellt. Ihre erzeugte Datei sollte die Dateiendung „.BAS“ haben, vorzugsweise benennen Sie diese „RUN\_V07.BAS“.
- Ihre roloBasic-Datei kann mit einer Magic-Cookie-Zeile anfangen. Es wird empfohlen folgende Magic-Zeile zu verwenden:

### #roloFlash 2, v07+

Falls ein Magic angegeben ist, dann muss der Anfang „**#roloFlash 2**“ lauten. Es kann, wie eben gezeigt, optional eine Versionsnummer (z.B. „**v07**“) angegeben werden, mit dem Zusatz „+“ werden auch später höhere Versionen von roloFlash die Datei akzeptieren. Andernfalls wird der Compiler das Übersetzen verweigern. Sie können auch die Verwendung des Skripts mit „**#roloFlash 2, v07.\***“ auf v07 einschränken. Die Versionsnummer entspricht dem Major-Teil der Softwareversion des roloFlash.

- Varianten roloFlash 2 und roloFlash 2 AVR: Die Basic-Dateien sind auf allen Varianten ausführbar. Es gibt keinen separaten Compiler. Die Magic-Cookie-Zeile ist identisch.
- Ihr Skript kann dabei auf übliche „.HEX“-Dateien (Intel-HEX-Format: „I8HEX“, „I16HEX“ und „I32HEX“), auf Motorola S-Record/SREC/S19/S28/S37, auf „.ELF“-Dateien oder auf „.RAW“-Dateien verweisen, die auf das Targetboard geflasht werden sollen.
- Bitte rufen sie anschließend auf dem PC den mitgelieferten Compiler „**rbv07.exe**“ auf. Wenn Sie ihn ohne Parameter aufrufen, dann übersetzt er als Default die Datei „**RUN\_V07.BAS**“ in die Image-Datei „**RUN\_V07.BIN**“.

Sie können die Dateien mit den Skripten („.BAS“), die kompilierten Dateien („.BIN“) und den Compiler nach eigenem Ermessen auf dem PC und/oder auf der microSD-Karte speichern. Zum Ausführen auf dem roloFlash sind lediglich die Datei „**RUN\_V07.BIN**“ sowie etwaige durch den Code referenzierten Dateien auf der microSD-Karte relevant.

**Hinweis:** Die roloFlash-2-Familie mit einer Firmware vor V05.AA verarbeitet immer die Datei „**RUN.BIN**“. Ab Version V05.AA wird die Majorversionsnummer mit in den Dateinamen aufgenommen, also „**RUN\_V05.BIN**“ bzw. „**RUN\_V07.BIN**“.

Dadurch ist es möglich, mehrere „**RUN\_Vxx.BIN**“ auf der microSD-Karte abzulegen, so daß diese Karte anschließend mit roloFlash mit verschiedenen Firmwareversionen ab V05.AA benutzt werden kann. Der jeweilig verwendete roloFlash wählt die zu seiner Firmware passende „**RUN\_Vxx.BIN**“-Datei aus.



## 7.2 Flashen der Targetboards

### **Z. B. ungeschultes Personal in der Produktion**

Hier ist der Ablauf denkbar einfach:

- Targetboard mit Energie versorgen.
- roloFlash auf den passenden Stecker des Targetboards aufstecken oder die Verbindung mit einem Adapter herstellen.
- roloFlash wird vom Targetboard mit Energie versorgt und beginnt automatisch mit der Abarbeitung der Datei „RUN\_V07.BIN“. Hierdurch wird üblicherweise das Flashen vorgenommen. Währenddessen kann z. B. ein grünes Lauflicht den Flashvorgang signalisieren.
- Wenn die RUN\_V07.BIN abgearbeitet wurde, was üblicherweise durch eine grün leuchtende LED 5 angezeigt wird, roloFlash abziehen – fertig.

# IV Aktualisieren von roloFlash

## 1 Aktualisieren des Bootloaders

Mit der Firmware-Version mit der Major-Version 7 wurde ein neuer Bootloader eingeführt.

**Wenn roloFlash erstmalig von einer Firmware-Version vor v07.AA auf eine aktuelle Version aktualisiert werden soll, muss vorab der Bootloader aktualisiert werden. Ansonsten ist dieses Kapitel irrelevant.**

Sie können anschließend bei Bedarf auch mit dem neuen Bootloader auf alte Firmware-Versionen zurückkehren, falls das erforderlich sein sollte.

Zur Aktualisierung des Bootloaders dient eine **Bootloader-Update-Firmware** mit dem Dateinamen:

- RF2\_06Z\_.HMP für roloFlash 2
- RF2A06Z\_.HMP für roloFlash 2 AVR

Diese ist aus Sicht des alten Bootloaders eine normale Firmware und kann wie gewohnt aufgebracht werden.

**Das Update besteht aus zwei Schritten, die unbedingt beide in dieser Reihenfolge ausgeführt werden müssen:**

- 1. Aufbringen der Bootloader-Update-Firmware**
- 2. Aktualisieren des Bootloaders**

### **1. Aufbringen der Bootloader-Update-Firmware**

- Es muß sich exakt eine Firmware-Datei (RF2\_V06Z\_.HMP oder RF2AV06Z\_.HMP) im Hauptverzeichnis der microSD-Karte befinden. Sind mehrere Firmware-Dateien vorhanden, wird der Vorgang nicht gestartet.
- Der Vorgang wird ausgelöst, wenn der roloFlash **ohne** microSD-Karte auf ein beliebiges Targetboard aufgesteckt und **anschließend** die microSD-Karte eingesteckt wird.
- Das Targetboard dient dabei nur zur Energieversorgung.

- Der Vorgang wird mittels der LEDs angezeigt, siehe Kapitel „roloFlash-Aktualisierung“.
- Solange die microSD-Karte noch nicht eingesteckt ist, leuchtet LED 1 rot.
- Während der Aktualisierung blinken LED 2 und LED 3 im Wechsel grün.  
**roloFlash sollte jetzt nicht abgezogen werden.**  
Falls roloFlash doch abgezogen worden sein sollte, kann es sein, daß die Firmware defekt ist. In diesem Zustand sollte roloFlash von selbst auf einer erneuten Aktualisierung bestehen.  
D. h. Bei der nächsten Energieversorgung wartet roloFlash solange, bis durch das Einschieben der microSD-Karte eine neue Firmware zur Verfügung steht. Diese wird erneut geflasht.  
**Falls eine Aktualisierung unterbrochen wurde, führen Sie auf jeden Fall eine erneute Aktualisierung durch, auch wenn Sie den Eindruck haben, daß die Aktualisierung eventuell doch erfolgreich war.**
- Bei Erfolg leuchten anschließend LED 1 und LED 2 grün.
- roloFlash bleibt in diesem Zustand, bis er abgezogen wird. Bitte ziehen Sie roloFlash nun ab.
- Bitte löschen Sie die Datei für die Bootloader-Update-Firmware auf der microSD-Karte.

## 2. Aktualisieren des Bootloaders

- Bitte stecken Sie roloFlash mit eingelegter microSD-Karte auf das Target erneut auf (die microSD-Karte kann leer sein).
- Nun startet die eigentliche Aktualisierung des Bootloaders. LED 2 leuchtet rot.
- Während der Aktualisierung blinken LED 3 und LED 4 im Wechsel grün.  
**roloFlash darf jetzt auf keinen Fall abgezogen werden.**  
**Falls roloFlash doch abgezogen worden sein sollte, kann es sein, daß der Bootloader defekt ist. Dieses kann nur noch durch Einschieben an den Hersteller repariert werden.**
- Bei Erfolg leuchten anschließend LED 2 rot und LED 5 grün.
- roloFlash bleibt in diesem Zustand, bis er abgezogen wird. Bitte ziehen Sie roloFlash nun ab.
- Ab dem nächsten Einstecken läuft roloFlash mit dem aktualisierten Bootloader. Sie sollten jetzt direkt mit dem Aktualisieren der Firmware fortfahren (nächstes Kapitel).

---

## 2 Aktualisieren der Firmware

roloFlash verfügt selbst über eine eigene Firmware, die aktualisiert werden kann.

### Versionsnummern

Die Versionsnummer setzt sich aus `major` und `minor` zusammen:

- `major`:  
Major wird angepaßt wenn:
  - sich die roloBasic-Schnittstelle ändert.
- `minor`:  
Minor wird angepaßt bei Änderungen, die nicht die roloBasic-Schnittstelle betreffen, z. B. bei:
  - Beseitigen von Bugs
  - Hinzufügen von Einträgen in die Target-Datenbank
  - Geschwindigkeits-Optimierungen

Daraus folgt, daß solange `major` nicht geändert wurde, auch kein Update des roloBasic-Compilers notwendig ist und bereits kompilierte `RUN_V07.BIN`-Dateien gültig bleiben.

### Dateinamen für das Firmware-Update

Der Dateiname für das Firmware-Update hält sich an die beim FAT-Dateisystem übliche 8.3-Namens-Konvention und ist wie folgt aufgebaut:

`RF2Aaabb.HMP` mit:

- **aa** = major (als Zahl, z.B. „01“)
- **bb** = minor (als Buchstaben, z.B. „AA“)

### Starten der Aktualisierung

- Trennen Sie den roloFlash von einem Targetboard. Kopieren Sie die gewünschte Firmware auf die microSD-Karte und stecken Sie diese in den roloFlash. Zum Aktualisieren muß sich exakt eine Firmware-Datei im Hauptverzeichnis der microSD-Karte befinden. Sind mehrere Dateien vorhanden, wird die Aktualisierung nicht gestartet.
- Verbinden Sie roloFlash mit einem Targetboard zur Stromversorgung. Die Aktualisierung beginnt und wird durch LED-Muster angezeigt (siehe nächstes Kapitel „Der Vorgang des Aktualisierens“).
- Es erfolgt keine Überprüfung, ob die Firmware auf der microSD-Karte neuer oder älter ist. Damit ist es auch möglich, zu einer älteren Version zurückzukehren, falls das erforderlich sein sollte.
- Nach erfolgreicher Aktualisierung entfernen Sie bitte die Firmware von der microSD-Karte um eine erneute Aktualisierung zu vermeiden.

Hinweis: Bei Auslieferung befindet sich die aktuelle Version in einem Unterverzeichnis. Die Datei wird erst dann zum Flashen herangezogen, wenn sie in das Hauptverzeichnis der microSD-Karte verschoben bzw. kopiert wurde.

#### **Der Vorgang des Aktualisierens**

- Das Targetboard dient dabei nur zur Energieversorgung.
- Der Vorgang wird mittels der LEDs angezeigt, siehe Kapitel „roloFlash-Aktualisierung“. Insbesondere leuchtet während und nach der Aktualisierung die LED 1 rot.
- Während der Aktualisierung leuchtet LED 1 ständig rot. LED 2 und LED 3 leuchten im Wechsel mehrere Sekunden grün. **roloFlash sollte jetzt nicht abgezogen werden.**  
Falls roloFlash doch abgezogen worden sein sollte, kann es sein, daß die Firmware defekt ist. In diesem Zustand sollte roloFlash von selbst auf einer erneuten Aktualisierung bestehen.  
D. h. Bei der nächsten Energieversorgung wartet roloFlash solange, bis durch das Einschieben der microSD-Karte eine neue Firmware zur Verfügung steht. Diese wird erneut geflasht.  
**Falls eine Aktualisierung unterbrochen wurde, führen Sie auf jeden Fall eine erneute Aktualisierung durch, auch wenn Sie den Eindruck haben, daß die Aktualisierung eventuell doch erfolgreich war.**
- Bei Erfolg bleibt LED 1 rot, LED 5 leuchtet grün.

- roloFlash bleibt in diesem Zustand, bis er abgezogen wird. Bitte ziehen Sie roloFlash nun ab.
- Bitte löschen Sie nun die Firmware-Datei auf der SD-Karte.
- Ab dem nächsten Einstecken läuft roloFlash mit der aktualisierten Firmware.

Falls die Aktualisierung nicht erfolgreich gewesen sein sollte, verwenden Sie bitte eine frisch unter Windows 7 oder höher mit FAT32 formatierte microSD-Karte, auf der sich ausschließlich die Datei für die Firmware-Aktualisierung befindet.

**Hinweis:**

Für eine Produktion oder die Weitergabe des roloFlash an Ihre Kunden wird empfohlen, keine Datei für eine Firmware-Aktualisierung auf der microSD-Karte zu belassen.

## V Liste der mitgelieferten roloBasic-Skripte

- **"Hello world"**

- `scripts\hello-world\RUN_V07.BAS`
- Zusätzlich befindet sich dieses Skript und die kompilierte `RUN_V07.BIN` bei Auslieferung im Hauptverzeichnis der SD-Karte.

**Vorbereitung:**

- Zum Verwenden kopieren Sie bitte das Skript als `RUN_V07.BAS` in das Hauptverzeichnis der microSD-Karte.
- Starten Sie den Compiler mittels „`compile_V07.bat`“, um aus der `RUN_V07.BAS` die benötigte `RUN_V07.BIN` zu erzeugen.

**Funktion:**

- Löscht eine eventuell vorhandene vorherige `LOG.TXT`-Datei.
- Schreibt in die `LOG.TXT`-Datei einen Text, unter anderem „Hello world“.
- Startet ein grünes Lauflicht von LED 1 zur LED 4 für 3 Sekunden.
- Startet ein rotes Lauflicht von LED 1 zur LED 4 für 3 Sekunden.
- Startet ein grünes Lauflicht von LED 4 zur LED 1 für 3 Sekunden.
- Startet ein rotes Lauflicht von LED 4 zur LED 1 für 3 Sekunden.
- Zum Abschluss leuchtet LED 5 grün.

- **"Versions"**

- `scripts\versions\RUN_V07.BAS`

**Vorbereitung:**

- Zum Verwenden kopieren Sie bitte das Skript als `RUN_V07.BAS` in das Hauptverzeichnis der microSD-Karte.

- Starten Sie den Compiler mittels „`compile_V07.bat`“, um aus der `RUN_V07.BAS` die benötigte `RUN_V07.BIN` zu erzeugen.

**Funktion:**

- Löscht eine eventuell vorhandene vorherige `LOG.TXT`-Datei.
- Schreibt in die `LOG.TXT`-Datei Versionsnummern etc. des roloFlash:
  - Company Name
  - Device name
  - Software Version
  - Hardware Version
  - Bootloader Version
  - Image Version
- Zum Abschluß leuchtet LED 5 grün.

- **"Erase-and-Flash"**

- `scripts\Microchip_Atme\AVR\ISP\erase-and-flash\RUN_V07.BAS`
- `scripts\Microchip_Atme\AVR\TPI\erase-and-flash\RUN_V07.BAS`
- `scripts\Microchip_Atme\AVR\PDI\erase-and-flash\RUN_V07.BAS`
- `scripts\Microchip_Atme\AVR\UPDI\erase-and-flash\RUN_V07.BAS`
- `scripts\Microchip_Atme\AVR32\Wire\erase-and-flash\RUN_V07.BAS`

**Vorbereitung:**

- Das Skript gibt es jeweils in einer Version für Atmel ISP-, TPI-, PDI- und UPDI-Controller.
- Zum Verwenden kopieren Sie bitte die passende Version als `RUN_V07.BAS` in das Hauptverzeichnis der microSD-Karte.
- Passen Sie bitte anschließend in der Datei den Namen Ihres Targets und die Dateinamen der HEX-Datei an. Zusätzlich zur Angabe einer HEX-Datei für den Flashspeicher können Sie auch eine weitere HEX-Datei für das EEPROM angeben.
- Optional können Sie auch die Busgeschwindigkeit sowie die Geschwindigkeit des roloFlash anpassen.



- Starten Sie den Compiler mittels „compile\_V07.bat“, um aus der RUN\_V07.BAS die benötigte RUN\_V07.BIN zu erzeugen.

#### **Funktion:**

- Startet ein Lauflicht von LED 1 zu LED 4, um einen Flash-Vorgang darzustellen.
- Löscht eine eventuell vorhandene vorherige LOG.TXT-Datei.
- Öffnet den jeweiligen Bus zum Target.
- Liest aus der internen Datenbank des roloFlash spezifische Informationen für den von Ihnen angegebenen Controller aus, darunter die ID in Form einer Signature bzw. einer Device-ID (bei Atmel ISP / TPI / PDI / UPDI), sowie andere für das Flashen benötigte Parameter.
- Liest die ID des angeschlossenen Targets aus und vergleicht diese mit den Werten aus der Datenbank.
- Wenn die ID nicht stimmen sollte (z. B. anderer Controller), dann wird der weitere Ablauf mit Ausgabe einer Fehlermeldung abgebrochen
- Löscht das Target (erase).
- Wenn von Ihnen angegeben: Ihre HEX-Datei wird in das Flash des Targets geschrieben und verifiziert.
- Wenn von Ihnen angegeben: Ihre HEX-Datei wird in das EEPROM des Targets geschrieben und verifiziert.
- Währenddessen läuft ein grünes Lauflicht, am Ende bleibt bei Erfolg LED 5 auf Grün.
- Schreibt die Ergebnisse ins Log-File (LOG.TXT)

- **"Read"**

- scripts\Microchip\_Atmel\AVR\ISP\read\RUN\_V07.BAS
- scripts\Microchip\_Atmel\AVR\TPI\read\RUN\_V07.BAS
- scripts\Microchip\_Atmel\AVR\PDI\read\RUN\_V07.BAS
- scripts\Microchip\_Atmel\AVR\UPDI\read\RUN\_V07.BAS
- scripts\Microchip\_Atmel\AVR\Wire\read\RUN\_V07.BAS

#### **Vorbereitung:**

- Das Skript gibt es jeweils in einer Version für Atmel ISP-, TPI-, PDI- und UPDI-Controller.
- Zum Verwenden kopieren Sie bitte die passende Version als `RUN_V07.BAS` in das Hauptverzeichnis der microSD-Karte.
- Passen Sie bitte anschließend in der Datei den Namen Ihres Targets und die Dateinamen der HEX-Datei an. Zusätzlich zur Angabe einer HEX-Datei für den Flashspeicher können Sie auch eine weitere HEX-Datei für das EEPROM angeben.
- Optional können Sie auch die Busgeschwindigkeit sowie die Geschwindigkeit des roloFlash anpassen.
- Starten Sie den Compiler mittels „`compile_V07.bat`“, um aus der `RUN_V07.BAS` die benötigte `RUN_V07.BIN` zu erzeugen.

#### **Funktion:**

- Startet ein Lauflicht von LED 4 zu LED 1, um einen Lese-Vorgang darzustellen.
- Löscht eine eventuell vorhandene vorherige `LOG.TXT`-Datei.
- Öffnet den jeweiligen Bus zum Target.
- Liest aus der internen Datenbank des roloFlash spezifische Informationen für den von Ihnen angegebenen Controller, darunter die ID in Form einer Signature bzw. einer Device-ID (bei Atmel ISP / TPI / PDI / UPDI), sowie andere für das Lesen benötigte Parameter (Größe des Speichers), aus.
- Liest die ID des angeschlossenen Targets aus und vergleicht diese mit den Werten aus der Datenbank.
- Wenn die ID nicht stimmen sollte (z. B. anderer Controller), dann wird der weitere Ablauf mit Ausgabe einer Fehlermeldung abgebrochen
- Wenn von Ihnen angegeben: Das Flash des Targets wird komplett ausgelesen und in die von Ihnen angegebene HEX-Datei geschrieben.
- Wenn von Ihnen angegeben: Das EEPROM des Targets wird komplett ausgelesen und in die von Ihnen angegebene HEX-Datei geschrieben.
- Währenddessen läuft ein grünes Lauflicht, am Ende bleibt bei Erfolg LED 5 auf Grün.
- Schreibt die Ergebnisse ins Log-File (`LOG.TXT`)



## VI roloFlash-API (Liste der Prozeduren und Funktionen)

Mit API (Application Programming Interface) ist die Schnittstelle gemeint, durch die roloBasic Zugriff auf alle roloFlash-spezifischen Prozeduren und Funktionen erlangt.

### **Prozeduren:**

Prozeduren haben keinen Rückgabewert. Die übergebenen Parameter müssen ohne Klammern angegeben werden.

Beispiel:

```
delay 1000
```

### **Funktionen:**

Funktionen haben einen Rückgabewert. Die übergebenen Parameter müssen in Klammern gesetzt werden.

Beispiel:

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

Hat die Funktion keine Parameter, dann können die Klammern weggelassen werden.

Beispiel:

```
value = getTargetBoardVoltage
```

oder

```
value = getTargetBoardVoltage()
```

### **Groß-/Kleinschreibung:**

Für roloBasic ist es unerheblich, ob Groß- oder Kleinschreibung verwendet wird, zur besseren Lesbarkeit werden allerdings folgende Konventionen verwendet:

- Zusammensetzung von mehreren Wörtern in Namen von Funktionen, Prozeduren und Variablen: der erste Buchstabe des Namens (und des ersten Wortes nach einem Unterstrich ("\_")) ist ein Kleinbuchstabe, jedes weitere Wort fängt mit einem Großbuchstaben an. Beispiel:  
`loaderUsed = target_getLoaderUsage(targetHandle)`
- Konstanten werden komplett groß geschrieben, Beispiel:  
`target_writeFromFile targetHandle, 0, fileName, HEX, FLASH, WRITEVERIFY`

---

## 1 Interne Datenbank

In roloFlash ist eine Datenbank integriert, die zu vielen Targets Informationen enthält. Die Informationen dienen folgenden Zwecken:

- Um in roloBasic zu prüfen, ob das gewünschte Target wirklich angeschlossen ist (z. B. Atmel ISP-Signature oder Atmel PDI-Device-ID).
- Um für das Flashen benötigte Daten zur Verfügung zu stellen.

Über den gewünschten Namen des Controllers kann man von der Datenbank ein Handle bekommen und mit diesem die weiteren Informationen abfragen. Das Handle muß anschließend nicht geschlossen werden.

### 1.1 db\_getHandle

Unter Angabe eines Target-Namens kann ein dazu passendes Datenbank-Handle ermittelt werden.

```
dbHandle = db_getHandle(<name>)
```

#### Vorbedingung:

- keine

#### Parameter:

**name**

Name des Targets. Es kann sein, daß der Name in der Datenbank verkürzt abgespeichert ist. Das ist dann der Fall, falls es mehrere Targets gibt, die sich z.B. nur in der Gehäuseform unterscheiden und ansonsten gleiche Parameter haben. Bitte schauen Sie für Ihren Controller im Kapitel „**Spezifikationen**“ nach, wie die korrekte Schreibweise ist. Bitte achten Sie auch auf die dort verwendete Groß-/Kleinschreibung.

**Rückgabewert:**

- ein Datenbank-Handle. Dieses kann benutzt werden, um mittels `db_get` Informationen zu diesem Target abzufragen.

**Exceptions:**

`unknownTarget`  
`apiTypeFault`

Target nicht bekannt  
Unzulässiger Typ für name

## 1.2 `db_get`

Unter Angabe eines bereits mittels `db_getHandle` erhaltenden Handles können weitere Informationen abgefragt werden.

`Value = db_get(<dbHandle>, <property>)`

**Vorbedingung:**

- gültiges `dbHandle`

**Parameter:**

**`dbHandle`**

Handle zum Zugriff auf die Datenbank, siehe `db_getHandle`

**`property`**

Angabe, welche Information ermittelt werden soll. Es stehen nicht für alle `dbHandles` alle Properties zur Verfügung. In dem Fall, daß die Information nicht ermittelt werden kann, wird eine Exception erzeugt.

Mögliche Werte für property sind:

**DB\_NAME:** Name des Targets. (Dieser kann kürzer sein als der Name, der zur Ermittlung des dbHandles angegeben wurde)

**DB\_FAMILY:** Ein Wert, der die Zugehörigkeit zu einer bestimmten Familie angibt. Dieser Wert wird zum Erhalten eines Target-Handles (siehe target\_open) benötigt.

**DB\_FLASHSIZE:** Angaben über die Größe des Flashs in Bytes.

**DB\_FLASHPAGESIZE:** Angabe einer Page-Größe in Bytes für das Schreiben von Speicher mit bestimmten Page-Größen (z. B. Atmel AVR und Atmel Xmega)

**DB\_EEPROMSIZE:** Angaben über die Größe des EEPROMs in Bytes.

**DB\_EEPROMPAGESIZE:** Angabe einer Page-Größe in Bytes für das Schreiben von EEPROM mit bestimmten Page-Größen (z. B. Atmel Xmega)

**DB\_DEVICEID:** Angabe der Device ID bzw. Signature (z. B. Atmel) (Array mit 3 Bytes).

### **Rückgabewert:**

- der abgefragte Wert für das Property

### **Exceptions:**

propertyNotFound

Der gewünschte Wert ist nicht bekannt oder existiert nicht (z.B. DB\_SIGNATURE bei Atmel PDI-Targets)

apiTypeFault

Unzulässiger Typ für dbHandle oder property

---

## **2 Busse**

Bei roloFlash wird grundsätzlich jede Schnittstelle, über die ein Target geflasht werden kann, als Bus aufgefaßt.

Dieses gilt auch, wenn an dieser Schnittstelle prinzipbedingt nur ein einziger Mikrocontroller angeschlossen sein kann (z. B. wird die ISP-Schnittstelle für Atmel AVR als Bus aufgefaßt).

- Grundsätzlich muß ein Bus erst geöffnet werden.

- Beim Öffnen wird geprüft, ob der Bus zur Verfügung steht. Sollte der Bus schon geöffnet sein, wird eine Exception erzeugt (resourceUnavailable). Die gleiche Exception erhalten Sie, falls Sie schon einen anderen Bus geöffnet haben und die Signale oder interne Ressourcen sich überschneiden würden.
- Ein an einem Bus angeschlossener Mikrocontroller (Target) kann erst angesprochen werden, wenn man von diesem Bus ein Target-Handle erhalten hat.
- Die Verbindung zu einem Target-Handle kann auch wieder geschlossen werden.
- Ein Bus kann auch geschlossen werden. In diesem Fall werden die betroffenen Leitungen wieder hochohmig.

## 2.1 bus\_open

```
busHandle = bus_open(<busType>, <index>, <speed>...)
```

Öffnet den entsprechenden Bus <busType> und stellt ein busHandle zur Verfügung. Je nach Bus können hierbei Leitungen initialisiert werden.

Es kann je nach verwendeten Bus weitere Parameter geben. In der Regel wird eine Busgeschwindigkeit angegeben, bei Abweichungen davon finden Sie die entsprechende Funktion im Unterkapitel für den jeweiligen Bus.

### Vorbedingung:

- keine

### Parameter:

#### **bus**

Gibt an, was für ein Typ Bus geöffnet werden soll. Die verfügbaren Busse sind:

- ISP
- PDI
- UPDI
- TPI

#### **index**



Gibt an, der wievielte Bus geöffnet werden soll. Der erste Bus hat den Index 0.

### **speed**

Die Geschwindigkeit des Busses, Angabe in Hz. Die unterstützten Geschwindigkeiten sind von der CPU-Clock (`sys_setCpuClock`) des roloFlash abhängig. Verfügbare Geschwindigkeiten finden Sie im jeweiligen Unterkapitel für den verwendeten Bus.

Falls die angegebene Frequenz nicht unterstützt wird, dann wird intern auf den nächsten möglichen Wert abgerundet.

### **Rückgabewert:**

- ein `busHandle`. Dieses kann benutzt werden, um weitere Funktionen wie z.B. `target_open` aufzurufen.

### **Exceptions:**

`apiValueRange`

`apiTypeFault`

`resourceUnavailable`

Unzulässiger Wert für `index` oder `speed`

Unzulässiger Typ für `index` oder `speed`

Der Bus kann nicht geöffnet werden. Mögliche Gründe:

- Der Bus wurde bereits geöffnet
- ein anderer Bus wurde geöffnet, und das gleichzeitige Öffnen ist nicht möglich

## **2.2 bus\_close**

`bus_close <busHandle>`

Schließt den entsprechenden Bus. Die betroffenen Leitungen werden dabei abgeschaltet.

Sollten auf dem Bus noch geöffnete Targets vorhanden sein, dann werden diese abgetrennt und die Target-Handles ungültig.

### **Vorbedingung:**

- gültiges `BusHandle`

**Parameter:**

**busHandle**

Das BusHandle auf den geöffneten Bus.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

invalidHandle  
apiTypeFault

Handle ist schon geschlossen  
Unzulässiger Typ für busHandle

## 2.3 bus\_setSpeed

bus\_setSpeed(<busHandle>, <speed>)

Ändert bei einem bereits geöffneten Bus die Busgeschwindigkeit. Die maximale Busgeschwindigkeit wird auf „speed“ begrenzt. Falls an diesem Bus ein Target angeschlossen ist, dann ergibt sich hieraus die Programmiertgeschwindigkeit für das Target.

**Vorbedingung:**

- gültiges busHandle

**Parameter:**

**busHandle**

Das von bus\_open erhaltene Bus-Handle.

**speed**

Die Geschwindigkeit des Busses, Angabe in Hz. Die unterstützten Geschwindigkeiten sind von der CPU-Clock (`sys_setCpuClock`) des roloFlash abhängig. Verfügbare Geschwindigkeiten finden Sie im jeweiligen Unterkapitel für den verwendeten Bus.

Falls die angegebene Frequenz nicht unterstützt wird, dann wird intern auf den nächsten möglichen Wert abgerundet.

**Hinweis:**

Falls Sie die Schnittstelle schon geöffnet haben und dann mittels `sys_setCpuClock` den Takt des roloFlash ändern, dann ändert sich auch die Geschwindigkeit des Busses. Daher wird empfohlen:

- Verwenden Sie `sys_setCpuClock` zuerst und öffnen dann erst den Bus.
- Oder setzen Sie nach `sys_setCpuClock` die Busgeschwindigkeit erneut mittels `bus_setSpeed`.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

`apiValueRange`  
`apiTypeFault`

Unzulässiger Wert für `speed`.  
Unzulässiger Typ für `busHandle` oder `speed`

## 2.4 `bus_getSpeed`

`Speed = bus_getSpeed(<busHandle>)`

Fragt bei einem bereits geöffneten Bus die aktuelle Busgeschwindigkeit ab. Diese kann gleich oder geringer sein als die bei `bus_open` bzw. `bus_setSpeed` angegebene Busgeschwindigkeit.

**Vorbedingung:**

- gültiges busHandle

**Parameter:**

**busHandle**

Das von bus\_open erhaltene Bus-Handle.

**Rückgabewert:**

- Busgeschwindigkeit in Hz

**Exceptions:**

apiTypeFault

Unzulässiger Typ für busHandle

## 2.5 Atmel ISP-Bus

Allgemeine Informationen zu Bussen finden Sie im übergeordneten Kapitel. Hier wird darauf aufbauend auf das spezifische Verhalten bei dem ISP-Bus eingegangen.

### 2.5.1 bus\_open(ISP, ...) und verfügbare Geschwindigkeiten

busHandle = bus\_open(ISP, <indexOfBus>, <speed>)

Öffnet den ISP-Bus und initialisiert die Leitungen. Die maximale Busgeschwindigkeit wird auf „speed“ begrenzt. Setzt die Programmiergeschwindigkeit für das Target.

**Vorbedingung:**

- keine

**Parameter:**

**busType**

ISP für IPS-Bus.

## indexOfBus

Muß 0 sein (es gibt jeweils nur einen Bus).

## speed

Die Geschwindigkeit des Busses, Angabe in Hz. Die unterstützten Busgeschwindigkeiten sind von der CPU-Clock (sys\_setCpuClock) des roloFlash abhängig.

Bei maximaler CPU-Clock = 120 Mhz werden die folgenden Busgeschwindigkeiten unterstützt:

15000000	7500000	5000000	3750000	3000000
2500000	2142857	1875000	1666666	1500000
1363636	1250000	1153846	1071428	1000000
937500	882352	833333	789473	750000
714285	681818	652173	625000	600000
576923	555555	535714	517241	500000
483870	468750	454545	441176	428571
416666	405405	394736	384615	375000
365853	357142	348837	340909	333333
326086	319148	312500	306122	300000
294117	288461	283018	277777	272727
267857	263157	258620	254237	250000
245901	241935	238095	234375	230769
227272	223880	220588	217391	214285
211267	208333	205479	202702	200000
197368	194805	192307	189873	187500
185185	182926	180722	178571	176470
174418	172413	170454	168539	166666
164835	163043	161290	159574	157894
156250	154639	153061	151515	150000
148514	147058	145631	144230	142857
141509	140186	138888	137614	136363
135135	133928	132743	131578	130434
129310	128205	127118	126050	125000
123966	122950	120967	119047	117187
115384	113636	111940	110294	108695
107142	105633	104166	102739	101351
100000	98684	97402	96153	94936
93750	92592	91463	90361	89285
88235	87209	86206	84745	83333

81967	80645	79365	78125	76923
75757	74626	73529	72463	71428
70422	69124	67873	66666	65502
64377	63291	62240	61224	60000
58823	57692	56603	55555	54545
53380	52264	51194	50167	49019
47923	46875	45871	44776	43731
42613	41551	40540	39473	38461
37406	36319	35294	34246	33185
32119	31055	30000	28957	27932
26929	25906	24875	23847	22831
21802	20775	19762	18750	17730
16722	15706	14705	13698	12690
11682	10676	9671	8670	7668
6666	5664	4662	3661	2660
1659				

Bei minimaler CpuClock = 24 Mhz werden die folgenden Busgeschwindigkeiten unterstützt:

1500000	750000	500000	375000	300000
250000	214285	187500	166666	150000
136363	125000	115384	107142	100000
93750	88235	83333	78947	75000
71428	68181	65217	62500	60000
57692	55555	53571	51724	50000
48387	46875	45454	44117	42857
41666	40540	39473	38461	36585
34883	33333	31914	30612	29411
28301	27272	25862	24590	23437
22388	21126	20000	18987	17857
16853	15789	14705	13636	12605
11538	10489	9433	8426	7425
6410	5395	4385	3378	2377
1376				

Falls die angegebene Frequenz nicht unterstützt wird, dann wird intern auf den nächsten möglichen Wert abgerundet.

### **Hinweis:**

Falls Sie die Schnittstelle schon geöffnet haben und dann mittels `sys_setCpuClock` den Takt des roloFlashs ändern, dann ändert sich auch die Geschwindigkeit des Busses. Daher wird empfohlen:

- Verwenden Sie `sys_setCpuClock` zuerst und öffnen dann erst den Bus.
- Oder setzen Sie nach `sys_setCpuClock` die Busgeschwindigkeit erneut mittels `bus_setSpeed`.

### **Rückgabewert:**

- ein `busHandle`. Dieses kann benutzt werden, um weitere Funktionen wie z.B. `getTargetPresent` aufzurufen.

### **Exceptions:**

<code>apiValueRange</code>	Unzulässiger Wert für index oder speed.
<code>apiTypeFault</code>	Unzulässiger Typ für index oder speed
<code>resourceUnavailable</code>	Der Bus kann nicht geöffnet werden. Mögliche Gründe:
	- Der Bus wurde bereits geöffnet
	- ein anderer Bus wurde geöffnet, und das gleichzeitige Öffnen ist nicht möglich

## **2.5.2 Reset-Mode einstellen**

`bus_resetMode <busHandle> <resetMode>`

Setzt für den ISP-Bus den Reset-Mode.

Nach dem Öffnen des ISP-Busses ist der ResetMode auf `pushpull` gesetzt. D.h.:

- Wenn kein Reset angelegt ist, ist die RST-Leitung aktiv high.
- Wenn ein Reset angelegt ist, ist die RST-Leitung aktiv low

**Vorbedingung:**

- gültiges busHandle

**Parameter:**

**busHandle**

Das von bus\_open erhaltene Bus-Handle.

**resetMode**

- **PIN\_ACTIVELOW:**
  - Wenn kein Reset angelegt ist, ist die RST-Leitung hochohmig.
  - Wenn ein Reset angelegt ist, ist die RST-Leitung aktiv low.
- **PIN\_ACTIVEHIGH:**
  - Wenn kein Reset angelegt ist, ist die RST-Leitung hochohmig.
  - Wenn ein Reset angelegt ist, ist die RST-Leitung aktiv high.
- **PIN\_PUSHPULL:**
  - Wenn kein Reset angelegt ist, ist die RST-Leitung aktiv high.
  - Wenn ein Reset angelegt ist, ist die RST-Leitung aktiv low.
- **PIN\_INVERTED:**
  - Wenn kein Reset angelegt ist, ist die RST-Leitung aktiv low.
  - Wenn ein Reset angelegt ist, ist die RST-Leitung aktiv high.

**Hinweis:**

- Die resetModes PIN\_ACTIVEHIGH und PIN\_INVERTED sind gegenüber der üblichen Reset-Funktion invertiert und ziehen die Leitung für einen Reset auf high. Dieses ist nur für Controller sinnvoll, bei denen Reset high aktiv ist. In diesem Fall wird PIN\_INVERTED empfohlen.

**Rückgabewert:**

- keiner.

**Exceptions:**

apiTypeFault

Unzulässiger Typ für busHandle



## 2.6 Atmel TPI-Bus

Allgemeine Informationen zu Bussen finden Sie im übergeordneten Kapitel. Hier wird darauf aufbauend auf das spezifische Verhalten bei dem TPI-Bus eingegangen.

### 2.6.1 bus\_open(TPI, ...) und verfügbare Geschwindigkeiten

```
busHandle = bus_open(TPI, <indexOfBus>, <speed>)
```

Öffnet den TPI-Bus und initialisiert die Leitungen. Die maximale Busgeschwindigkeit wird auf „speed“ begrenzt. Setzt die Programmiergeschwindigkeit für das Target.

#### Vorbedingung:

- keine

#### Parameter:

##### **busType**

TPI für TPI-Bus.

##### **indexOfBus**

Muß 0 sein (es gibt jeweils nur einen Bus).

##### **speed**

Die Geschwindigkeit des Busses, Angabe in Hz. Die unterstützten Busgeschwindigkeiten sind von der CPU-Clock (sys\_set - CpuClock) des roloFlash abhängig.

Bei maximaler CPU-Clock = 120 Mhz werden die folgenden Busgeschwindigkeiten unterstützt:

15000000	7500000	5000000	3750000	3000000
2500000	2142857	1875000	1666666	1500000
1363636	1250000	1153846	1071428	1000000
937500	882352	833333	789473	750000
714285	681818	652173	625000	600000
576923	555555	535714	517241	500000

483870	468750	454545	441176	428571
416666	405405	394736	384615	375000
365853	357142	348837	340909	333333
326086	319148	312500	306122	300000
294117	288461	283018	277777	272727
267857	263157	258620	254237	250000
245901	241935	238095	234375	230769
227272	223880	220588	217391	214285
211267	208333	205479	202702	200000
197368	194805	192307	189873	187500
185185	182926	180722	178571	176470
174418	172413	170454	168539	166666
164835	163043	161290	159574	157894
156250	154639	153061	151515	150000
148514	147058	145631	144230	142857
141509	140186	138888	137614	136363
135135	133928	132743	131578	130434
129310	128205	127118	126050	125000
123966	122950	120967	119047	117187
115384	113636	111940	110294	108695
107142	105633	104166	102739	101351
100000	98684	97402	96153	94936
93750	92592	91463	90361	89285
88235	87209	86206	84745	83333
81967	80645	79365	78125	76923
75757	74626	73529	72463	71428
70422	69124	67873	66666	65502
64377	63291	62240	61224	60000
58823	57692	56603	55555	54545
53380	52264	51194	50167	49019
47923	46875	45871	44776	43731
42613	41551	40540	39473	38461
37406	36319	35294	34246	33185
32119	31055	30000	28957	27932
26929	25906	24875	23847	22831
21802	20775	19762	18750	17730
16722	15706	14705	13698	12690
11682	10676	9671	8670	7668
6666	5664	4662	3661	2660
1659				

Bei minimaler CpuClock = 24 Mhz werden die folgenden Busgeschwindigkeiten unterstützt:

1500000	750000	500000	375000	300000
250000	214285	187500	166666	150000
136363	125000	115384	107142	100000
93750	88235	83333	78947	75000
71428	68181	65217	62500	60000
57692	55555	53571	51724	50000
48387	46875	45454	44117	42857
41666	40540	39473	38461	36585
34883	33333	31914	30612	29411
28301	27272	25862	24590	23437
22388	21126	20000	18987	17857
16853	15789	14705	13636	12605
11538	10489	9433	8426	7425
6410	5395	4385	3378	2377
1376				

Falls die angegebene Frequenz nicht unterstützt wird, dann wird intern auf den nächsten möglichen Wert abgerundet.

#### **Hinweis:**

Falls Sie die Schnittstelle schon geöffnet haben und dann mittels `sys_setCpuClock` den Takt des roloFlashs ändern, dann ändert sich auch die Geschwindigkeit des Busses. Daher wird empfohlen:

- Verwenden Sie `sys_setCpuClock` zuerst und öffnen dann erst den Bus.
- Oder setzen Sie nach `setCpuClock` die Busgeschwindigkeit erneut mittels `bus_setSpeed`.

#### **Rückgabewert:**

- ein Bus-Handle. Dieses kann benutzt werden, um weitere Funktionen wie z.B. `getTargetPresent` aufzurufen.

**Exceptions:**

<code>apiValueRange</code>	Unzulässiger Wert für index oder speed.
<code>apiTypeFault</code>	Unzulässiger Typ für index oder speed
<code>resourceUnavailable</code>	Der Bus kann nicht geöffnet werden. Mögliche Gründe: <ul style="list-style-type: none"><li>- Der Bus wurde bereits geöffnet</li><li>- ein anderer Bus wurde geöffnet, und das gleichzeitige Öffnen ist nicht möglich</li></ul>

## 2.6.2 Reset-Mode einstellen

`bus_resetMode <busHandle> <resetMode>`

Setzt für den TPI-Bus den Reset-Mode.

Nach dem Öffnen des TPI-Busses ist der ResetMode auf `pushpull` gesetzt. D.h.:

- Wenn kein Reset angelegt ist, ist die RST-Leitung aktiv high.
- Wenn ein Reset angelegt ist, ist die RST-Leitung aktiv low

**Vorbedingung:**

- gültiges Bus-Handle

**Parameter:**

**busHandle**

Das von `bus_open` erhaltene Bus-Handle.

**resetMode**

- **PIN\_ACTIVELOW:**
  - Wenn kein Reset angelegt ist, ist die RST-Leitung hochohmig.
  - Wenn ein Reset angelegt ist, ist die RST-Leitung aktiv low.
- **PIN\_ACTIVEHIGH:**

- Wenn kein Reset angelegt ist, ist die RST-Leitung hochohmig.
- Wenn ein Reset angelegt ist, ist die RST-Leitung aktiv high.

- **PIN\_PUSH\_PULL:**

- Wenn kein Reset angelegt ist, ist die RST-Leitung aktiv high.
- Wenn ein Reset angelegt ist, ist die RST-Leitung aktiv low.

- **PIN\_INVERTED:**

- Wenn kein Reset angelegt ist, ist die RST-Leitung aktiv low.
- Wenn ein Reset angelegt ist, ist die RST-Leitung aktiv high.

**Hinweis:**

- Die resetModes PIN\_ACTIVEHIGH und PIN\_INVERTED sind gegenüber der üblichen Reset-Funktion invertiert und ziehen die Leitung für einen Reset auf high. Dieses ist nur für Controller sinnvoll, bei denen Reset high aktiv ist. In diesem Fall wird PIN\_INVERTED empfohlen.

**Rückgabewert:**

- keiner

**Exceptions:**

apiTypeFault

Unzulässiger Typ für busHandle

## 2.7 Atmel PDI-Bus

Allgemeine Informationen zu Bussen finden Sie im übergeordneten Kapitel. Hier wird darauf aufbauend auf das spezifische Verhalten bei dem PDI-Bus eingegangen.

### 2.7.1 bus\_open(PDI, ...) und verfügbare Geschwindigkeiten

busHandle = bus\_open(PDI, <indexOfBus>, <speed>)

Öffnet den PDI-Bus und initialisiert die Leitungen. Die maximale Busgeschwindigkeit wird auf „speed“ begrenzt. Setzt die Programmiergeschwindigkeit für das Target.

**Vorbedingung:**

- keine

**Parameter:**

**busType**

PDI für PDI-Bus.

**indexOfBus**

Muß 0 sein (es gibt jeweils nur einen Bus).

**speed**

Die Geschwindigkeit des Busses, Angabe in Hz. Die unterstützten Busgeschwindigkeiten sind von der CPU-Clock ( `sys_setCpuClock` ) des roloFlash abhängig.

Bei maximaler CPU-Clock = 120 Mhz werden die folgenden Busgeschwindigkeiten unterstützt:

15000000	7500000	5000000	3750000	3000000
2500000	2142857	1875000	1666666	1500000
1363636	1250000	1153846	1071428	1000000
937500	882352	833333	789473	750000
714285	681818	652173	625000	600000
576923	555555	535714	517241	500000
483870	468750	454545	441176	428571
416666	405405	394736	384615	375000
365853	357142	348837	340909	333333
326086	319148	312500	306122	300000
294117	288461	283018	277777	272727
267857	263157	258620	254237	250000
245901	241935	238095	234375	230769
227272	223880	220588	217391	214285
211267	208333	205479	202702	200000
197368	194805	192307	189873	187500
185185	182926	180722	178571	176470

174418	172413	170454	168539	166666
164835	163043	161290	159574	157894
156250	154639	153061	151515	150000
148514	147058	145631	144230	142857
141509	140186	138888	137614	136363
135135	133928	132743	131578	130434
129310	128205	127118	126050	125000
123966	122950	120967	119047	117187
115384	113636	111940	110294	108695
107142	105633	104166	102739	101351
100000				

Bei minimaler CpuClock = 24 Mhz werden die folgenden Busgeschwindigkeiten unterstützt:

1500000	750000	500000	375000	300000
250000	214285	187500	166666	150000
136363	125000	115384	107142	100000

Falls die angegebene Frequenz nicht unterstützt wird, dann wird intern auf den nächsten möglichen Wert abgerundet. Die minimale Busgeschwindigkeit wird von Atmel mit 100 kHz angegeben. Bei Angabe von kleineren Werten wird auf 100 kHz aufgerundet.

### **Hinweis:**

Falls Sie die Schnittstelle schon geöffnet haben und dann mittels `sys_setCpuClock` den Takt des roloFlashs ändern, dann ändert sich auch die Geschwindigkeit des Busses. Daher wird empfohlen:

- Verwenden Sie `sys_setCpuClock` zuerst und öffnen dann erst den Bus.
- Oder setzen Sie nach `sys_setCpuClock` die Busgeschwindigkeit erneut mittels `bus_setSpeed`.

### **Rückgabewert:**

- ein Bus-Handle. Dieses kann benutzt werden, um weitere Funktionen wie z.B. `getTargetPresent` aufzurufen.

**Exceptions:**

apiValueRange	Unzulässiger Wert für index oder speed.
apiTypeFault	Unzulässiger Typ für index oder speed
resourceUnavailable	Der Bus kann nicht geöffnet werden. Mögliche Gründe: <ul style="list-style-type: none"><li>- Der Bus wurde bereits geöffnet</li><li>- ein anderer Bus wurde geöffnet, und das gleichzeitige Öffnen ist nicht möglich</li></ul>

## 2.8 Atmel UPDI-Bus / aWire-Bus

Allgemeine Informationen zu Bussen finden Sie im übergeordneten Kapitel. Hier wird darauf aufbauend auf das spezifische Verhalten bei dem UPDI-Bus bzw. aWire-Bus eingegangen.

### 2.8.1 bus\_open(UPDI / AWIRE, ...) und verfügbare Geschwindigkeiten

```
busHandle = bus_open(UPDI, <indexOfBus>, <speed>)
```

```
busHandle = bus_open(AWIRE, <indexOfBus>, <speed>)
```

Öffnet den UPDI-Bus bzw. aWire-Bus und initialisiert die Leitungen. Die maximale Busgeschwindigkeit wird auf „speed“ begrenzt. Setzt die Programmiergeschwindigkeit für das Target.

**Vorbedingung:**

- keine

**Parameter:**

**busType**

UPDI für UPDI-Bus bzw. AWIRE für aWire-Bus

**indexOfBus**

Muß 0 sein (es gibt jeweils nur einen Bus).

**speed**



Die Geschwindigkeit des Busses, Angabe in Hz. Die unterstützten Busgeschwindigkeiten sind von der CPU-Clock (sys\_set - CpuClock) des roloFlash abhängig.

Bei maximaler CPU-Clock = 120 Mhz werden die folgenden Busgeschwindigkeiten unterstützt:

500000	483870	468750	454545	441176
428571	416666	405405	394736	384615
375000	365853	357142	348837	340909
333333	326086	319148	312500	306122
300000	294117	288461	283018	277777
272727	267857	263157	258620	254237
250000	245901	241935	238095	234375
230769	227272	223880	220588	217391
214285	211267	208333	205479	202702
200000	197368	194805	192307	189873
187500	185185	182926	180722	178571
176470	174418	172413	170454	168539
166666	164835	163043	161290	159574
157894	156250	154639	153061	151515
150000	148514	147058	145631	144230
142857	141509	140186	138888	137614
136363	135135	133928	132743	131578
130434	129310	128205	127118	126050
125000	123966	122950	120967	119047
117187	115384	113636	111940	110294
108695	107142	105633	104166	102739
101351	100000	98684	97402	96153
94936	93750	92592	91463	90361
89285	88235	87209	86206	84745
83333	81967	80645	79365	78125
76923	75757	75000		

Bei minimaler CpuClock = 24 Mhz werden die folgenden Busgeschwindigkeiten unterstützt:

375000	300000	250000	214285	187500
166666	150000	136363	125000	115384
107142	100000	93750	88235	83333
78947	75000			

Falls die angegebene Frequenz nicht unterstützt wird, dann wird intern auf den nächsten möglichen Wert abgerundet. Die minimale Busgeschwindigkeit wird von Atmel mit 100 kHz angegeben. Bei Angabe von kleineren Werten wird auf 100 kHz aufgerundet.

#### **Hinweis:**

Falls Sie die Schnittstelle schon geöffnet haben und dann mittels `sys_setCpuClock` den Takt des roloFlashs ändern, dann ändert sich auch die Geschwindigkeit des Busses. Daher wird empfohlen:

- Verwenden Sie `sys_setCpuClock` zuerst und öffnen dann erst den Bus.
- Oder setzen Sie nach `sys_setCpuClock` die Busgeschwindigkeit erneut mittels `bus_setSpeed`.

#### **Rückgabewert:**

- ein Bus-Handle. Dieses kann benutzt werden, um weitere Funktionen wie z.B. `getTargetPresent` aufzurufen.

#### **Exceptions:**

<code>apiValueRange</code>	Unzulässiger Wert für index oder speed.
<code>apiTypeFault</code>	Unzulässiger Typ für index oder speed
<code>resourceUnavailable</code>	Der Bus kann nicht geöffnet werden. Mögliche Gründe: <ul style="list-style-type: none"><li>- Der Bus wurde bereits geöffnet</li><li>- ein anderer Bus wurde geöffnet, und das gleichzeitige Öffnen ist nicht möglich</li></ul>

---

## **3 Target allgemein**

Um den Zugriff auf ein Target zu erhalten, muß vom zuvor geöffneten Bus ein Target-Handle angefordert werden. Alle Funktionen mit dem Target geschehen dann unter Angabe dieses Target-Handles. Bei roloFlash wird grundsätzlich jede Schnittstelle, über die ein Target geflasht werden kann, als Bus aufgefaßt. Dieses gilt auch, wenn an dieser Schnittstelle prinzipbedingt nur ein einziger Mikrocontroller angeschlossen sein kann (z. B. wird die ISP-Schnittstelle für Atmel AVR als Bus aufgefaßt).

- Grundsätzlich muß vorab der passende Bus geöffnet worden sein.
- Ein an dem Bus angeschlossener Mikrocontroller (Target) kann erst angesprochen werden, wenn man von dem Bus ein Target-Handle erhalten hat.
- Die Verbindung zu einem Target kann auch wieder geschlossen werden.
- Wenn der Bus geschlossen wird, wird das Target auch geschlossen.

### 3.1 target\_open

```
targetHandle = target_open(<busHandle>, <index>, <family>)
```

Ermöglicht den Zugriff auf ein Target und liefert ein Target-Handle.

#### **Hinweis:**

Die Funktion prüft nicht, ob tatsächlich ein Target angeschlossen ist. Falls das geprüft werden soll, kann target\_getPresent verwendet werden.

#### **Vorbedingung:**

- gültiges BusHandle

#### **Parameter:**

##### **busHandle**

Das Bus-Handle auf den geöffneten Bus.

##### **index**

Gibt an, welches Target auf dem Bus geöffnet werden soll. Die Zählweise ist vom Bus abhängig. In den meisten Fällen sind die Targets durchnummeriert – das erste Target hat den Index 0.

Bei Bussen, die nur ein Target unterstützen, muß als Index immer die 0 angegeben werden.

#### **Hinweis:**

Bitte bei Bussen, an denen nur ein Target angeschlossen sein kann (z.B. ISP-Bus) immer 0 angeben.

### **family**

Mit diesem Parameter wird festgelegt, welcher Controller-Familie der Controller zugeordnet ist. Der Wert dazu kann direkt angegeben werden oder gegebenenfalls aus der internen Datenbank ausgelesen werden. Mögliche Familien: ATMELISP, ATMELPDI, ATMELUPDI, ATMELAVR32, ATMELTPI.

### **Rückgabewert:**

- ein Target-Handle. Dieses kann benutzt werden, um weitere Funktionen wie z.B. `target_getPresent` aufzurufen.

### **Exceptions:**

`apiValueRange`  
`apiTypeFault`  
`invalidHandle`

Unzulässiger Wert für index  
Unzulässiger Typ für busHandle oder index  
Das BusHandle ist ungültig (z.B. schon geschlossen)

## **3.2 target\_close**

`target_close <targetHandle>`

Schließt das entsprechende Target.

### **Vorbedingung:**

- gültiges Target-Handle

### **Parameter:**

**targetHandle**

Das Target-Handle auf das geöffnete Target.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

invalidHandle

Das Target-Handle ist schon geschlossen, oder der dazu gehörige Bus wurde schon geschlossen.

apiTypeFault

Unzulässiger Typ für targetHandle

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**Rückgabewert:**

0 = kein Target gefunden

1 = Target gefunden

**Exceptions:**

invalidHandle

Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.

apiTypeFault

Unzulässiger Typ des Target-Handles

### 3.3 target\_getPresent

value = target\_getPresent(<targetHandle>)

Ermittelt, ob ein Target angeschlossen ist. Der Modus bleibt dabei unverändert. Es findet auf jeden Fall eine Kommunikation mit dem Target statt, so daß man eine aktuelle Information erhält.

### **Hinweis Atmel ISP-Bus:**

Falls das Target im RunMode ist, dann wird bei dem Target vorübergehend ein Reset angelegt und es in den ProgramMode versetzt. Nach der Abfrage wird der Reset aufgehoben und der RunMode erreicht. Ein evtl. auf dem Target laufendes Programm wird dadurch neu gestartet.

Falls das Target schon im ProgramMode ist, dann wird ebenso eine Abfrage gestartet. Das Target bleibt im ProgramMode.

### **Hinweis Atmel PDI-Bus und Atmel UPDI-Bus:**

Unabhängig davon, ob das Target im RunMode oder ProgramMode ist, wird über den PDI-Bus/UPDI-Bus eine Abfrage gestartet. Das Target verbleibt im jeweiligen Modus. Ein Reset findet nicht statt.

### **Anmerkung:**

Bei roloFlash sollte immer ein Target angeschlossen sein, weil sonst roloFlash nicht mit Energie versorgt wäre. Die Funktion ist hauptsächlich für Programmiergeräte gedacht, die über eine eigene Energieversorgung verfügen.

Des weiteren ist es denkbar, daß roloFlash auf etwas anderes als ein Target aufgesteckt wird. Daher baut diese Funktion tatsächlich eine Kommunikation mit dem Target auf.

### **Vorbedingung:**

- gültiges Target-Handle

### **Parameter:**

#### **targetHandle**

Das Target-Handle auf das anzusprechende Target

### **Rückgabewert:**

0 = kein Target gefunden

1 = Target gefunden

**Exceptions:**

invalidHandle

Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.

apiTypeFault

Unzulässiger Typ des Target-Handles

Das Target kann sich in folgenden Betriebsmodi befinden:

- **RunMode:** Target läuft ganz normal, als ob roloFlash nicht angeschlossen wäre.
- **ProgramMode:** Target kann programmiert werden.

Die Prozedur target\_setMode wechselt den Betriebsmodus.

Andere Prozeduren bzw. Funktionen sind auf einen bestimmten Modus angewiesen. In diesem Fall steht das in der jeweiligen Beschreibung.

### 3.4 target\_setMode

target\_setMode <targetHandle>, <targetMode>

Bringt das Target und roloFlash in den angegebenen Betriebsmodus.

Das Target kann sich in folgenden Betriebsmodi befinden:

- **RunMode:** Target läuft ganz normal, als ob roloFlash nicht angeschlossen wäre.
- **ProgramMode:** Target kann programmiert werden.

Andere Prozeduren bzw. Funktionen sind auf einen bestimmten Modus angewiesen. In diesem Fall steht das in der jeweiligen Beschreibung.

**Vorbedingung:**

- gültiges targetHandle

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**targetMode**

Angabe des gewünschten Modus:

**PROGRAMMODE:** Dieser Modus ist Voraussetzung für die meisten weiteren Funktionen mit dem Target, insbesondere für das Beschreiben des Flashspeichers. Dabei kann je nach Targetfamilie das Target gestoppt werden.

**RUNMODE:** Das Target läuft. Eine etwaige auf dem Target befindliche Software wird ausgeführt.

**Rückgabewert:**

- keiner (Prozedur)

**Hinweis Atmel ISP-Bus:**

- programMode: Falls das Target im RunMode ist, dann wird das Target in den „Programming Enable Mode“ geschaltet und im Reset gehalten. Ein evtl. auf dem Target laufendes Programm wird dadurch angehalten.
- runMode: Der „Programming Enable Mode“ wird aufgehoben, der Reset wird weggenommen. Das Target läuft danach sofort los.

**Hinweis Atmel PDI-Bus:**

- programMode: Hat keinen Einfluss darauf, ob das Target gerade läuft oder steht. Es werden hier Initialisierungen für den Zugriff über den PDI-Bus durchgeführt.
- runMode: Der PDI-Takt wird gestoppt, folgend wird der „Programming Mode“ beendet. Das Target führt einen Reset durch und läuft los.



### **Hinweis Atmel UPDI-Bus:**

- **programMode:** Falls das Target im RunMode ist, dann wird das Target in den „Programming Enable Mode“ geschaltet und im Reset gehalten. Ein evtl. auf dem Target laufendes Programm wird dadurch angehalten.
- **runMode:** Der „Programming Enable Mode“ wird aufgehoben, der Reset wird weggenommen. Das Target läuft danach sofort los.
- Befindet sich das Target in dem „Programming Enable Mode“ und roloFlash wird abgezogen, dann verbleibt das Target in diesem Mode. Ein auf dem Target befindliches Programm startet nicht bis die Stromversorgung für das Target kurzzeitig unterbrochen wird. Das Starten des Targets kann erzwungen werden, indem vor dem Entfernen des roloFlashs `target_setMode` mit dem Parameter `runMode` aufgerufen wird. Alternativ kann auch das `targetHandle` mittels `target_close` geschlossen werden.

### **Exceptions:**

<code>targetCommunication</code>	Die Kommunikation mit dem Target funktioniert nicht.
<code>invalidHandle</code>	Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
<code>apiTypeFault</code>	Unzulässiger Typ für das Target-Handle

## **3.5 target\_restart**

`target_restart <targetHandle>`

Startet das Target neu und nimmt wieder denselben Zustand ein:

### **RunMode**

Es wird kurzzeitig der Reset aktiviert, dann deaktiviert. Das Target läuft daher vom Anfang los. Der RunMode wird beibehalten.

### **ProgramMode**

Es wird ebenso ein Reset durchlaufen, dann wieder der ProgramMode hergestellt. Zwischenzeitlich kann eine evtl. auf dem Target befindliche Firmware bereits für kurze Zeit losgelaufen sein.

Es wird empfohlen, dieses Kommando nur einzusetzen, wenn dieses entweder nicht kritisch ist oder sich noch keine Firmware auf dem Target befindet.

### **Hinweis Atmel ISP-Bus:**

Der „Programming Enable Mode“ wird aufgehoben, der Reset wird weggenommen. Das Target läuft damit nach einem Reset los.

### **RunMode**

Es wird kurzzeitig (100 ms) der Reset aktiviert, dann deaktiviert. Das Target läuft daher vom Anfang los. Der RunMode wird beibehalten.

### **ProgramMode**

Der Reset wird kurzzeitig (3 ms) aufgehoben, dann wieder der ProgramMode hergestellt. Zwischenzeitlich kann eine evtl. auf dem Target befindliche Firmware bereits für kurze Zeit losgelaufen sein.

### **Anwendungsbeispiel für Targets mit Atmel ISP-Interface:**

Die Prozedur wird benötigt, wenn man z. B. Fuses auf dem Target ändert und die Änderungen sofort aktiviert werden sollen. Das gilt insbesondere für das Aktivieren eines Quarzes für das Target, was dann anschließend eine höhere Programmiergeschwindigkeit ermöglicht:

```
! Quarz aktivieren, damit höhere
! Programmiergeschwindigkeit möglich
target_writeBits(targetHandle, FUSES_LOW, value)
! Durch target_restart die Änderung aktivieren
target_restart targetHandle
bus_setSpeed(bushandle, 1000000) ! z.B. 1 MHz
target_writeFromFile ...
```

### **Hinweis Atmel PDI-Bus:**

Der Reset ist zwar Teil des PDI-Busses, wird aber bei PDI nicht als Reset genutzt. Folglich kann der Bus benutzt werden, ohne das Target im Reset zu halten.

### **RunMode**

Es wird kurzzeitig (100 ms) der Reset aktiviert, dann deaktiviert. Das Target läuft daher vom Anfang los. Der RunMode wird beibehalten.

### **ProgramMode**

Der PDI-Bus wird deaktiviert, ein Reset ausgelöst (100 ms), anschließend der PDI-Bus aktiviert und der ProgramMode wieder hergestellt. Das Target läuft vom Anfang los.

### **Hinweis UPDI-Bus:**

Da beim UPDI-Bus keine Resetleitung vorhanden ist, steht dieser Befehl nicht zur Verfügung.

### **Vorbedingung:**

- gültiges Target-Handle

### **Parameter:**

#### **targetHandle**

Das Target-Handle auf das anzusprechende Target

### **Rückgabewert:**

- keiner (Prozedur)

### **Exceptions:**

targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für einen der Parameter.

## 3.6 Target-Memorymap lesen/schreiben

Für verschiedene Speicherarten in den Targets unterstützt roloFlash eine sogenannte MemoryMap. Diese kann je nach Target und Speicherart Informationen über verschiedene Eigenschaften (sog. Properties) des Speichers zur Verfügung stellen bzw. diese Eigenschaften können hier eingestellt werden. Manche Eigenschaften müssen vor dem Flashen richtig gesetzt werden. Oft finden Sie die benötigten Werte in der Datenbank.

Ein guter Ansatzpunkt dazu sind die Beispielskripte.

### 3.6.1 target\_setMemoryMap

```
target_setMemoryMap <targetHandle>, <memType>, <memProperty> <value>
```

Setzt für den angegebenen Speichertyp das entsprechende Property auf den angegebenen Wert.

#### Vorbedingung:

- gültiges Target-Handle

#### Parameter:

##### **targetHandle**

Das Target-Handle auf das anzusprechende Target

##### **memType**

die gewählte Speicherart:

**FLASH:** Flash-Speicher

**RAM:** RAM-Speicher

**EEPROM:** EEPROM-Speicher

##### **memProperty**

das gewählte Property:

**MEM\_STARTADDR:** Startadresse des Speichers

**MEM\_SIZE:** Größe des Speichers in Bytes

**MEM\_PAGESIZE:** für bestimmte Targets: Größe einer Speicherseite

**value**

der zu setzende Wert

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
FunctionNotSupported	Unzulässige Kombination von MemType und Property
apiTypeFault	Unzulässiger Typ für einen der Parameter.
ValueNotAllowed	Unzulässiger Wert für value

### 3.6.2 target\_getMemoryMap

Value = target\_getMemoryMap(<targetHandle>, <memType>, <memProperty>)

Ermittelt für den angegebenen Speichertyp den Wert für das entsprechende Property.

**Vorbedingung:**

- gültiges targetHandle

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

### **memType**

die gewählte Speicherart:

**FLASH:** Flash-Speicher

**RAM:** RAM-Speicher

**EEPROM:** EEPROM-Speicher

### **memProperty**

das gewählte Property:

**MEM\_STARTADDR:** Startadresse des Speichers

**MEM\_SIZE:** Größe des Speichers in Bytes

**MEM\_PAGESIZE:** für bestimmte Targets: Größe einer Speicherseite

### **Rückgabewert:**

- gelesener Wert

### **Exceptions:**

targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
FunctionNotSupported	Unzulässige Kombination von MemType und Property
apiTypeFault	Unzulässiger Typ für einen der Parameter.
valueUnknown	Der Wert kann nicht ermittelt werden

### **3.6.3 target\_clearMemoryLayout**

target\_clearMemoryLayout <targetHandle>

Löscht ein bereits vorhandenes Speicherlayout (Flash- und EEPROM-Layout).

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

targetWrongMode

Target ist nicht im "ProgramMode".

invalidHandle

Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.

apiTypeFault

Unzulässiger Typ für das Target-Handle.

## 3.7 Target löschen, schreiben, lesen und verifizieren

### 3.7.1 target\_erase

target\_erase <targetHandle>

target\_erase <targetHandle>, <memType>

Löscht den angegebenen Speicher.

Wenn memType nicht angegeben ist oder als memType FLASH angegeben ist, dann wird das gesamte Flash (Mass-Erase) des Targets gelöscht.

Bei STM32L0, STM32L1 und STM32WB wird dies nicht unterstützt. Diese Targets müssen pageweise gelöscht werden.

Bei manchen Targets wird dabei automatisch auch das EEPROM gelöscht (z.B. Atmel-Fuse „EESAVE“). Informationen dazu finden Sie im jeweiligen Datenblatt des Targets.

`target_erase <targetHandle, memType, eraseScope, number>`

Löscht den angegebenen Speicher des Targets in gegebenem Umfang.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**memType**

(optional)

Die Art des Speichers, der gelöscht werden soll.

Zulässige Werte:

- FLASH

**eraseScope**

(zusammen mit number optional, memType muss angegeben sein)

(nur bei STM32L0, STM32L1 und STM32WB)

Zulässige Werte:

- PAGE: Zum Löschen einzelner pages

**number**

(zusammen mit eraseScope optional, memType muss angegeben sein)



(nur bei STM32L0, STM32L1 und STM32WB)

Die Bedeutung ist von der verwendeten Familie abhängig:

- STM32L0 und STM32L1: erste Adresse innerhalb der Page, die gelöscht werden soll
- STM32WB: Index der zu löschenden Page

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für das Target-Handle.

### **3.7.2 target\_writeFromFile**

target\_writeFromFile <targetHandle>, <fileSystem>, <fileName>, <fileFormat>, <memType>, <verify>, <startAddr>, <cryptSpec>

Schreibt eine Datei in den Speicher des Targets.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**fileSystem**

Gibt an, auf welchem Dateisystem sich die Datei befindet. Mögliche Werte sind:

**SDCARD, FLASHVARS, FLASHDISK**

#### **fileName**

Es gelten die Bedingungen für Dateinamen, siehe Kapitel „Flash-Da-ta“.

#### **fileFormat**

Gibt an, in welchem Format die Datei vorliegt. Mögliche Werte sind:  
**HEX**: Intel-HEX-Format (ASCII-Datei). (Als Angabe wird auch HEX32 akzeptiert und gleich behandelt)

**SREC**: Motorola SREC-Format (S19, S28, S37, ASCII-Datei). (Als Angabe wird auch S19, S28 und S37 akzeptiert und gleich behandelt)

**ELF**: ELF-Format, welches von vielen Compilern als Zwischenschritt erzeugt wird. Es enthält oft viele weitere Informationen, die für das Flashen nicht benötigt und ignoriert werden.

**RAW**: Raw-Format (Binärdatei mit Rohdaten ohne Adreßangabe)

#### **memType**

Gibt an, welcher Speicher beschrieben werden soll. Die Angabe ist für die jeweilige Targetfamilie spezifisch und ist in den jeweiligen Kapiteln beschrieben.

#### **verify**

Gibt an, ob ein Verify durchgeführt werden soll. Mögliche Werte sind:

**WRITEONLY**: Schreiben ohne Verify

**VERIFYONLY**: Die Daten werden nur verifiziert (kein Schreiben auf dem Target)

**WRITEVERIFY**: Schreiben und Verifizieren

#### **startAddr**

(optional). Der Parameter ist ausschließlich für das Raw-Format vorgesehen. Da im Raw-Format in der Datei keine Adresse angegeben ist, muß hiermit die Startadresse spezifiziert werden.

#### **cryptSpec**

optionaler Parameter zum Schreiben einer verschlüsselten Datei. Weitere Informationen zum Aufbau des Parameters finden Sie in dem Kapitel „Target löschen, schreiben, lesen und verifizieren“

Für den Parameter padding innerhalb der enthaltenden dataSpec gilt:

- **0-15:** Anzahl der Bytes, die nach der Entschlüsselung ignoriert werden sollen.
- **SEC\_NOPADDING:** Nach der Entschlüsselung werden die Daten nicht gekürzt. Dieser Mode ist nur bei SEC\_CTR anwendbar und dafür empfohlen.
- **SEC\_PKCS7:** Nach der Entschlüsselung werden die Daten gemäß PKCS7-Padding gekürzt. Sollten die Daten nicht PKCS7 entsprechen, dann wird eine Exception "paddingError" erzeugt.

#### **Rückgabewert:**

- keiner (Prozedur)

#### **Hinweis zu Verify = WRITEVERIFY**

Die geschriebenen Daten werden vom Target zurückgelesen und mit den aus der Datei gelesenen Daten im roloFlash verglichen. Die Daten werden dazu nicht ein zweites Mal von der microSD-Karte gelesen und dekodiert. Etwaige Lesefehler von der microSD-Karte werden so nicht erkannt. Allerdings werden bei Hex-Dateien auch die CRC-Werte ausgelesen und überprüft, so daß Lesefehler dementsprechend unwahrscheinlich sind.

Wenn Sie die Sicherheit weiter erhöhen wollen, dann benutzen Sie bitte zwei Aufrufe: ein Aufruf mit verify = WRITEONLY und den zweiten Aufruf mit verify = VERIFYONLY. Dieses Vorgehen kann länger dauern als ein einziger Aufruf mit verify = WRITEVERIFY.

#### **Hinweis zu HEX- und SREC-Dateien**

Zeilen, die mit einem ';' anfangen, werden als Kommentar verstanden und ignoriert.

### **Exceptions:**

targetMemoryLayout, hexFileSize, hexFileCRC, hexFileSyntax, srecRecordTypeTooSmall	Siehe Kapitel „Exceptions des roloFlash“.
targetWrongMode targetCommunication	Target ist nicht im "ProgramMode". Die Kommunikation mit dem Target funktioniert nicht.
targetError apiTypeFault invalidHandle	Es gibt einen Fehler auf dem Target Unzulässiger Typ für einen der Parameter. Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
targetVerify	Beim Verify wurden andere Daten gelesen. Mögliche Ursachen: - Kommunikationsprobleme - Zu hohe Datenrate - Das Target ist vorher nicht gelöscht worden (betrifft vornehmlich Flash-Speicher)
<diverse Exceptions des Dateisystems>	Siehe Kapitel „Exceptions des Dateisystems“.
cryptError secParamError secPaddingError	Allgemeiner Fehler bei der Berechnung. Die angegeben CryptSpec ist fehlerhaft. PKCS7 ist angegeben, die entschlüsselten Daten entsprechen nicht der PKCS7 Kodierung.

### **3.7.3 target\_readToFile**

target\_readToFile <targetHandle>, <fileSystem>, <fileName>, <fileFormat>, <memType>, <startAddr>, <length>

Liest aus dem Speicher des Targets, erzeugt eine neue Datei und schreibt die Daten im angegebenen Format in die Datei.

### **Vorbedingung:**

- gültiges targetHandle
- das Target muß im ProgramMode sein.

### **Parameter:**

### targetHandle

Das targetHandle auf das anzusprechende Target.

### fileSystem

Gibt an, auf welchem Dateisystem die Datei angelegt werden soll.  
Mögliche Werte sind:

**SDCARD, FLASHVARS, FLASHDISK**

### fileName

Es gelten die Bedingungen für Dateinamen, siehe Kapitel „Flash-Da-ta“. Falls die Datei schon existieren sollte, dann wird sie überschrie-ben.

### fileFormat

Gibt an, in welchem Format die Daten geschrieben werden sollen.  
Mögliche Werte sind:

**HEX:** Intel-HEX-Format (ASCII-Datei). Wenn der Adressbereich unter 1MB liegt, dann wird der Extended Segment Address Record (Typ 02) verwendet (Format I16HEX), andernfalls wird der Extended Linear Address Record (Typ 04) verwendet (Format I32HEX).

**HEX32:** Intel-HEX-Format (ASCII-Datei). Unabhängig des Adressbe-reiches wird immer der Extended Linear Address Record (Typ 04) verwendet (Format I32HEX).

**SREC:** Motorola-SREC-Format (ASCII-Datei). Je nach Adressbe-reich wird die kleinstmögliche Kodierung verwendet (S19, S28 oder S37)

**S19:** Motorola-SREC-Format im S19 Format (ASCII-Datei). Sollte der Adressbereich (64KB) nicht ausreichend sein, wird die Exception ad-dressTooBigForFileFormat erzeugt.

**S28:** Motorola-SREC-Format im S28 Format (ASCII-Datei). Sollte der Adressbereich (16MB) nicht ausreichend sein, wird die Exception ad-dressTooBigForFileFormat erzeugt.

**S37:** Motorola-SREC-Format im S37 Format (ASCII-Datei).

**RAW:** Raw-Format (Binärdatei mit Rohdaten ohne Adreßangabe)

Bei Intel-Hex-Format und bei Motorola SREC-Format kann optional die Anzahl der Datenbytes pro Zeile dazu verodert werden. Wenn dieses nicht angegeben wird, dann wird der Default der jeweiligen Formate verwendet (Intel-Hex-Format: 16 Bytes, Motorola SREC-Format: 32 Bytes). Maximal ist ein Wert von 255 zulässig, größere Werte führen zu unerwarteten Fehlfunktion. Während bei den Intel-

Hex-Formaten 255 Bytes möglich sind, ist bei den verschiedenen Motorola SREC-Formaten die maximale Anzahl wenige Bytes geringer. Der angegebene Wert wird automatisch korrigiert. Somit erhalten Sie mit der Angabe von 255 immer die maximal mögliche Anzahl von Bytes pro Zeile.

Bei Intel-Hex-Format und bei Motorola SREC-Format werden die Zeilenende mit <cr><lf> kodiert. Sollen Dateien mit ausschließlich <cr> oder <lf> kodiert werden, so kann die Konstante **CR** bzw. **LF** dazu verodert werden.

Beispiel: **HEX32 or 64 or LF** erzeugt I32HEX-Files mit 64 Bytes Daten pro Zeile und nur einem <lf> für die Zeilenenden.

### **memType**

Gibt an, welcher Speicher gelesen werden soll. Die Angabe ist für die jeweilige Targetfamilie spezifisch und ist in den jeweiligen Kapiteln beschrieben.

### **startAddr**

Bestimmt die erste Adresse, ab der gelesen werden soll.

### **length**

Bestimmt die Anzahl in Bytes, die gelesen werden sollen.

### **Rückgabewert:**

- keiner (Prozedur)

### **Hinweis zu Verify beim Lesen**

Um ein Verify, ähnlich wie beim Schreiben ins Target, zu erreichen, kann man die gelesene Datei anschließend mittels `target_writeFromFile` mit `verify = verifyOnly` überprüfen.

### **Exceptions:**

targetMemoryLayout,  
hexFileSize,  
hexFileCRC,  
hexFileSyntax und  
srecRecordTypeTooSmall

Siehe Kapitel „Exceptions des roloFlash“.

targetWrongMode  
targetCommunication

Target ist nicht im "ProgramMode".  
Die Kommunikation mit dem Target funktioniert nicht.

targetError  
apiTypeFault  
invalidHandle

Es gibt einen Fehler auf dem Target  
Unzulässiger Typ für einen der Parameter.  
Das Target-Handle ist schon geschlossen oder  
der Bus wurde schon geschlossen.

<diverse Exceptions des  
Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

### **3.7.4 target\_write**

target\_write <targetHandle>, <dataArray>, <memType>,  
<verify>, <startAddr>

Schreibt ein Daten-Array aus dem roloBasic in den Speicher des Targets.

### **Vorbedingung:**

- gültiges targetHandle
- das Target muß im ProgramMode sein.

### **Parameter:**

#### **targetHandle**

Das Target-Handle auf das anzusprechende Target

#### **dataArray**

Ein Char-Array mit den zu schreibenden Daten.

#### **memType**

Gibt an, welcher Speicher beschrieben werden soll. Die Angabe ist für die jeweilige Targetfamilie spezifisch und ist in den jeweiligen Kapiteln beschrieben.

### verify

Gibt an, ob ein Verify durchgeführt werden soll. Mögliche Werte sind:

**WRITEONLY:** Schreiben ohne Verify

**VERIFYONLY:** Die Daten werden nur verifiziert (kein Schreiben auf dem Target)

**WRITEVERIFY:** Schreiben und Verifizieren

### startAddr

Die Adresse, an die Daten im Target geschrieben werden sollen.

### Rückgabewert:

- keiner (Prozedur)

### Exceptions:

targetMemoryLayout  
targetWrongMode  
targetCommunication

Siehe Kapitel „Exceptions des roloFlash“.  
Target ist nicht im "ProgramMode".  
Die Kommunikation mit dem Target funktioniert nicht.

targetError  
apiTypeFault  
invalidHandle

Es gibt einen Fehler auf dem Target  
Unzulässiger Typ für einen der Parameter.  
Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.

targetVerify

Beim Verify wurden andere Daten gelesen.  
Mögliche Ursachen:

- Kommunikationsprobleme
- Zu hohe Datenrate
- Das Target ist vorher nicht gelöscht worden (betrifft vornehmlich Flash-Speicher)

<diverse Exceptions des Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

### 3.7.5 target\_read

```
DataArray = target_read(<targetHandle>, <memType>,
<startAddr>, <length>)
```



Liest aus dem Speicher des Targets, erzeugt ein Char-Array im Basic und befüllt dieses Array mit den gelesenen Daten.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target.

**memType**

Gibt an, welcher Speicher gelesen werden soll. Die Angabe ist für die jeweilige Targetfamilie spezifisch und ist in den jeweiligen Kapiteln beschrieben.

**startAddr**

Bestimmt die erste Adresse, ab der gelesen werden soll.

**length**

Bestimmt die Anzahl in Bytes, die gelesen werden sollen.

**Rückgabewert:**

- Char-Array mit den ausgelesenen Daten

**Hinweis zu Verify beim Lesen**

Um ein Verify, ähnlich wie beim Schreiben ins Target, zu erreichen, kann man die gelesene Datei anschließend mittels `target_write` mit `verify = VERIFYONLY` überprüfen.

### **Exceptions:**

OutOfMemory	Es steht nicht genug Speicher zur Verfügung, um das Basic Array anzulegen
targetMemoryLayout	Siehe Kapitel „Exceptions des roloFlash“.
targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
targetError	Es gibt einen Fehler auf dem Target
apiTypeFault	Unzulässiger Typ für einen der Parameter.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
<diverse Exceptions des Dateisystems>	Siehe Kapitel „Exceptions des Dateisystems“.

## **3.8 Target Atmel AVR (ISP-Interface)**

Es werden alle Funktionen der Kapitel „Target allgemein“ bis „Target löschen, schreiben, lesen und verifizieren“ inklusive aller Unterkapitel unterstützt.

Es wird kein Loader verwendet.

### **MemTypes:**

Unterstützte memTypes beim Schreiben:

- FLASH
- EEPROM

Unterstützte memTypes beim Lesen:

- FLASH
- EEPROM

### **3.8.1 target\_getDeviceId**

```
s = target_getDeviceId(<targetHandle>)
```

Liest die Signature / Device ID des Targets. Anhand dieser lassen sich die verschiedenen Controller unterscheiden.

### **Hinweis:**

In den Dokumenten des Herstellers werden über die verschiedenen Controller die Begriffe „Device ID“ und „Signature“ benutzt. Unabhängig davon wird bei roloFlash immer von einer Device ID ausgegangen.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das targetHandle auf das anzusprechende Target

**Rückgabewert:**

Ausgelesene Device ID bzw. Signature. Die Device ID wird in einem Byte-Array mit 3 Bytes geliefert. Sie können die Device ID mit einer Device ID aus der Datenbank vergleichen.

**Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für Target-Handle.

**3.8.2 target\_readBits**

value = target\_readBits(<targetHandle>, <index>)

Liest Fuse- bzw. Lock-Bits als Byte aus.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

## **targetHandle**

Das Target-Handle auf das anzusprechende Target

## **index**

Gibt an, welche Fuses bzw. Lock-Bits gelesen werden sollen. Es gibt dazu die Konstanten FUSES\_LOW, FUSES\_HIGH, FUSES\_EXT und LOCK\_BITS.

Bei Controllern, die keine Extended-Fuses haben, ist der ausgelesene Wert bei FUSES\_EXT unbestimmt (es wird keine Exception erzeugt).

## **Rückgabewert:**

Ausgelesene Fuse- bzw. Lock-Bits als Byte.

## **Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
apiValueRange	Unzulässiger Wert für index.
invalidHandle	Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für einen der Parameter.

## **Beispiel:**

Beispiel für FUSE\_LOW auf einem ATmega8 und einem Rückgabewert von \$1E:

\$1E entspricht binär 00011110, es sind daher folgende Fuse-Bits gesetzt:

- Bit 1 (CKSEL1)
- Bit 2 (CKSEL2)
- Bit 3 (CKSEL3)
- Bit 4 (SUT0)
- Alle anderen Fuse-Bits in den Low-Fuses sind gelöscht (also 0).

### 3.8.3 target\_writeBits

target\_writeBits <targetHandle>, <index>, <value>

Schreibt die angegebenen Fuse- bzw. Lock-Bits.

#### **Vorsicht:**

- Es kann Kombinationen geben, die Ihren Chip unbrauchbar machen. Achten Sie bitte auf die Werte und prüfen Sie diese im Handbuch des Controllers.
- Setzen Sie die Lock-Bits erst, nachdem Sie alle anderen Zugriffe auf den Chip ausgeführt haben.
- Falls Sie einen durch Lock-Bits gesperrten Chip bearbeiten wollen, führen Sie als erstes ein target\_eraseFlash aus. Dieses setzt auch die Lock-Bits wieder zurück.

#### **Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

#### **Hinweis:**

Manche Änderungen der Fuses sind erst nach einem Reset wirksam bzw. mittels target\_readBits sichtbar. Näheres finden Sie dazu im Handbuch des jeweiligen Targets. Sie können dazu das Kommando target\_restart verwenden.

#### **Parameter:**

##### **targetHandle**

Das Target-Handle auf das anzusprechende Target

##### **index**

Gibt an, welche Fuses bzw. Lock-Bits beschrieben werden sollen. Es gibt dazu die Konstanten FUSES\_LOW, FUSES\_HIGH, FUSES\_EXT und LOCK\_BITS.

Bei Controllern, die keine Extended-Fuses haben, findet bei FUSES\_EXT kein Schreibvorgang statt (es wird keine Exception erzeugt).

#### **value**

Zu schreibende Fuse-Bit-Kombination als Byte.

#### **Rückgabewert:**

- keiner (Prozedur)

#### **Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
apiValueRange	Unzulässiger Wert für index oder value
invalidHandle	Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für einen der Parameter.

#### **Beispiel:**

Beispiel für FUSE\_LOW auf einem ATmega8:

Wenn die folgenden Fuse-Bits gesetzt werden sollen

- Bit 1 (CKSEL1)
- Bit 2 (CKSEL2)
- Bit 3 (CKSEL3)
- Bit 4 (SUT0)
- und alle anderen Fuse-Bits in den Low-Fuses gelöscht werden sollen

Dann entspricht das dem Binärwert 00011110, d.h. hexadezimal \$1E. Als <value> ist also \$1E anzugeben.

### **3.8.4 target\_setExtendedAddressMode**

target\_setExtendedAddressMode <targetHandle>, <value>

Für Controller mit 256 kB Flash oder mehr ist der normale Befehlssatz zum Programmieren über das ISP-Interface nicht mehr ausreichend. Es wird dann ein Extended Address Mode benötigt.

Beim Einstellen der Größe des Flash-Speichers (mittels `target_setMemoryMap` mit `memType = flash` und `memProperty = mm_size`) wird der Wert automatisch passend gesetzt.

Mit dieser Funktion kann der Wert überschrieben werden.

#### **Vorbedingung:**

- gültiges `targetHandle`
- das Target muß im `ProgramMode` sein.

#### **Parameter:**

##### **targetHandle**

Das Target-Handle auf das anzusprechende Target

##### **value**

- 0: Extended address mode nicht verwenden
- sonst: Extended address mode verwenden

#### **Rückgabewert:**

- keiner (Prozedur)

#### **Exceptions:**

`targetWrongMode`  
`invalidHandle`

`apiTypeFault`

Target ist nicht im "ProgramMode".  
Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.  
Unzulässiger Typ für einen der Parameter.

## **3.9 Atmel TPI (TPI-Interface)**

Es werden alle Funktionen der Kapitel „Target allgemein“ bis „Target löschen, schreiben, lesen und verifizieren“ inklusive aller Unterkapitel unterstützt.

Es wird kein Loader verwendet.

**MemTypes:**

Unterstützte memTypes beim Schreiben:

- FLASH

Unterstützte memTypes beim Lesen:

- FLASH

### **3.9.1 target\_getDeviceId**

`s = target_getDeviceId(<targetHandle>)`

Liest die Signature / Device ID des Targets. Anhand dieser lassen sich die verschiedenen Controller unterscheiden.

**Hinweis:**

In den Dokumenten des Herstellers werden über die verschiedenen Controller die Begriffe „Device ID“ und „Signature“ benutzt. Unabhängig davon wird bei roloFlash immer von einer Device ID ausgegangen.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das targetHandle auf das anzusprechende Target

**Rückgabewert:**

Ausgelesene Device ID bzw. Signature. Die Device ID wird in einem Byte-Array mit 3 Bytes geliefert. Sie können die Device ID mit einer Device ID aus der Datenbank vergleichen.



### **Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für Target-Handle.

### **3.9.2 target\_readBits**

value = target\_readBits(<targetHandle>, <index>)

Liest Fuse- bzw. Lock-Bits als Byte aus.

### **Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

### **Parameter:**

#### **targetHandle**

Das Target-Handle auf das anzusprechende Target

#### **index**

0: Fuse Byte 0 bzw. Configuration Byte

**Lock-Bits:** für die Lockbits

### **Rückgabewert:**

Ausgelesene Fuse- bzw. Lock-Bits als Byte.

### **Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
apiValueRange	Unzulässiger Wert für index.
invalidHandle	Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für einen der Parameter.

### 3.9.3 target\_writeBits

target\_writeBits <targetHandle>, <index>, <value>

Schreibt die angegebenen Fuse- bzw. Lock-Bits.

#### **Vorsicht:**

- Es kann Kombinationen geben, die Ihren Chip unbrauchbar machen. Achten Sie bitte auf die Werte und prüfen Sie diese im Handbuch des Controllers.
- Setzen Sie die Lock-Bits erst, nachdem Sie alle anderen Zugriffe auf den Chip ausgeführt haben.
- Falls Sie einen durch Lock-Bits gesperrten Chip bearbeiten wollen, führen Sie als erstes ein target\_eraseFlash aus. Dieses setzt auch die Lock-Bits wieder zurück.

#### **Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

#### **Hinweis:**

Manche Änderungen der Fuses sind erst nach einem Reset wirksam bzw. mittels target\_readBits sichtbar. Näheres finden Sie dazu im Handbuch des jeweiligen Targets. Sie können dazu das Kommando target\_restart verwenden.

#### **Parameter:**

##### **targetHandle**

Das Target-Handle auf das anzusprechende Target

##### **index**

0: Fuse Byte 0 bzw. Configuration Byte

**Lock-Bits:** für die Lockbits

**value**

Zu schreibende Fuse-/Lock-Bit-Kombination als Byte.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
apiValueRange	Unzulässiger Wert für index oder value
invalidHandle	Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für einen der Parameter.

### 3.10 Target Atmel PDI (PDI-Interface)

Es werden alle Funktionen der Kapitel „Target allgemein“ bis „Target löschen, schreiben, lesen und verifizieren“ inklusive aller Unterkapitel unterstützt.

Es wird kein Loader verwendet.

**MemTypes:**

Unterstützte memTypes beim Schreiben:

- FLASH
- EEPROM

Unterstützte memTypes beim Lesen:

- FLASH
- EEPROM

#### 3.10.1 target\_getDeviceId

`id = target_getDeviceId(<targetHandle>)`

Liest die Device ID des Targets. Anhand dieser lassen sich die verschiedenen Controller unterscheiden.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**TargetHandle**

Das Target-Handle auf das anzusprechende Target

**Rückgabewert:**

Ausgelesene Device-ID. Die Device-ID wird in einem Byte-Array mit 3 Bytes geliefert. Sie können die Device-ID mit einer Device-ID aus der Datenbank vergleichen.

**Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
invalidHandle	Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für TargetHandle.

### 3.10.2 target\_readBits

value = target\_readBits(<targetHandle>, <index>)

Liest Fuse- bzw. Lock-Bits als Byte aus.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**index**

- 0: Fuse-Byte 0
- 1: Fuse-Byte 1
- 2: Fuse-Byte 2
- 3: <nicht zulässig>
- 4: Fuse-Byte 4
- 5: Fuse-Byte 5
- 6: <nicht zulässig>
- 7: Lock-Bits

Hinweis: für die Lock-Bits kann auch die Konstante LOCK\_BITS verwendet werden.

**Rückgabewert:**

Ausgelesene Fuse- bzw. Lock-Bits als Byte.

**Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
apiValueRange	Unzulässiger Wert für index.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für einen der Parameter.

### 3.10.3 target\_writeBits

target\_writeBits <targetHandle>, <index>, <value>

Schreibt die angegebenen Fuse- bzw. Lock-Bits.

**Vorsicht:**

- Es kann Kombinationen geben, die Ihren Chip unbrauchbar machen. Achten Sie bitte auf die Werte und prüfen Sie diese im Handbuch des Controllers.
- Setzen Sie die Lock-Bits erst, nachdem Sie alle anderen Zugriffe auf den Chip ausgeführt haben.
- Falls Sie einen durch Lock-Bits gesperrten Chip bearbeiten wollen, führen Sie als erstes ein `target_eraseFlash` aus. Dieses setzt auch die Lock-Bits wieder zurück.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Hinweis:**

Manche Änderungen der Fuses sind erst nach einem Reset wirksam bzw. mittels `target_readBits` sichtbar. Näheres finden Sie dazu im Handbuch des jeweiligen Targets. Sie können dazu das Kommando `target_restart` verwenden.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**index**

- 0: Fuse-Byte 0
- 1: Fuse-Byte 1
- 2: Fuse-Byte 2
- 3: <nicht zulässig>
- 4: Fuse-Byte 4
- 5: Fuse-Byte 5
- 6: <nicht zulässig>
- 7: Lock-Bits

Hinweis: für die Lock-Bits kann auch die Konstante `LOCK_BITS` verwendet werden.

**values**

Zu schreibende Fuse-/Lock-Bit-Kombination als Byte.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
apiValueRange	Unzulässiger Wert für index oder value
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für einen der Parameter.

### 3.11 Target Atmel UPDI (UPDI-Interface)

Es werden alle Funktionen der Kapitel „Target allgemein“ bis „Target löschen, schreiben, lesen und verifizieren“ inklusive aller Unterkapitel unterstützt.

Es wird kein Loader verwendet.

**MemTypes:**

Unterstützte memTypes beim Schreiben:

- FLASH
- EEPROM
- USERSIGNATURE

Unterstützte memTypes beim Lesen:

- FLASH
- EEPROM
- USERSIGNATURE

#### 3.11.1 target\_getDeviceId

```
id = target_getDeviceId(<targetHandle>)
```

Liest die Signature / Device ID des Targets. Anhand dieser lassen sich die verschiedenen Controller unterscheiden.

**Hinweis:**

In den Dokumenten des Herstellers werden über die verschiedenen Controller die Begriffe „Device ID“ und „Signature“ benutzt. Unabhängig davon wird bei roloFlash immer von einer Device ID ausgegangen.

**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**TargetHandle**

Das Target-Handle auf das anzusprechende Target

**Rückgabewert:**

Ausgelesene Device ID bzw. Signature. Die Device ID wird in einem Byte-Array mit 3 Bytes geliefert. Sie können die Device ID mit einer Device ID aus der Datenbank vergleichen.

**Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
invalidHandle	Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für TargetHandle.

### **3.11.2 target\_readBits**

`value = target_readBits(<targetHandle>, <index>)`

Liest Fuse- bzw. Lock-Bits als Byte aus.



**Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

**Parameter:**

**targetHandle**

Das Target-Handle auf das anzusprechende Target

**index** (aus Hersteller Dokumentation zu ATtiny417/817)

- 0: WDTCFG
- 1: BODCFG
- 2: OSCCFG
- 3: <nicht zulässig>
- 4: TCD0CFG
- 5: SYSCFG0
- 6: SYSCFG1
- 7: APPEND
- 8: BOOTEND
- 9: <nicht zulässig>
- 10: Lock-Bits

Hinweis: für die Lock-Bits kann auch die Konstante LOCK\_BITS verwendet werden.

**Rückgabewert:**

Ausgelesene Fuse- bzw. Lock-Bits als Byte.

**Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
apiValueRange	Unzulässiger Wert für index.
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für einen der Parameter.

### 3.11.3 target\_writeBits

target\_writeBits <targetHandle>, <index>, <value>

Schreibt die angegebenen Fuse- bzw. Lock-Bits.

#### **Vorsicht:**

- Es kann Kombinationen geben, die Ihren Chip unbrauchbar machen. Achten Sie bitte auf die Werte und prüfen Sie diese im Handbuch des Controllers.
- Setzen Sie die Lock-Bits erst, nachdem Sie alle anderen Zugriffe auf den Chip ausgeführt haben.
- Falls Sie einen durch Lock-Bits gesperrten Chip bearbeiten wollen, führen Sie als erstes ein target\_eraseFlash aus. Dieses setzt auch die Lock-Bits wieder zurück.

#### **Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

#### **Hinweis:**

Manche Änderungen der Fuses sind erst nach einem Reset wirksam bzw. mittels target\_readBits sichtbar. Näheres finden Sie dazu im Handbuch des jeweiligen Targets. Sie können dazu das Kommando target\_restart verwenden.

#### **Parameter:**

##### **targetHandle**

Das Target-Handle auf das anzusprechende Target

**index** (aus Hersteller Dokumentation zu ATtiny417/817)

- 0: WDTCFG
- 1: BODCFG
- 2: OSCCFG
- 3: <nicht zulässig>
- 4: TCD0CFG

- 5: SYSCFG0
- 6: SYSCFG1
- 7: APPEND
- 8: BOOTEND
- 9: <nicht zulässig>
- 10: Lock-Bits

Hinweis: für die Lock-Bits kann auch die Konstante LOCK\_BITS verwendet werden.

### values

Zu schreibende Fuse-/Lock-Bit-Kombination als Byte.

### Rückgabewert:

- keiner (Prozedur)

### Exceptions:

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
apiValueRange	Unzulässiger Wert für index oder value
invalidHandle	Das Target-Handle ist schon geschlossen, oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für einen der Parameter.

## 3.12 Target Atmel AVR32 (aWire-Interface)

Es werden alle Funktionen der Kapitel „Target allgemein“ bis „Target löschen, schreiben, lesen und verifizieren“ inklusive aller Unterkapitel unterstützt.

Es wird kein Loader verwendet.

### MemTypes:

Unterstützte memTypes beim Schreiben:

- FLASH
- RAM

Unterstützte memTypes beim Lesen:

- FLASH

- RAM
- READMEMORY

### 3.12.1 target\_getDeviceId

`id = target_getDeviceId(<targetHandle>)`

Liest die Signature / Device ID des Targets. Anhand dieser lassen sich die verschiedenen Controller unterscheiden.

#### **Hinweis:**

In den Dokumenten des Herstellers werden über die verschiedenen Controller die Begriffe „Device ID“ und „Signature“ benutzt. Unabhängig davon wird bei roloFlash immer von einer Device ID ausgegangen.

#### **Vorbedingung:**

- gültiges Target-Handle
- das Target muß im ProgramMode sein.

#### **Parameter:**

##### **TargetHandle**

Das Target-Handle auf das anzusprechende Target

#### **Rückgabewert:**

Ausgelesene Device ID bzw. Signature. Die Device ID wird in einem Byte-Array mit 4 Bytes geliefert. Sie können die Device ID mit einer Device ID aus der Datenbank vergleichen.

### **Exceptions:**

targetWrongMode	Target ist nicht im "ProgramMode".
targetCommunication	Die Kommunikation mit dem Target funktioniert nicht.
invalidHandle	Das Target-Handle ist schon geschlossen oder der Bus wurde schon geschlossen.
apiTypeFault	Unzulässiger Typ für TargetHandle.

---

## **4 Flash-Data**

Der Begriff "Flash-Data" bezeichnet einen internen Speicherbereich in roloFlash, auf den der Benutzer wie folgt zugreifen kann:

- mit Dateisystem-Funktionen (fs\_...)
- mit darauf spezialisierten Flash-Data-Funktionen (fd\_...)

Dieser Speicherbereich (Flash-Data) ist in zwei Teilbereiche untergliedert:

- Flash-Disk (Konstante: FLASHDISK)
- Flash-Vars (Konstante: FLASHVARS)  
(Vars als Abkürzung von "Variables")

welche je nach roloFlash-Variante unterschiedlich ausfallen:

<b>Version</b>	<b>Flash-Disk*</b>	<b>Flash-Vars</b>
roloFlash 2	640 kByte singleBuffer - oder - 256 kByte doubleBuffer	16 kByte doubleBuffer
roloFlash 2 AVR	640 kByte singleBuffer - oder - 256 kByte doubleBuffer	16 kByte doubleBuffer

\*Flash-Disk: Die angegebene Größe bezieht sich auf dieses Release. Es ist möglich, daß es andere Firmware für roloFlash geben wird, die auf Kosten der Flash-Disk andere Features implementieren. Die Flash-Vars sind davon nicht betroffen.

## Technische Unterschiede zwischen Flash-Disk und Flash-Vars

- die Größe
- Flash-Vars lassen sich nicht auf singleBuffer umstellen

## Einsatz als Filesystem

In beiden Speichersystemen können Sie Dateien anlegen, lesen, schreiben, so wie es bei einem Dateisystem üblich ist. Die entsprechenden Funktionen sind im nächsten Kapitel "Dateien" beschrieben. Die Flash-Disk /Flash-Vars verhalten sich dabei weitgehend wie eine SD-Karte.

Wenn die RUN\_V07.BIN-Datei und die auf das Target zu flashenden Daten in der Flash-Disk abgelegt sind, können Sie auf den Einsatz der SD-Karte komplett verzichten. Ein vorhandene RUN\_V07.BIN-Datei in der Flash-Disk wird vorrangig einer eventuell vorhandenen RUN\_V07.BIN auf der SD-Karte behandelt.

## Weitere Funktionen

Zusätzlich können Sie hier auch unter beliebigen IDs beliebige Werte speichern.

Vorgaben für IDs:

- Zahlen
- array of char (Zeichenketten, ohne Einschränkung der verwendeten Zeichen), maximal 511 Zeichen, case-sensitiv, '/' ist unterschiedlich zu '\'

Werte können sein:

- Zahlen
- array of char, array of int oder array of long (außer vari-Array)

## CRC

Desweiteren können Werte mit einem CRC32 abgesichert werden. Beim Auslesen wird dieses dann automatisch geprüft.

### Interne Darstellung

Die Bereiche Flash-Disk und Flash-Vars werden im internen Flashspeicher des roloFlash abgelegt. Dieser interne Flashspeicher ist in Sektoren organisiert, die sich nur komplett löschen lassen und dabei altern. Der Hersteller des Chips garantiert dabei 10.000 Löschkzyklen.

Daher wurde Wert darauf gelegt, die Daten möglichst effizient zu speichern, um damit das Löschen eines Sektors so selten wie möglich auszulösen.

### Inkrementtyp

Häufig müssen Werte gespeichert werden, die immer wieder um eins hoch- oder runtergezählt werden. Zur besonders effizienten Speicherung solcher Werte kann bei Anlegen die Größe eines Bitfeldes angegeben werden. Dann kann ein Hochzählen oder Runterzählen um eins durch das Löschen eines einzigen Bits intern gespeichert werden. Damit kann z.B. ein Zähler von bereits durchgeführten Flashvorgängen extrem effizient realisiert werden.

### DoubleBuffer versus SingleBuffer

Im **Doublebuffer-Verfahren** wird der Speicher in zwei Blöcke aufgeteilt, bei denen einer immer frei ist. Es steht daher weniger Speicher zu Verfügung. Allerdings wird hier der Platz von gelöschten oder überschriebenen Daten wieder freigegeben. Zu löschende Bereiche werden intern markiert. Wenn dann eine Schreiboperation nicht mehr ausführbar ist, werden die Daten in den anderen Block kopiert und damit der Platz der gelöschten bzw. überschriebenen Daten zurückgewonnen. Anschließend wird die Schreiboperation durchgeführt. Vor dem Wechsel des Blocks wird außerdem geprüft, ob die Schreiboperation nach einem Wechsel möglich sein wird. Falls nicht, wird der Wechsel nicht durchgeführt, daher er nicht zum Ziel führt.

Im **Singlebuffer-Verfahren** ist der gesamte Speicher in einem Block untergebracht. Daher kann der Speicher von gelöschten oder überschriebenen Daten nicht mehr freigegeben werden. (außer mit `fd_format`). Jedoch wird auch im Singlebuffer-Verfahren solange wie möglich doch intern das Doublebuffer-Verfahren verwendet. Somit kann bis zu der Grenze, an der die Daten nicht mehr im Doublebuffer-Verfahren gespeichert werden können, doch gelöschter bzw. überschriebener Speicher freigegeben werden.

### Simulation der Funktionen für Flash-Data für die SD-Karte

Für viele Funktionen und Prozeduren, die in diesem Kapitel beschrieben werden, wird bei Verwendung der SD-Karte ein ähnliches Verhalten simuliert.

Damit ergibt sich die Möglichkeit, Abläufe erst mit der SD-Karte zu testen und anschließend in der Flash-Disk / Flash-Vars einzusetzen.

#### <FileName> und <id>

Bei vielen Funktionen für Dateizugriffe (beginnend mit "`fs_`", siehe nächstes Kapitel) wird ein Parameter `<fileName>` für den Dateinamen benötigt.

Dieser entspricht dem Parameter `<id>` bei vielen Funktionen für Flash-Data (beginnend mit "`fd_`" in diesem Kapitel).

Wenn Sie sich bei der Vergabe einer ID an die Vorgaben eines Dateinames halten, können Sie mit Funktionen für Dateien und Flash-Data auf die Daten zugreifen.

### Atomare Operation

Viele Funktionen sind atomar: Auch wenn an beliebiger Stelle während der Abarbeitung einer Funktion, die als "atomar" gekennzeichnet ist, die Versorgung des roloFlashs ausfällt, ist folgendes sichergestellt:

- Die Daten wurden komplett übernommen
- oder
- Die Daten wurden überhaupt nicht übernommen (keine Änderung)

Dieses gilt ausschließlich für Flash-Data, nicht für SD-Karte.



## Einsatzzweck Flash-Disk versus Flash-Vars

- Einsatzzweck Flash-Disk: Sollte vorzugsweise für größere und nicht häufig zu ersetzende Dateien wie Images zum Flashen oder auch die RUN\_V07.BIN verwendet werden.
- Einsatzzweck Flash-Vars: Sollte vorzugsweise für kleine Informationen, wie Zähler etc. verwendet werden.

### 4.1 fd\_write

`fd_write <fileSystem>, <id>, <data>, <countingBytes>, <crcMode>`

Schreibt unter der angegebenen id die Daten. Sollten unter dieser id schon Daten gespeichert sein, dann werden sie ersetzt. Mit CountingBytes kann die Größe eines Bitfeldes angegeben werden. Es kann ein CRC gesetzt werden.

#### **Atomar:**

- ja (Verhalten SD-Karte unspezifiziert): Die Daten werden bei einem Stromausfall entweder komplett oder gar nicht übernommen.

#### **Vorbedingung:**

- keine

#### **Parameter:**

##### **fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD:** Es wird eine Datei auf der SD-Karte angelegt, in der diese Information gespeichert wird. Das verwendete Dateiformat ist spezifisch für roloFlash. Das für den Inkrementtypen bei Flash-Data vorgesehene Bitfeld wird nicht angelegt, sondern nur in seiner Funktion simuliert.

**FLASHVARS**  
**FLASHDISK**

### **id**

Eine Zahl oder ein array of Byte. Sollte als <fileSystem> SDCARD angegeben sein, muss id einem gültigen Dateinamen für die SD-Karte entsprechen.

### **data**

Eine Zahl oder ein array of Byte, array of Int oder array of Long. Bei Array of Byte können anschließend auch Funktionen (mit "fs\_" beginnend) für Dateien eingesetzt werden.

### **countingBytes**

(optional, Werte größer 0 sind nur zulässig, wenn <data> eine Zahl ist, Default = 0) Bei Werten größer 0 wird ein Inkrementtyp angelegt und (mindestens) die Anzahl an Bytes für ein Bitfeld reserviert. Bei erneuten Aufrufen dieser Funktion muss der Parameter nicht mehr angegeben werden oder er sollte den gleichen Wert haben. In diesem Fall wird versucht, den neuen Wert durch Löschen von Bits aus dem Bitfeld darzustellen. Sollte dieses nicht möglich sein, wird der Inkrementtyp erneut angelegt. Sie müssen sich daher nicht darum kümmern, ob/wann das Bitfeld nicht mehr ausreichend ist.

Mit dem Inkrementtyp können Werte, die häufig inkrementiert oder dekrementiert werden, extrem effizient intern gespeichert werden (z.B. Flashzyklenzähler).

### **crcMode**

(optional, Default = NOCRC)

Mögliche Werte sind:

**NOCRC**

**USECRC** (bei Inkrementtyp nicht anwendbar): Es wird eine CRC32-Prüfsumme berechnet und mit gespeichert.

### **Rückgabewert:**

- keiner (Prozedur)

### **Exceptions:**

<diverse Exceptions des Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

## 4.2 fd\_createArray

```
fd_createArray <fileSystem>, <id>, <size>, <type>,  
<crcMode>
```

Erzeugt ein leeres Array vom angegebenen Typ und Größe. Es kann ein CRC vorgesehen (nicht gesetzt) werden. Das Array ist mit Nullen initialisiert.

### **Atomar:**

- ja (Verhalten SD-Karte unspezifiziert): Das Array wird bei einem Stromausfall entweder komplett oder gar nicht angelegt.

### **Vorbedingung:**

- keine

### **Parameter:**

#### **fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD:** Es wird eine Datei auf der SD-Karte angelegt, in der diese Information gespeichert wird. Das verwendete Dateiformat ist spezifisch für roloFlash.

**FLASHVARS**

**FLASHDISK**

#### **id**

Eine Zahl oder ein array of Byte. Sollte als <fileSystem> SDCARD angegeben sein, muss id einem gültigen Dateinamen für die SD-Karte entsprechen.

#### **size**

Größenangabe in Elementen.

#### **type**

**CHAR:** Es wird ein Array of Char angelegt.

**INT:** Es wird ein Array of Int angelegt.

**LONG:** Es wird ein Array of Long angelegt

#### **crcMode**

(optional, Default = NOCRC)

Mögliche Werte sind:

##### **NOCRC**

**PLANNEDCRC:** Da das Array noch nicht beschrieben ist, kann der CRC noch nicht berechnet werden. Es wird hier der Platz für einen späteren CRC reserviert. Der CRC kann später durch `fd_setCrc` berechnet und gespeichert werden.

#### **Rückgabewert:**

- keiner (Prozedur)

#### **Exceptions:**

<diverse Exceptions des  
Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

### **4.3 fd\_writeArrayElem**

`fd_writeArrayElem <fileSystem>, <id>, <index>, <data>`

Überschreibt in dem unter der angegebenen id vorhandene Array an der Position index ein Datum.

Es können dabei nur Bits gesetzt werden. Andernfalls wird eine Exception (`flashWriteError`) gesetzt und die Daten bleiben unverändert. Wurde das Array mit `fd_createArray` erzeugt, sind initial alle Werte auf null gesetzt. Damit ist garantiert, daß sich jede Position mindestens einmal beschreiben lässt.

#### **Atomar:**

- ja (Verhalten SD-Karte unspezifiziert): Das Datum wird bei einem Stromausfall entweder komplett oder gar nicht übernommen.

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD:** Es wird in der auf der SD-Karte angelegten Datei die Änderung gespeichert. Das verwendete Dateiformat ist spezifisch für roloFlash.

**FLASHVARS**

**FLASHDISK**

**id**

Eine Zahl oder ein array of Byte. Sollte als <fileSystem> SDCARD angegeben sein, muss id einem gültigen Dateinamen für die SD-Karte entsprechen.

**index**

Die Position, an der das Datum geschrieben werden soll.

**data**

Eine Zahl, die in den Wertebereich des Arraytypen passt.

**Hinweis zu CRC:**

Da hierdurch das Array verändert wird, darf ein CRC nicht gesetzt sein. Das Array muss daher entweder mit fd\_write ohne CRC oder mit fd\_createArray erzeugt worden sein. Die Prozedur fd\_setCrc darf noch nicht aufgerufen worden sein.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

<diverse Exceptions des Dateisystems>      Siehe Kapitel „Exceptions des Dateisystems“.

## 4.4 fd\_writeSubArray

fd\_writeSubArray <fileSystem>, <id>, <index>, <data>

Überschreibt in dem unter der angegebenen id vorhandene Array ab der Position index Daten.

Es können dabei nur Bits gesetzt werden. Andernfalls wird eine Exception (flashWriteError) gesetzt, in diesem Fall bleiben die Daten komplett unverändert. Wurde das Array mit fd\_createArray erzeugt, sind initial alle Werte auf null gesetzt. Damit ist garantiert, daß sich jede Position mindestens einmal beschreiben lässt.

**Atomar:**

- nein (Verhalten SD-Karte unspezifiziert): Bei einem Stromausfall kann es passieren, daß das Array nur teilweise übernommen wurde.

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind

**SDCARD:** Es wird in der auf der SD-Karte angelegten Datei die Änderung gespeichert. Das verwendete Dateiformat ist spezifisch für roloFlash.

**FLASHVARS**

**FLASHDISK**

**id**

Eine Zahl oder ein array of Byte. Sollte als <fileSystem> SDCARD angegeben sein, muss id einem gültigen Dateinamen für die SD-Karte entsprechen.

**index**

Die Position, ab der die Daten geschrieben werden sollen.

**data**

Ein Array vom selben Typ.

**Hinweis zu CRC:**

Da hierdurch das Array verändert wird, darf ein CRC nicht gesetzt sein. Das Array muss daher entweder mit fd\_write ohne CRC oder mit fd\_createArray erzeugt worden sein. Die Prozedur fd\_setCrc darf noch nicht aufgerufen worden sein.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

<diverse Exceptions des Dateisystems>      Siehe Kapitel „Exceptions des Dateisystems“.

## 4.5 fd\_read

data = fd\_read(<fileSystem>, <id>, <crcMode>)

Liest die unter der angegebenen id gespeicherten Daten.

**Atomar:**

- Reine Lesefunktion

**Vorbedingung:**

- Unter der id müssen Daten vorhanden sein. Bei SD-Karte müssen die Daten im roloFlash-spezifischen Format vorliegen, also mittels fd\_write bzw. fd\_createArray erzeugt worden sein.

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Funktion ausgeführt werden soll. Mögliche Werte sind:

**SDCARD:** Es wird eine Datei auf der SD-Karte gelesen, die von roloFlash mittels fd\_write bzw. fd\_createArray angelegt wurde, in der diese Information gespeichert ist.

**FLASHVARS**

**FLASHDISK**

**id**

Eine Zahl oder ein array of Byte.

Sollte als <fileSystem> SDCARD angegeben sein, muss id einem gültigen Dateinamen für die SD-Karte entsprechen.

**crcMode**

(optional, Default = TRYCRC)

Mögliche Werte sind:

**NOCRC:** CRC muss nicht vorhanden sein. Falls vorhanden, wird er ignoriert.

**TRYCRC:** CRC muss nicht vorhanden sein. Falls vorhanden ist, wird er ausgewertet.

**USECRC:** CRC muss vorhanden sein und wird ausgewertet.

**Rückgabewert:**

- die angeforderten Daten

**Exceptions:**

<diverse Exceptions des  
Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.



## 4.6 fd\_readArrayElem

```
value = fd_readArrayElem(<fileSystem, <id>, <index>,  
<crcMode>)
```

Liest ein Element aus dem unter der angegebenen id gespeicherten Array.

### Atomar:

- Reine Lesefunktion

### Vorbedingung:

- Unter der id müssen Daten vorhanden sein. Bei SD-Karte müssen die Daten im roloFlash-spezifischen Format vorliegen, also mittels fd\_write bzw. fd\_createArray erzeugt worden sein.

### Parameter:

#### **fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD:** Es wird eine Datei auf der SD-Karte gelesen, die von roloFlash mittels fd\_write bzw. fd\_createArray angelegt wurde, in der diese Information gespeichert ist.

**FLASHVARS**

**FLASHDISK**

#### **id**

Eine Zahl oder ein array of Byte.

Sollte als <fileSystem> SDCARD angegeben sein, muss id einem gültigen Dateinamen für die SD-Karte entsprechen.

#### **index**

Eine Position innerhalb des Arrays, von der der Wert gelesen werden soll. Der Index muss sich innerhalb des Arrays befinden.

#### **crcMode**

(optional, Default = TRYCRC)

Mögliche Werte sind:

**NOCRC:** CRC muss nicht vorhanden sein. Falls vorhanden, wird er ignoriert.

**TRYCRC:** CRC muss nicht vorhanden sein. Falls vorhanden ist, wird er ausgewertet.

**USECRC:** CRC muss vorhanden sein und wird ausgewertet.

**Rückgabewert:**

- der angeforderte Wert

**Hinweis:**

Unter der angegebenen id müssen Daten von einem der Arraytypen vorhanden sein.

**Exceptions:**

dataTypeError

Versuchter Zugriff auf Daten, die nicht vom Typ eines Arrays sind.

<diverse Exceptions des Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

## 4.7 fd\_readSubArray

```
data = fd_readSubArray(<fileSystem, <id>, <index>, <size>, <crcMode>)
```

Liest einen Bereich aus dem unter der angegebenen id gespeicherten Array.

**Atomar:**

- Reine Lesefunktion

**Vorbedingung:**

- Unter der id müssen Daten vorhanden sein. Bei SD-Karte müssen die Daten im roloFlash-spezifischen Format vorliegen, also mittels fd\_write bzw. fd\_createArray erzeugt worden sein.

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD:** Es wird eine Datei auf der SD-Karte gelesen, die von roloFlash mittels fd\_write bzw. fd\_createArray angelegt wurde, in der diese Information gespeichert ist.

**FLASHVARS**

**FLASHDISK**

**id**

Eine Zahl oder ein array of Byte.

Sollte als <fileSystem> SDCARD angegeben sein, muss id einem gültigen Dateinamen für die SD-Karte entsprechen.

**index**

Eine Position innerhalb des Arrays, von der das Array gelesen werden soll.

**size**

Größe des geforderten Bereiches (Anzahl der Elemente). Der durch index und size gegebene Bereich muss sich innerhalb des Arrays befinden.

**crcMode**

(optional, Default = TRYCRC)

Mögliche Werte sind:

**NOCRC:** CRC muss nicht vorhanden sein. Falls vorhanden, wird er ignoriert.

**TRYCRC:** CRC muss nicht vorhanden sein. Falls vorhanden ist, wird er ausgewertet.

**USECRC:** CRC muss vorhanden sein und wird ausgewertet.

**Rückgabewert:**

- Array mit den geforderten Daten.

**Hinweis:**

Unter der angegebenen id müssen Daten eines Array-Typen vorhanden sein.

**Exceptions:**

dataTypeError

Versuchter Zugriff auf Daten, die nicht vom Typ eines Arrays sind.

<diverse Exceptions des Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

## 4.8 fd\_remove

fd\_remove <fileSystem>, <id>

Löscht die unter der id abgelegten Daten.

**Atomar:**

- ja (Verhalten SD-Karte unspezifiziert): Das Datum wird bei einem Stromausfall entweder komplett gelöscht oder gar nicht.

**Hinweis:**

- die Prozedur entspricht fs\_remove, mit dem Unterschied, daß hier bei id gegebenenfalls auch eine Zahl verwendet werden kann.

**Vorbedingung:**

- SD-Karte: Die über die id angesprochene Datei darf nicht geöffnet sein.

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD, FLASHVARS und FLASHDISK**

**id**

Eine Zahl oder ein array of Byte. Sollte als <fileSystem> SDCARD angegeben sein, muss id einem gültigen Dateinamen für die SD-Karte entsprechen.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

FileNotFound

fileIsOpen

<diverse Exceptions des  
Dateisystems>

Die angegebene Datei existiert nicht.

Die angegebene Datei ist noch geöffnet.

Siehe Kapitel „Exceptions des Dateisystems“.

## 4.9 fd\_getItemCount

```
count = fd_getItemCount(<fileSystem>)
```

Ermittelt die Anzahl der gespeicherten Elemente

**Atomar:**

- Reine Lesefunktion

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind (SDCARD ist nicht unterstützt):

**FLASHVARS** und **FLASHDISK**

**Rückgabewert:**

- Anzahl der gespeicherten Elemente

**Exceptions:**

<diverse Exceptions des Dateisystems>      Siehe Kapitel „Exceptions des Dateisystems“.

## 4.10 fd\_getId

id = fd\_getId(<fileSystem>, <index>)

Ermittelt die Id des gespeicherten Elementes mit dem gegebenen Index.

**Atomar:**

- Reine Lesefunktion

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind (SDCARD ist nicht unterstützt):  
**FLASHVARS** und **FLASHDISK**

**index**

Index auf das Element, dieser muss im Bereich von 0 bis 1 unter dem Wert, den die Funktion fd\_getItemCount liefert, liegen.

**Rückgabewert:**

- id des gespeicherten Elementes mit dem gegebenen Index

**Hinweis:**

- Zusammen mit der Funktion fd\_getItemCount können alle gespeicherten Elemente angesprochen werden.

**Exceptions:**

<diverse Exceptions des Dateisystems>      Siehe Kapitel „Exceptions des Dateisystems“.

## 4.11 fd\_idExists

```
found = fd_idExists(<fileSystem>, <id>)
```

Ermittelt ob unter der angegebenen Id etwas gespeichert ist.

**Atomar:**

- Reine Lesefunktion

**Hinweis:**

- die Funktion entspricht fs\_fileExists mit dem Unterschied, daß hier bei id gegebenenfalls auch eine Zahl verwendet werden kann.

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD, FLASHVARS und FLASHDISK**

**id**

Eine Zahl oder ein array of Byte. Sollte als <fileSystem> SDCARD angegeben sein, muss id einem gültigen Dateinamen für die SD-Karte entsprechen.

**Rückgabewert:**

0 = Datei existiert nicht

1 = Datei existiert

**Exceptions:**

<diverse Exceptions des Dateisystems>      Siehe Kapitel „Exceptions des Dateisystems“.

## 4.12 fd\_isArray

```
isArray = fd_isArray(<fileSystem>, <id>)
```

Ermittelt ob unter der angegeben Id ein Array gespeichert ist.

**Atomar:**

- Reine Lesefunktion

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD:** Es wird eine Datei auf der SD-Karte gelesen, die von roloFlash mittels fd\_write bzw. fd\_createArray angelegt wurde, in der diese Information gespeichert ist.

**FLASHVARS** und **FLASHDISK**

**id**

Eine Zahl oder ein array of Byte. Sollte als <fileSystem> SDCARD angegeben sein, muss id einem gültigen Dateinamen für die SD-Karte entsprechen.



**Rückgabewert:**

0 = kein Array (Zahl oder Inkrementtyp)

1 = Array of char, int oder long

**Exceptions:**

<diverse Exceptions des Dateisystems>      Siehe Kapitel „Exceptions des Dateisystems“.

## 4.13 fd\_getArraySize

```
size = fd_getArraySize(<fileSystem>, <id>
```

Ermittelt unter für das unter der angegeben Id gespeicherten Array die Anzahl der Elemente.

**Atomar:**

- Reine Lesefunktion

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD:** Es wird eine Datei auf der SD-Karte gelesen, die von roloFlash mittels fd\_write bzw. fd\_createArray angelegt wurde, in der diese Information gespeichert ist.

**FLASHVARS** und **FLASHDISK**

**id**

Eine Zahl oder ein array of Byte. Sollte als <fileSystem> SDCARD angegeben sein, muss id einem gültigen Dateinamen für die SD-Karte entsprechen.

**Rückgabewert:**

Anzahl der Elemente

**Hinweis:**

Unter der angegebenen id müssen Daten eines Array-Typen vorhanden sein.

**Exceptions:**

dataTypeError

Versuchter Zugriff auf Daten, die nicht vom Typ eines Arrays sind.

<diverse Exceptions des Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

## 4.14 fd\_getType

```
myType = fd_getType(<fileSystem>, <id>)
```

Ermittelt den Typ der unter id gespeicherten Zahl oder den Typ des darunter gespeicherten Arrays.

- Unter der id ist eine Zahl gespeichert:  
char, int oder long, je nach aktuellem Wert der Zahl
- Unter der id ist ein Array gespeichert:  
char, int oder long, je nach Typ der Elemente des Arrays

**Atomar:**

- Reine Lesefunktion

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD**: Es wird eine Datei auf der SD-Karte gelesen, die von roloFlash mittels `fd_write` bzw. `fd_createArray` angelegt wurde, in der diese Information gespeichert wird.

**FLASHVARS** und **FLASHDISK**

#### **id**

Eine Zahl oder ein array of Byte. Sollte als `<fileSystem>` SDCARD angegeben sein, muss `id` einem gültigen Dateinamen für die SD-Karte entsprechen.

#### **Rückgabewert:**

- Typ der Daten, bzw. Typ der Elemente bei einem Array.

#### **Exceptions:**

`<diverse Exceptions des Dateisystems>`

Siehe Kapitel „Exceptions des Dateisystems“.

## **4.15 fd\_getCountingBytes**

```
count = fd_getCountingBytes(<fileSystem>, <id>)
```

Gibt beim Inkrementtyp an, wieviele Bytes für das Bitfeld reserviert sind. Die Zahl kann leicht größer sein als ursprünglich angefordert. Bei allen anderen Typen wird 0 zurückgegeben.

#### **Atomar:**

- Reine Lesefunktion

#### **Vorbedingung:**

- keine

#### **Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD:** Es wird eine Datei auf der SD-Karte gelesen, die von roloFlash mittels `fd_write` bzw. `fd_createArray` angelegt wurde, in der diese Information gespeichert wird.

**FLASHVARS** und **FLASHDISK**

#### **id**

Eine Zahl oder ein array of Byte. Sollte als `<fileSystem>` SDCARD angegeben sein, muss `id` einem gültigen Dateinamen für die SD-Karte entsprechen.

#### **Rückgabewert:**

- Anzahl der Bytes für das Bitfeld bei Inkrementtyp.

#### **Exceptions:**

`<diverse Exceptions des Dateisystems>`

Siehe Kapitel „Exceptions des Dateisystems“.

### **4.16 fd\_setCrc**

`fd_setCrc <fileSystem>, <id>`

Setzt den CRC. Dieser musste vorab bei `fd_createArray` mittels `PLANNEDCRC` reserviert worden sein. Sollte der CRC schon gesetzt sein und nicht stimmen, dann wird eine Exception ausgelöst.

#### **Atomar:**

- ja (Verhalten SD-Karte unspezifiziert): Der CRC wird bei einem Stromausfall entweder gesetzt oder bleibt ungesetzt.

#### **Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD:** Es wird eine Datei auf der SD-Karte gelesen, die von roloFlash mittels fd\_write bzw. fd\_createArray angelegt wurde, in der diese Information gespeichert ist.

**FLASHVARS und FLASHDISK**

**id**

Eine Zahl oder ein array of Byte. Sollte als <fileSystem> SDCARD angegeben sein, muss id einem gültigen Dateinamen für die SD-Karte entsprechen.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

<diverse Exceptions des Dateisystems>      Siehe Kapitel „Exceptions des Dateisystems“.

## 4.17 fd\_getCrc

```
crc = fd_getCrc(<fileSystem>, <id>)
```

Liest den gespeicherten CRC aus. Falls dieser noch nicht gesetzt ist, wird eine 0 ausgegeben.

**Atomar:**

- Reine Lesefunktion

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD:** Es wird eine Datei auf der SD-Karte gelesen, die von roloFlash mittels `fd_write` bzw. `fd_createArray` angelegt wurde, in der diese Information gespeichert ist.

**FLASHVARS** und **FLASHDISK**

**id**

Eine Zahl oder ein array of Byte. Sollte als `<fileSystem>` SDCARD angegeben sein, muss `id` einem gültigen Dateinamen für die SD-Karte entsprechen.

**Rückgabewert:**

- bei gesetztem CRC: Wert des gespeicherten CRC
- sonst: 0

**Exceptions:**

`<diverse Exceptions des Dateisystems>`      Siehe Kapitel „Exceptions des Dateisystems“.

## 4.18 `fd_calcCrc`

```
crc = fd_calcCrc(<fileSystem>, <id>)
```

Berechnet den CRC für die gespeicherten Daten. Dies kann auch ausgeführt werden, wenn kein CRC geplant war.

**Atomar:**

- Reine Lesefunktion

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD:** Es wird eine Datei auf der SD-Karte gelesen, die von roloFlash mittels `fd_write` bzw. `fd_createArray` angelegt wurde, in der diese Information gespeichert ist.

**FLASHVARS** und **FLASHDISK**

**id**

Eine Zahl oder ein array of Byte. Sollte als `<fileSystem>` SDCARD angegeben sein, muss `id` einem gültigen Dateinamen für die SD-Karte entsprechen.

**Rückgabewert:**

- berechneter CRC

**Exceptions:**

`<diverse Exceptions des Dateisystems>`

Siehe Kapitel „Exceptions des Dateisystems“.

## 4.19 `fd_hasCrc`

`hasCrc = fd_hasCrc(<fileSystem>, <id>)`

Gibt an, ob für die gespeicherten Daten ein CRC vorgesehen ist.

**Atomar:**

- Reine Lesefunktion

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD:** Es wird eine Datei auf der SD-Karte gelesen, die von roloFlash mittels `fd_write` bzw. `fd_createArray` angelegt wurde, in der diese Information gespeichert ist.

**FLASHVARS** und **FLASHDISK**

**id**

Eine Zahl oder ein array of Byte. Sollte als `<fileSystem>` SDCARD angegeben sein, muss `id` einem gültigen Dateinamen für die SD-Karte entsprechen.

**Rückgabewert:**

0 = kein CRC vorgesehen

1 = CRC vorgesehen

**Exceptions:**

`<diverse Exceptions des Dateisystems>`

Siehe Kapitel „Exceptions des Dateisystems“.

## 4.20 `fd_getFreeMem`

`size = fd_getFreeMem(<fileSystem>)`

Gibt an, wieviel Speicher noch zur Verfügung steht.



**Hinweis:**

Das Verhalten unterscheidet sich je nach Double- versus Singlebuffer-Verfahren: Bei Doublebuffer-Verfahren, und im Singlebuffer-Verfahren soweit dieses noch möglich ist, steht der Speicher von gelöschten oder überschriebenen Daten sofort wieder zur Verfügung.

**Atomar:**

- Reine Lesefunktion

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Funktion ausgeführt werden soll. Mögliche Werte sind (SDCARD ist nicht unterstützt):  
**FLASHVARS** und **FLASHDISK**

**Rückgabewert:**

Größe des freien Speichers in Bytes

**Exceptions:**

<diverse Exceptions des Dateisystems>      Siehe Kapitel „Exceptions des Dateisystems“.

## 4.21 fd\_getBytesWritten

```
count = fd_getBytesWritten(<fileSystem>)
```

Gibt an, wieviele Bytes im Speicher im aktuellen Powercycle geschrieben wurden. Damit kann beurteilt werden, wie sich die Vorgänge auf die Alterung des Speichers auswirken werden.

Z.B. ergibt sich hier bei der Verwendung des Inkrementtyps, dass ein Erhöhen oder Erniedrigen keinen Einfluss auf diese Funktion hat, sofern das Bitfeld die Änderung noch aufnehmen kann.

**Atomar:**

- Reine Lesefunktion

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Funktion ausgeführt werden soll. Mögliche Werte sind (SDCARD ist nicht unterstützt):

**FLASHVARS** und **FLASHDISK**

**Rückgabewert:**

Anzahl der in diesem Powercycle geschrieben Bytes.

**Exceptions:**

<diverse Exceptions des Dateisystems>      Siehe Kapitel „Exceptions des Dateisystems“.

## 4.22 fd\_setSingleBufferMode

fd\_setSingleBufferMode <fileSystem>, <value>

Stellt den Speicher auf Doublebuffering oder Singlebuffering um. Dieses geschieht ohne Datenverlust. Sollte es ohne Datenverlust nicht möglich sein, dann gibt es eine Exception. In diesem Fall müssen erst Daten gelöscht werden, gegebenenfalls muss formatiert werden.

**Atomar:**

- Ja, Entweder die Umschaltung erfolgt komplett oder gar nicht. Es kann auch sein, daß sie beim nächsten Einschalten erst noch abgeschlossen wird.

**Vorbedingung:**

- Genug Speicher zur Verfügung um die Daten entsprechend um zu organisieren.

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Möglicher Wert ist (FLASHVARS und SDCARD sind nicht unterstützt):

**FLASHDISK**

**value**

Bool Wert:

- 0: Der Speicher wird auf Doublebuffering umgestellt
- 1: Der Speicher wird auf Singlebuffering umgestellt

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

<diverse Exceptions des Dateisystems>      Siehe Kapitel „Exceptions des Dateisystems“.

## 4.23 fd\_getSingleBufferMode

value = fd\_getSingleBufferMode(<fileSystem>)

Ermittelt, ob der Speicher im Single- oder Doublebuffermode betrieben wird.

**Atomar:**

- Reine Lesefunktion

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Funktion ausgeführt werden soll. Mögliche Werte sind (SDCARD ist nicht unterstützt):

**FLASHVARS** und **FLASHDISK**

**Rückgabewert:**

0 = Doublebuffer-Verfahren

1 = Singlebuffer-Verfahren

**Exceptions:**

<diverse Exceptions des  
Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

## 4.24 fd\_cleanup

fd\_cleanup <fileSystem>

Sorgt dafür, dass gelöschte oder überschriebene Werte im Speicher wirklich physikalisch gelöscht sind (Security-Feature).

**Atomar:**

- Nein: Bei einer Unterbrechung in der Stromversorgung kann es sein, daß erst ein Teil der obsoleten Daten überschrieben wurde.

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Funktion ausgeführt werden soll. Mögliche Werte sind (SDCARD ist nicht unterstützt):  
**FLASHVARS** und **FLASHDISK**

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

<diverse Exceptions des Dateisystems>      Siehe Kapitel „Exceptions des Dateisystems“.

## 4.25 fd\_format

fd\_format <fileSystem>

Löscht alle Daten. Die Daten werden dabei auch im Speicher wirklich physikalisch gelöscht sind (Security-Feature).

**Atomar:**

- Nein: Bei einer Unterbrechung in der Stromversorgung kann es sein, daß erst ein Teil der obsoleten Daten überschrieben wurde.

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Funktion ausgeführt werden soll. Mögliche Werte sind (SDCARD ist nicht unterstützt):

**FLASHVARS** und **FLASHDISK**

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

<diverse Exceptions des  
Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

---

## 5 Dateien

**Dateisysteme:**

**Es können je nach roloFlash-Version auf folgende Dateisystem zugegriffen werden.**

- SD-Karte (Konstante: SDCARD)
- Flash-Vars (Konstante: FLASHVARS)
- Flash-Disk (Konstante: FLASHDISK)

### **Vorgaben für Dateinamen für die SD-Karte:**

- Dateinamen müssen der 8.3-Regel folgen: „XXXXXXXX.YYY“.
- Es sind nur die Zeichen „A“ - „Z“, „0“ - „9“, sowie „\_“ und „-“ zulässig.
- Es dürfen nur Großbuchstaben verwendet werden.

### **Vorgaben für Verzeichnisnamen für die SD-Karte:**

- Verzeichnisnamen dürfen maximal aus acht Zeichen bestehen: „XXXXXXXX“.
- Ansonsten gelten dieselben Konventionen wie bei Dateinamen.
- '/' (empfohlen) und '\' werden gleich behandelt.

### **Vorgaben für IDs (Dateinamen und Verzeichnisnamen) für Flash-Data:**

- Hier können beliebige Zeichenketten (array of char) oder Zahlen verwendet werden
- Bei Zeichenketten sind alle Zeichen erlaubt (auch 0-Zeichen, Carriage-return und Linefeed), case sensitiv. Es wird zwischen '/' und '\' unterschieden

### **Aktuelles Verzeichnis ist immer das Hauptverzeichnis:**

- Es gibt kein „change directory“. Der aktuelle Pfad bleibt immer das Hauptverzeichnis. Ein Dateiname muß daher immer den kompletten Pfad beinhalten.
- Als Trennzeichen zwischen Verzeichnissen und Dateiname werden „/“ und „\“ unterstützt.

#### **ACHTUNG:**

Wir empfehlen, als Pfadtrenner "/" zu verwenden statt des unter DOS und Windows üblichen "\".

Falls Sie "\" als Pfadtrenner verwenden wollen, müssen Sie jedes Vorkommen von "\" verdoppeln (escapen), also z.B. "mysubdir\\myfile.txt".

Alle Funktionen beginnen mit dem Präfix "fs\_" ("fileSystem").

## 5.1 fs\_mediaExists

```
bool fs_mediaExists(<fileSystem>)
```

Prüft, ob das angegebene Medium existiert.

### Vorbedingung:

- keine

### Parameter:

#### **fileSystem**

Gibt an, auf welchem Dateisystem die Funktion ausgeführt werden soll. Mögliche Werte sind:

**SDCARD**, **FLASHVARS** und **FLASHDISK**

Aus Kompatibilitätsgründen wird auch eine 0 für die SD-Karte akzeptiert.

### Hinweis:

- **SDCARD**: Ermittelt, ob eine SD-Karte eingesteckt ist.
- **FLASHVARS**: Ermittelt, ob der Speicher FLASHVARS existiert.
- **FLASHDISK**: Ermittelt, ob der Speicher FLASHDISK existiert.

### Rückgabewert:

0 = Medium existiert nicht

1 = Medium existiert

### Exceptions:

<diverse Exceptions des  
Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

## 5.2 fs\_create

```
fs_create <fileSystem>, <fileName>, <size>
```



Erzeugt die angegebene Datei. Die Datei ist danach noch immer geschlossen. Falls die Datei schon existiert, hat die Prozedur keine Wirkung.

Wenn man eine Datei erzeugen und in diese etwas schreiben möchte, muß man die Datei zusätzlich noch öffnen, z.B.:

```
fs_create SDCARD, "TEST.TXT"
```

```
handle = fs_open(SDCARD, "TEST.TXT")
```

### **Vorbedingung:**

- keine

### **Parameter:**

#### **fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD**, **FLASHVARS** und **FLASHDISK**

Aus Kompatibilitätsgründen wird auch eine 0 für die SD-Karte akzeptiert.

#### **fileName**

Für SD-Karte: Der Dateiname muss einem gültigen Dateinamen für die SD-Karte entsprechen, siehe „Vorgaben für Dateinamen“.

Für Flash-Data: Der Dateiname muss einer gültigen ID für Flash-Data entsprechen, siehe „Vorgaben für IDs“.

#### **size**

Größe der anzulegenden Datei, die Datei wird mit Nullen gefüllt.

Für SDCARD: Der Parameter ist optional.

Für Flash-Data: Der Parameter ist erforderlich.

### **Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

<diverse Exceptions des Dateisystems>      Siehe Kapitel „Exceptions des Dateisystems“.

## 5.3 fs\_rename

fs\_rename <fileSystem>, <fileNameOld>, <fileNameNew>

Benennt die Datei um.

**Vorbedingung:**

- <fileNameNew> darf vorher nicht existieren.
- Für SD-Karte: Falls in <fileNameOld> und <fileNameNew> Pfade enthalten sind, dann müssen diese gleich sein.  
Für Flash-Data: Keine Einschränkung.

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD, FLASHVARS und FLASHDISK**

Aus Kompatibilitätsgründen wird auch eine 0 für die SD-Karte akzeptiert.

**fileNameOld, fileNameNew**

Für SD-Karte: Der Dateiname muss einem gültigen Dateinamen für die SD-Karte entsprechen, siehe „Vorgaben für Dateinamen“.

Für Flash-Data: Der Dateiname muss einer gültigen ID für Flash-Data entsprechen, siehe „Vorgaben für IDs“.

**Rückgabewert:**

- keiner (Prozedur)

### **Exceptions:**

FileNotFound

Die angegebene Datei existiert nicht.

filesOpen

Die angegebene Datei ist noch geöffnet.

<diverse Exceptions des  
Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

## **5.4 fs\_remove**

`fs_remove <fileSystem>, <fileName>`

Löscht die angegebene Datei oder das angegebene Verzeichnis, falls vorhanden.

### **Vorbedingung:**

- SD-Karte: Die Datei darf nicht geöffnet sein.

### **Parameter:**

#### **fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD, FLASHVARS und FLASHDISK**

Aus Kompatibilitätsgründen wird auch eine 0 für die SD-Karte akzeptiert.

#### **fileName**

Für SD-Karte: Der Dateiname muss einem gültigen Dateinamen für die SD-Karte entsprechen, siehe „Vorgaben für Dateinamen“.

Für Flash-Data: Der Dateiname muss einer gültigen ID für Flash-Data entsprechen, siehe „Vorgaben für IDs“.

### **Rückgabewert:**

- keiner (Prozedur)

### **Exceptions:**

FileNotFound

Die angegebene Datei existiert nicht.

filesOpen

Die angegebene Datei ist noch geöffnet.

<diverse Exceptions des  
Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

## 5.5 fs\_mkDir

fs\_mkDir <fileSystem>, <dirName>

Erzeugt das angegebene Verzeichnis. Falls das Verzeichnis schon existiert, hat die Prozedur keine Wirkung.

### Vorbedingung:

- keine

### Parameter:

#### **fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD** (FLASHVARS und FLASHDISK nicht unterstützt, es wird keine Fehlermeldung erzeugt)

Aus Kompatibilitätsgründen wird auch eine 0 für die SD-Karte akzeptiert.

#### **Hinweis für FLASHVARS und FLASHDISK:**

Sie können stattdessen trotzdem an jeder Stelle, an der Dateinamen bzw. IDs bei Flash-Data verwendet werden, Dateinamen mit "/" verwenden und damit die Funktionalität einer Dateisystem-Hierarchie nachbilden.

#### **dirName**

Für SD-Karte: Der Verzeichnisname muss einem gültigen Verzeichnisname für die SD-Karte entsprechen, siehe „Vorgaben für Dateinamen“.

### Rückgabewert:

- keiner (Prozedur)

**Exceptions:**

<diverse Exceptions des Dateisystems>      Siehe Kapitel „Exceptions des Dateisystems“.

## 5.6 fs\_fileExists

```
bool fs_fileExists(<fileSystem>, <fileName>)
```

Prüft, ob die angegebene Datei existiert.

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Funktion ausgeführt werden soll. Mögliche Werte sind:

**SDCARD, FLASHVARS und FLASHDISK**

Aus Kompatibilitätsgründen wird auch eine 0 für die SD-Karte akzeptiert.

**fileName**

Für SD-Karte: Der Dateiname muss einem gültigen Dateinamen für die SD-Karte entsprechen, siehe „Vorgaben für Dateinamen“.

Für Flash-Data: Der Dateiname muss einer gültigen ID für Flash-Data entsprechen, siehe „Vorgaben für IDs“.

**Rückgabewert:**

0 = Datei existiert nicht

1 = Datei existiert

**Exceptions:**

<diverse Exceptions des Dateisystems>      Siehe Kapitel „Exceptions des Dateisystems“.

## 5.7 fs\_fileSize

```
size = fs_fileSize(<fileSystem>, <fileName>)
```

Ermittelt die Größe der angegebenen Datei.

### Vorbedingung:

- Datei existiert.

### Parameter:

#### **fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD, FLASHVARS und FLASHDISK**

Aus Kompatibilitätsgründen wird auch eine 0 für die SD-Karte akzeptiert.

#### **fileName**

Für SD-Karte: Der Dateiname muss einem gültigen Dateinamen für die SD-Karte entsprechen, siehe „Vorgaben für Dateinamen“.

Für Flash-Data: Der Dateiname muss einer gültigen ID für Flash-Data entsprechen, siehe „Vorgaben für IDs“.

### Rückgabewert:

Es wird die Dateigröße in Bytes zurückgegeben.

### Hinweis:

Bei Zugriff auf FlashVars bzw. FlashDisk müssen unter dem angegebenen Dateinamen Daten vom Typ "array of byte" vorhanden sein.

### Exceptions:

dataTypeError

Versuchter Zugriff auf Daten, die nicht vom Typ "array of bytes" sind ( FlashVars bzw. FlashDisk)

<diverse Exceptions des Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

## 5.8 fs\_open

```
fileHandle = fs_open(<fileSystem>, <fileName>)
```

Öffnet die angegebene Datei.

### **Vorbedingung:**

Die Datei muß bereits existieren. Soll eine neue Datei geöffnet werden, muß vorher `fs_create` verwendet werden.

### **Parameter:**

#### **fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD**, **FLASHVARS** und **FLASHDISK**

Aus Kompatibilitätsgründen wird auch eine 0 für die SD-Karte akzeptiert.

#### **fileName**

Für SD-Karte: Der Dateiname muss einem gültigen Dateinamen für die SD-Karte entsprechen, siehe „Vorgaben für Dateinamen“.

Für Flash-Data: Der Dateiname muss einer gültigen ID für Flash-Data entsprechen, siehe „Vorgaben für IDs“.

### **Rückgabewert:**

Es wird ein Filehandle zum Zugriff auf die Datei (z. B. für `fs_read` und `fs_write`) zurückgegeben. Das Filehandle wird außerdem zum Schließen der Datei (`fs_close`) benötigt.

### **Hinweis:**

Bei Zugriff auf FlashVars bzw. FlashDisk müssen unter dem angegebenen Dateinamen Daten vom Typ array of Byte vorhanden sein.

**Exceptions:**

dataTypeError

Versuchter Zugriff auf Daten, die nicht vom Typ "array of bytes" sind ( FlashVars bzw. FlashDisk)

<diverse Exceptions des Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

## 5.9 fs\_read

```
a = fs_read(<fileHandle>, <position>, <count>)
```

Liest die angegebene Anzahl an Bytes aus der Datei.

**Vorbedingung:**

- Gültiges fileHandle mittels fs\_open.

**Parameter:**

**fileHandle**

Das von fs\_open erhaltene fileHandle.

**position**

Die Byte-Position, von der gelesen werden soll.

**count**

Die Anzahl der zu lesenden Bytes.

**Rückgabewert:**

Array of Byte mit den gelesenen Daten. Das Array hat die Größe von count. Falls nicht mehr genügend Daten zu lesen waren, ist das Array entsprechend kleiner. Wird am Dateiende oder darüber hinaus versucht zu lesen, wird ein leeres Array mit der Größe 0 zurückgegeben.



**Exceptions:**

<code>apiValueRange</code>	Unzulässiger Wert für <code>fileHandle</code> , <code>position</code> oder <code>count</code> .
<code>apiTypeFault</code>	Unzulässiger Typ für <code>fileHandle</code> , <code>position</code> oder <code>count</code> .
<code>&lt;diverse Exceptions des Dateisystems&gt;</code>	Siehe Kapitel „Exceptions des Dateisystems“.

## 5.10 `fs_write`

`fs_write <fileHandle>, <position>, <array>`

Schreibt die übergebenen Daten in die Datei.

Sollte die Position außerhalb der momentanen Dateigröße sein, wird die Datei bis zu dieser Position mit zufälligen Werten aufgefüllt.

**Vorbedingung:**

- Gültiges `fileHandle` mittels `fs_open`.

**Parameter:**

**`fileHandle`**

Das von `fs_open` erhaltene `fileHandle`.

**`position`**

Die Byte-Position, an die geschrieben werden soll.

**`array`**

Array of Byte mit den zu schreibenden Daten.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

<code>apiValueRange</code>	Unzulässiger Wert für <code>fileHandle</code> , <code>position</code> oder <code>count</code> .
<code>apiTypeFault</code>	Unzulässiger Typ für <code>fileHandle</code> , <code>position</code> oder <code>count</code> .
<diverse Exceptions des Dateisystems>	Siehe Kapitel „Exceptions des Dateisystems“.

## 5.11 `fs_truncate`

`fs_truncate` <`fileHandle`>, <`len`>

Kürzt die Datei auf die angegebene Länge. Falls die Datei schon kleiner ist, ist die Prozedur ohne Wirkung.

**Vorbedingung:**

- Gültiges `fileHandle` mittels `fs_open`.

**Parameter:**

**`fileHandle`**

Das von `fs_open` erhaltene `fileHandle`.

**`len`**

Die Länge in Bytes, auf die die Datei gekürzt wird.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

<code>apiValueRange</code>	Unzulässiger Wert für <code>fileHandle</code> .
<code>apiTypeFault</code>	Unzulässiger Typ für <code>fileHandle</code> oder <code>len</code> .
<diverse Exceptions des Dateisystems>	Siehe Kapitel „Exceptions des Dateisystems“.

## 5.12 fs\_close

fs\_close <fileHandle>

Schließt die Datei. Das angegebene Filehandle wird dadurch ungültig und darf nicht mehr verwendet werden.

### **Vorbedingung:**

- Gültiges fileHandle mittels fs\_open.

### **Parameter:**

#### **fileHandle**

Das von fs\_open erhaltene fileHandle.

### **Rückgabewert:**

- keiner (Prozedur)

### **Hinweis:**

Diese Prozedur hat bei Flash-Data keine Auswirkung.

### **Exceptions:**

apiValueRange

apiTypeFault

<diverse Exceptions des  
Dateisystems>

Unzulässiger Wert für fileHandle.

Unzulässiger Typ für fileHandle.

Siehe Kapitel „Exceptions des Dateisystems“.

## 5.13 fs\_sync

fs\_sync <fileSystem>

Stellt sicher, dass alle Daten, die evtl. noch nicht auf die Karte geschrieben wurden, nun geschrieben werden. Es wird empfohlen, wenn Schreibzugriffe auf die Karte stattfinden, diese Prozedur anschließend aufzurufen.

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD, FLASHVARS und FLASHDISK**

Aus Kompatibilitätsgründen wird auch eine 0 für die SD-Karte akzeptiert.

**Rückgabewert:**

- keiner (Prozedur)

**Hinweis:**

Diese Prozedur hat bei Flash-Data keine Auswirkung.

**Exceptions:**

<diverse Exceptions des  
Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

---

## 6 LEDs

**Immer nur eine LED an:**

- Es lässt sich aus roloBasic heraus immer nur maximal 1 LED zum Leuchten bringen, um die Strombelastung des Targets so gering wie möglich zu halten.

### **Numerierung und Farben:**

- Die LEDs sind in roloBasic genauso numeriert wie auf dem Gehäuse abgebildet: von 1 bis 5.
- Die LEDs können in Grün oder Rot leuchten. Dazu stehen die Konstanten `COLOR_GREEN` und `COLOR_RED` zur Verfügung.

### **Nicht blockierend:**

- Alle Prozeduren dieses Kapitels sind nicht blockierend. Das bedeutet, dass z. B. ein mit `led_runningLight` aktiviertes Lauflicht parallel zur weiteren Ausführung des roloBasic läuft.

## **6.1 led\_on**

`led_on <index>, <color>`

Schaltet die LED auf die angegebene Farbe.

### **Vorbedingung:**

- keine

### **Parameter:**

#### **index**

Nummer der LED

#### **color**

`COLOR_GREEN` bzw. `COLOR_RED`

### **Rückgabewert:**

- keiner (Prozedur)

### **Exceptions:**

`apiValueRange`  
`apiTypeFault`

Unzulässiger Wert für index oder color.  
Unzulässiger Typ für index oder color.

## 6.2 led\_off

led\_off

Schaltet alle LEDs aus.

**Vorbedingung:**

- keine

**Parameter:**

- keine

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

- keine

## 6.3 led\_blink

led-blink <index>, <color>, <speed>

Die LED blinkt mit der angegebenen Geschwindigkeit.

**Vorbedingung:**

- keine

**Parameter:**

**index**

Nummer der LED

**color**

COLOR\_GREEN bzw. COLOR\_RED

**speed**

Geschwindigkeit des Blinkens in [ms]

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange  
apiTypeFault

Unzulässiger Wert für index, color oder speed.  
Unzulässiger Typ für index, color oder speed.

## 6.4 led\_runningLight

led\_runningLight <from>, <to>, <color>, <speed>

Läßt ein Lauflicht laufen.

**Vorbedingung:**

- keine

**Parameter:**

**from, to**

Das Lauflicht läuft von LED 'from' bis LED 'to'.  
Ist 'from' kleiner als 'to', läuft das Lauflicht „anders herum“.  
Ist 'from' gleich 'to', leuchtet die eine LED ständig.

**color**

COLOR\_GREEN bzw. COLOR\_RED

**speed**

Geschwindigkeit des Lauflichts in [ms]

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange

Unzulässiger Wert für from, to, color oder speed.

apiTypeFault

Unzulässiger Typ für from, to, color oder speed.

## 6.5 led\_runningLightOutstanding

led\_runningLightOutstanding <from>, <to>, <color>,  
<speed>, <outstandingLedNumber>

Läßt ein Lauflicht laufen, bei der eine bestimmte LED die jeweilig andere Farbe aufweist.

**Vorbedingung:**

- keine

**Parameter:**

**from, to**

Das Lauflicht läuft von LED 'from' bis LED 'to'.

Ist 'from' kleiner als 'to', dann läuft das Lauflicht „anders herum“.

Ist 'from' gleich 'to', leuchtet die eine LED ständig.

**color**

COLOR\_GREEN bzw. COLOR\_RED

**speed**

Geschwindigkeit des Lauflichts in [ms]



### **outstandingLedNumber**

Nummer der LED, die mit der anderen Farbe aufleuchtet.

### **Rückgabewert:**

- keiner (Prozedur)

### **Exceptions:**

apiValueRange

Unzulässiger Wert für from, to, color, speed oder outstandingLedNumber.

apiTypeFault

Unzulässiger Typ für from, to, color, speed oder outstandingLedNumber.

---

## **7 SecureApi**

roloFlash kann:

- Prüfsummen (CRC32) berechnen: **sec\_crc**
- Hashes (MD5 und SHA1) berechnen: **sec\_hash**
- Daten ver- und entschlüsseln (AES 128, 192 und 256):  
**sec\_encrypt** und **sec\_decrypt**
- Verschlüsselte Dateien flashen: **target\_writeFromFile**
- Verschlüsselte Basic-Skripte ausführen: **chain**

### **Optional Parameter „opstate“**

Für die ersten 3 genannten Funktionen steht ein optionaler Parameter „opstate“ zur Verfügung. Er dient dazu bei größeren oder nicht zusammenhängenden Daten zu markieren, wann Daten beginnen und enden:

- **SEC\_SINGLEBLOCK** (default): Die Daten bestehen nur aus einem Block. Dieses ist der Defaultwert, wird also angenommen, wenn kein opstate angegeben wird. Außerdem ist er die Summe von SEC\_FIRSTBLOCK und SEC\_LASTBLOCK.
- **SEC\_FIRSTBLOCK**: Muss beim ersten Block der Daten gesetzt sein, eine neue Berechnung beginnt.
- **SEC\_NEXTBLOCK**: Der Block ist weder der erste noch der letzte Block einer Berechnung.

- **SEC\_LASTBLOCK:** Muss beim letzten Block angegeben werden, damit die Berechnung abgeschlossen wird.

### Für Ver- und Entschlüsselung:

Cryptspec wird bei folgenden Funktionen verwendet:

- decrypt / encrypt
- target\_writeFromFile
- chain

### cryptSpec:

cryptSpec ist ein Vari-Array, das immer exakt zwei weitere Vari-Arrays enthält:

**algoSpec** und **dataSpec**.

### algoSpec:

algoSpec ist ein Vari-Array mit folgenden Parametern:

- **algo(-rithmus):** Es wird AES unterstützt. Muss immer die Konstante SEC\_AES sein.
- **width:** Die Schlüsselbreite, Werte sind 128, 192 und 256
- **mode:** Der Mode des AES
  - **SEC\_ECB:** electronic code book mode, jeder Block wird für sich verschlüsselt.  
[https://de.wikipedia.org/wiki/Electronic\\_Code\\_Book\\_Mode](https://de.wikipedia.org/wiki/Electronic_Code_Book_Mode)
  - **SEC\_CBC:** cipher block chaining mode, beim ersten Block wird vor der Verschlüsselung ein zusätzlicher Initialisierungsvektor mit dem Klartext verXodert. Bei allen weiteren Blöcken wird vor der Verschlüsselung das Ergebnis des vorherigen Blockes mit dem Klartext verXodert.  
[https://de.wikipedia.org/wiki/Cipher\\_Block\\_Chaining\\_Mode](https://de.wikipedia.org/wiki/Cipher_Block_Chaining_Mode)
  - **SEC\_CTR:** counter mode, statt dem Klartext wird ein Counter verschlüsselt, beim ersten Block 0, dann 1 etc. Das Ergebnis wird mit dem Klartext verXodert. Bei diesem Mode sind Sie nicht an Blockgrößen gebunden, da die Daten nicht durch den Algorithmus gehen. Stattdessen erzeugt der Algorithmus einen Bitstream, mit dem die Daten verXodert werden. Padding wird da-

her nicht benötigt und nicht empfohlen.

[https://de.wikipedia.org/wiki/Counter\\_Mode](https://de.wikipedia.org/wiki/Counter_Mode)

### **dataSpec:**

- dataSpec ist ein Vari-Array mit folgenden Parametern:
- **key:** Ein array of char, array of int oder array of long (außer vari-Array), welches den Schlüssel enthält.
- **iv:** (nur bei algoSpec mode SEC\_CBC und SEC\_CTR) Ein array of char, array of int oder array of long (außer vari-Array), welches den Initialization Vektor enthält.
- **padding:** Padding mit der Möglichkeit PKCS7 einzusetzen:  
[https://en.wikipedia.org/wiki/PKCS\\_7](https://en.wikipedia.org/wiki/PKCS_7)  
Je nach verwendeter Funktion und ob es sich um eine Ver- oder Entschlüsselung handelt, stehen unterschiedliche Möglichkeiten zur Verfügung.
  - **0-15:** Anzahl der Bytes, die nach einer Entschlüsselung ignoriert werden sollen.
  - **SEC\_NOPADDING:** Es soll kein Padding angewendet werden. Dieses ist bei SEC\_CTR empfohlen.
  - **SEC\_PKCS7:** Beim Entschlüsseln werden anschließend die Daten gemäß PKCS7-Padding gekürzt. Sollten die Daten nicht PKCS7 entsprechen, dann wird eine Exception "paddingError" erzeugt. Beim Verschlüsseln werden vorab Padding Bytes so angehängt, dass die notwendige Blockgröße erreicht wird.

## **7.1 sec\_crc**

```
crcValue = sec_crc(<array>, <crcSpec>, <opstate>)
```

oder

```
crcValue = sec_crc(<fileSystem>, <fileName>, <crcSpec>,  
<opstate>)
```

Berechnet den **CRC-32** für ein Array oder für eine Datei.

### **Vorbedingung:**

- keine

**Parameter:**

**array (für Daten)**

Ein array of char, array of int oder array of long (außer vari-Array), welches die Daten enthält.

**FileSystem (oder für Datei)**

Gibt an, auf welchem Dateisystem die Datei sich befindet. Mögliche Werte sind:

**SDCARD, FLASHVARS, FLASHDISK**

**fileName (oder für Datei)**

Es gelten die Bedingungen für Dateinamen, siehe Kapitel „Flash-Da-ta“.

**crcSpec**

Bitte geben Sie hier die Konstante **SEC\_CRC32** an.

**opstate**

(optional)

Wird benötigt, wenn Sie den CRC von mehreren Daten zusammen berechnen möchten.

**Rückgabewert:**

- CRC Wert (auch als Zwischenergebnis bei opstate SEC\_FIRSTBLOCK oder SEC\_NEXTBLOCK)

**Hinweis:**

- Mittels **opstate** kann der CRC-Wert über mehrere Arrays und Dateien (auch gemischt) berechnet werden.

### **Exceptions:**

secError	- allgemeiner Fehler bei der Berechnung - Es wurde als opstate SEC_NEXTBLOCK oder SEC_LASTBLOCK angegeben ohne das vorher SEC_FIRSTBLOCK verwendet wurde.
apiValueRange	Unzulässiger Wert für crcSpec oder opstate.
apiTypeFault	Unzulässiger Typ für einen Parameter.

## **7.2 sec\_hash**

hashArray = sec\_hash(<array>, <hashSpec>, <opstate>)

oder

hashValue = sec\_hash(<fileSystem>, <fileName>, <hashSpec>, <opstate>)

Berechnet den **MD5-Hash** oder **SHA1-Hash** für ein Array oder für eine Datei. Über roloBasic können Hash-Funktionen benutzt werden, um HMAC-Hashes (Hashes mit Schlüssel) zu berechnen.

### **Vorbedingung:**

- keine

### **Parameter:**

#### **array (für Daten)**

Ein array of char, array of int oder array of long (außer vari-Array), welches die Daten enthält.

#### **FileSystem (oder für Datei)**

Gibt an, auf welchem Dateisystem die Datei sich befindet. Mögliche Werte sind:

**SDCARD, FLASHVARS, FLASHDISK**

#### **fileName (oder für Datei)**

Es gelten die Bedingungen für Dateinamen, siehe Kapitel „Flash-Datta“.

#### **hashSpec**

Bitte geben Sie hier die Konstante **SEC\_MD5** oder **SEC\_HASH** an.

### **opstate**

(optional)

Wird benötigt, wenn Sie den Hash von mehreren Daten zusammen berechnen möchten.

### **Rückgabewert:**

- 0 bei opstate SEC\_FIRSTBLOCK oder SEC\_NEXTBLOCK
- Hash-Wert in allen anderen Fällen

### **Hinweis:**

- Mittels **opstate** kann der Hash-Wert über mehrere Arrays und Dateien (auch gemischt) berechnet werden. Damit werden HMAC-Berechnung ermöglicht.

### **Exceptions:**

secError

- allgemeiner Fehler bei der Berechnung
- Es wurde als opstate SEC\_NEXTBLOCK oder SEC\_LASTBLOCK angegeben ohne das vorher SEC\_FIRSTBLOCK verwendet wurde.

apiValueRange

apiTypeFault

- Unzulässiger Wert für hashSpec oder opstate.
- Unzulässiger Typ für einen Parameter.

## **7.3 sec\_encrypt**

sec\_encrypt <array>, <cryptSpec>, <opstate>

Verschlüsselt die Daten in dem Array unter Anwendung der cryptSpec und dem optionalen opstate.

### **Vorbedingung:**

- keine

### **Parameter:**

**array**

Ein array of char, array of int oder array of long (außer vari-Array), welches die Daten enthält. Die Länge der Daten muß ein Vielfaches der Blockgröße der Verschlüsselung (AES: 16 Bytes) sein.

Ausnahmen:

- In den dataSpec innerhalb der cryptSpec ist als Padding Algorithmus SEC\_PKCS7 angegeben und es handelt sich um die letzten Daten einer Verschlüsselung (kein opstate angegeben oder opstate = SEC\_SINGLEBLOCK oder opstate = SEC\_LASTBLOCK)
- In den algoSpec innerhalb der cryptSpec ist SEC\_CTR angegeben, in dataSpec innerhalb der cryptSpec ist als Padding Algorithmus SEC\_NOPADDING angegeben und es handelt sich um die letzten Daten einer Verschlüsselung (kein opstate angegeben oder opstate = SEC\_SINGLEBLOCK oder opstate = SEC\_LASTBLOCK)

### **cryptSpec**

Enthält die Informationen zur Verschlüsselung, unter anderem den Schlüssel. Die Definition finden Sie am Anfang des Kapitels SecureApi.

### **opstate**

(optional)

Wird benötigt, wenn Sie mehrere Daten zusammen verschlüsseln wollen.

### **Rückgabewert:**

- keiner (Prozedur)

### **Hinweis:**

- Bei Verwendung von PKCS7 wird empfohlen das Array vorab mit einer Reserve von 16 Bytes anzulegen, da das Array durch das Padding größer werden kann. Sie erleichtern damit die interne Speicherverwaltung. Beispiel für 1024 Bytes:

```
array = reserve(char, 1024 + 16)
resize array , 1024
```

- Die Verschlüsselung arbeitet mit Daten in Arrays. In der Skriptsammlung finden Sie eine Funktion zum Ver- und Entschlüsseln von Dateien.

### **Exceptions:**

cryptError

- allgemeiner Fehler bei der Berechnung
- Es wurde als opstate SEC\_NEXTBLOCK oder SEC\_LASTBLOCK angegeben ohne das vorher SEC\_FIRSTBLOCK verwendet wurde.

secParamError  
apiValueRange  
apiTypeFault

- Die angegebenen CryptSpec ist fehlerhaft.
- Unzulässiger Wert für hashSpec oder opstate.
- Unzulässiger Typ für einen Parameter.

## **7.4 sec\_decrypt**

sec\_decrypt <array>, <cryptSpec>, <opstate>

Entschlüsselt die Daten in dem Array unter Anwendung der cryptSpec und dem optionalen opstate.

### **Vorbedingung:**

- keine

### **Parameter:**

#### **array**

Ein array of char, array of int oder array of long (außer vari-Array), welches die Daten enthält. Außer bei SEC\_CTR muß die Länge der Daten ein Vielfaches der Blockgröße der Verschlüsselung (AES: 16 Bytes) sein.

Ausnahme:

- In den algoSpec innerhalb der cryptSpec ist SEC\_CTR angegeben, in dataSpec innerhalb der cryptSpec ist als Padding Algorithmus SEC\_NOPADDING angegeben und es handelt sich um die letzten Daten einer Entschlüsselung (kein opstate angegeben oder opstate = SEC\_SINGLEBLOCK oder opstate = SEC\_LASTBLOCK)

#### **cryptSpec**



Enthält die Informationen zur Entschlüsselung, unter anderem den Schlüssel. Die Definition finden Sie am Anfang des Kapitels SecureApi.

### **opstate**

(optional)

Wird benötigt, wenn Sie mehrere Daten zusammen entschlüsseln wollen.

### **Rückgabewert:**

- keiner (Prozedur)

### **Hinweis:**

- Die Entschlüsselung arbeitet mit Daten in Arrays. In der Skriptsammlung finden Sie eine Funktion zum Ver- und Entschlüsseln von Dateien.

### **Exceptions:**

cryptError

- allgemeiner Fehler bei der Berechnung  
- Es wurde als opstate SEC\_NEXTBLOCK oder SEC\_LASTBLOCK angegeben ohne das vorher SEC\_FIRSTBLOCK verwendet wurde.

secParamError  
secPaddingError

Die angegebenen CryptSpec ist fehlerhaft.  
PKCS7 ist angegeben, die entschlüsselten Daten entsprechen nicht der PKCS7 Kodierung.  
Unzulässiger Wert für hashSpec oder opstate.  
Unzulässiger Typ für einen Parameter.

apiValueRange  
apiTypeFault

## **8 Abfrage von roloFlash-Eigenschaften**

Mit folgenden Systemfunktionen / Systemkonstanten können Sie verschiedene Informationen Ihres roloFlashs ermitteln.

### **8.1 Versionsnummern etc.**

Name	Wert / Bedeutung
sys_companyName	„halec < <a href="https://halec.de">https://halec.de</a> >“
sys_deviceName	„roloFlash 2“ oder „roloFlash 2 AVR“
sys_softwareVersion	Versionsnummer der Firmware
sys_hardwareVersion	Versionsnummer der Hardware
sys_bootloaderVersion	Versionsnummer des Bootloaders
sys_imageVersion	roloFlash erwartet das vom Compiler erzeugte Image in dieser Version. Bitte verwenden Sie daher den zur Firmware des roloFlash passenden Compiler.

## 8.2 sys\_serialNumber

Name	Wert / Bedeutung
sys_serialNumber	Exception „functionNotSupported“

Bei vorherigen roloFlash-Versionen (Firmware kleiner als v07.AA) war diese Funktion fehlerhaft implementiert und lieferte nicht zuverlässig eindeutige Ergebnisse zur Erkennung von verschiedenen roloFlashes.

Die Funktionalität steht jetzt korrigiert in der neuen Funktion sys\_uniqueld zur Verfügung (siehe nächstes Kapitel)

## 8.3 sys\_uniqueld

Name	Wert / Bedeutung
sys_uniqueID	Ein String mit 24 Zeichen, jedes Zeichen '0' - '9' oder 'A' - 'F'

Die uniqueld ist für jeden roloFlash eindeutig. Sie können somit roloBasic-Skripte erstellen, die nur auf bestimmten roloFlash ablaufen.

### **Beispiel:**

1. Einmalig die uniqueld ermitteln:

```
print "uniqueId: ", sys_uniqueId, "\r\n"
```

Aus Log-File entnehmen:

uniqueId: 1B9FE86E90B7660F08E387B

2. Ihr Skript soll nur auf diesem roloFlash laufen, andernfalls mit einer Exception abbrechen:

```
if sys_uniqueId <> "1B9FE86E90B7660F08E387B"  
    print "Falscher roloFlash, Abbruch\r\n"  
    throw userException  
endif
```

**Hinweis:**

Für die uniqueId wird intern eine vom Chiphersteller vorgegebene eindeutige Device-ID genutzt.

---

## 9 Sonstige

### 9.1 sys\_setLogMode

sys\_setLogMode <logMode>

Es wird der Mode für das Loggen (siehe folgendes Kapitel „[print](#)“) festgelegt.

Der Ausdruck erfolgt an das Ende der Datei „LOG.TXT“. Wenn die Datei noch nicht existiert, dann wird sie erzeugt.

**Vorbedingung:**

- keine

**Parameter:**

**logMode:**

**LOGMODE\_OFF:** print-Ausgaben werden unterdrückt.

**LOGMODE\_NORMAL:** Die Datei wird geöffnet und bleibt geöffnet. Print-Ausgaben werden zwischengespeichert und gelegentlich in die Datei geschrieben. Am Ende des Skripts werden noch gespeicherte Daten geschrieben und die Datei geschlossen.

**LOGMODE\_IMMEDIATE:** Bei jeder Print-Ausgabe wird die Log-Datei geöffnet, die Ausgabe in die Datei geschrieben und die Datei wieder geschlossen. Somit ist sichergestellt, daß bei Ausführung der nächsten Skriptzeile die Print-Ausgabe auf der microSD-Karte gespeichert ist.

**Rückgabewert:**

- keiner (Prozedur)

**Hinweis:** Default ist der logMode LOGMODE\_NORMAL.

**Empfehlungen:**

Verwenden Sie LOGMODE\_IMMEDIATE nur bei der Fehlersuche. Da jede print-Ausgabe erneut die Datei öffnet, beschreibt und wieder schließt, wird auf der microSD-Karte jedesmal die FAT (file allocation table) beschrieben. Dies kann zu einem erhöhtem Verschleiß der microSD-Karte und deren Ausfall führen.

Wenn Sie Log-Ausgaben gar nicht benötigen, können Sie am Anfang des Skripts auf LOGMODE\_OFF wechseln. Sie können auch innerhalb des Skripts den LogMode wechseln.

Wenn Sie mit LOGMODE\_NORMAL arbeiten, werden die Log-Ausgaben eventuell erst nach Beendigung des Skripts auf die microSD-Karte geschrieben. Falls Sie in Ihren Skripten am Ende die letzte LED grün aufleuchten lassen, dann tun Sie das bitte erst am Ende, damit das abschließende Schreiben noch in der Reaktionszeit des Benutzers abgeschlossen werden kann. Vermutlich wird dieser roloFlash abziehen.

**Exceptions:**

apiValueRange  
apiTypeFault

Unzulässiger Wert für logMode.  
Unzulässiger Typ für logMode.

## 9.2 print

```
print <a>, <b>, ...
```

Es werden die Parameter *a*, *b* etc. ausgedruckt. Die Anzahl der Parameter ist beliebig.

Der Ausdruck erfolgt an das Ende der Datei „LOG.TXT“. Wenn die Datei noch nicht existiert, dann wird sie erzeugt.

### **Vorbedingung:**

- keine

### **Parameter:**

**a, b, ...**

Hier können Zahlen und Arrays ausgegeben werden. Beispiel:

```
value = 42
```

```
print "Der Wert ist: ", value
```

Ist ein angegebener Parameter weder eine Zahl noch ein Char-Array, dann wird nichts ausgegeben.

### **Rückgabewert:**

- keiner (Prozedur)

**Hinweis:** Die Ausgabe ist vom gewählten Logmode (siehe vorheriges Kapitel „Sonstige“) abhängig.

### **Exceptions:**

<diverse Exceptions des  
Dateisystems>

Siehe Kapitel „Exceptions des Dateisystems“.

## 9.3 sprintf

```
s = sprintf(<a>, <b>, ...)
```

Es werden die Parameter *a*, *b* etc. als Strings zurückgeliefert. Die Anzahl der Parameter ist beliebig. Die Ausgabe ist gleich wie bei `print`, jedoch wird hier nicht in die Datei „LOG.TXT“ geschrieben, sondern das Ergebnis zurückgegeben.

**Vorbedingung:**

- keine

**Parameter:**

***a*, *b*, ...**

Hier können Zahlen und Arrays ausgegeben werden. Beispiel:

```
value = 42
```

```
s = sprint("Der Wert ist: ", value)
```

Ist ein angegebener Parameter weder eine Zahl noch ein Char-Array, dann wird nichts ausgegeben.

**Rückgabewert:**

- Ausgabe

**Exceptions:**

<keine>

## 9.4 delay

`delay <duration>`

Es wird die angegebene Zeitdauer in ms abgewartet. Erst danach kehrt die Prozedur zurück.

**Vorbedingung:**

- keine

**Parameter:**

**duration**

Zeitangabe in ms, die gewartet werden soll.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange  
apiTypeFault

Unzulässiger Wert für duration.  
Unzulässiger Typ für duration.

## 9.5 sys\_getSystemTime

t = sys\_getSystemTime

Ermittelt die abgelaufene Zeit seit Systemstart in ms.

**Vorbedingung:**

- keine

**Parameter:**

- keine

**Rückgabewert:**

Systemzeit in [ms].

**Exceptions:**

- keine

## 9.6 getTargetBoardVoltage

u = getTargetBoardVoltage

Ermittelt die Spannung in mV, die das Targetboard liefert.

**Vorbedingung:**

- keine

**Parameter:**

- keine

**Rückgabewert:**

Ausgelesene Spannung in mV.

**Exceptions:**

- keine

## 9.7 sys\_setCpuClock

sys\_setCpuClock <frequency>

Es wird der interne CPU-Takt des roloFlash geändert.

- ein höherer Takt benötigt mehr Energie vom Targetboard
- ein niedriger Takt benötigt evtl. längere Zeit, um das roloBasic-Skript inkl. Flashen abzuarbeiten.

Der Takt bei Start des roloFlash ist für niedrigen Energieverbrauch auf 24 MHz eingestellt.

Achtung:



Bereits geöffnete Busse können dabei Ihren Takt ändern. Den aktuellen Takt können Sie abfragen.

Empfehlung:

Ändern Sie bei Bedarf den Takt am Anfang Ihres Skriptes.

**Vorbedingung:**

- keine

**Parameter:**

**frequency**

Angabe in Hz.

Unterstützte Werte:

- CPU\_CLOCKMAX: 120000000 (120 MHz)
- CPU\_CLOCKMIN: 24000000 (24 MHz)

Der Takt wird immer auf den nächstkleineren Takt, jedoch mindestens 24 Mhz eingestellt.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange  
apiTypeFault

Unzulässiger Wert für frequency.  
Unzulässiger Typ für frequency.

## 9.8 sys\_getCpuClock

frequency = sys\_getCpuClock

Ermittelt den aktuellen Takt des roloFlash in Hz.

**Vorbedingung:**

- keine

**Parameter:**

- keine

**Rückgabewert:**

Ausgelesener Takt in Hz.

**Exceptions:**

- keine

## 9.9 sys\_getEraseCounters

```
eraseCounters = sys_getEraseCounters
```

Ermittelt, wie oft bestimmt Sektoren des roloFlashs gelöscht wurden.

**Vorbedingung:**

- keine

**Parameter:**

- keine

**Rückgabewert:**

- Ein Array of Long mit 12 Werten, die den Sektoren des roloFlash entsprechen. Gründe für das Löschen von Sektoren sind:

- Bei Flash-Data & Doublebuffering: Wenn eine Speicheranforderung nicht mehr direkt bedient werden konnte und damit ein Wechsel der Buffer ausgelöst wurde.
- Bei Flash-Data durch fd\_format, fd\_cleanup und fd\_getSingleBufferMode
- Bei Firmware-Updates

Zugehörigkeit der Sektoren:

Sektor	roloFlash 2 roloFlash 2 AVR
0	Intern
1	
2	FlashVars
3	
4	Firmware
5	
6	
7	FlashDisk
8	
9	
10	
11	

Der Hersteller des im roloFlash verbauten Microcontrollers garantiert 10.000 Löschzyklen.

#### **Exceptions:**

- keine

## **9.10 setBitBlock**

setBitBlock <destArray, sourceArray, position, length>

Kopiert die mit length angegebene Anzahl an Bits vom Anfang des sourceArrays an die angegebene Bitposition im destArray.

- Dest- und sourceArray müssen vom gleichen Array-Typ sein (Array of char, Array of int oder Array of long) und werden als Bit-Array interpretiert.
- Aus dem SourceArray wird immer von Position 0 an gelesen.
- Sollte das Source- oder das DestArray nicht die mit length angegebene Anzahl an Bits beinhalten bzw. aufnehmen können, dann werden entsprechend weniger Bits kopiert.

**Vorbedingung:**

- keine

**Parameter:**

**destArray**

Array (Array of char, Array of int oder Array of long), in das die Daten an die angegebene Position kopiert werden sollen.

**sourceArray**

Array (Array of char, Array of int oder Array of long), aus dem die Daten von Position 0 an kopiert werden sollen.

**position**

Position im DestArray, an den die Daten kopiert werden sollen. Die Position muss sich innerhalb des DestArrays befinden.

**length**

Anzahl der Bits. Diese wird gegebenenfalls soweit verringert, dass sowohl das SourceArray die Daten aufnehmen kann als auch das DestArray genügend Daten liefern kann.

**Rückgabewert:**

- keiner (Prozedur)

**Exceptions:**

apiValueRange  
apiTypeFault

Unzulässiger Wert für position oder length  
Unzulässiger Typ

## 9.11 getBitBlock

```
destArray = getBitBlock(<sourceArray, position,  
length>)
```

Liefert ein Array, in dem die mit length angegebene Anzahl an Bits aus dem destArray von der angegebenen Position enthalten sind.

.

- Das gelieferte Array ist von selben Typ (Array of char, Array of int oder Array of long) wie das sourceArray.
- Die Größe des gelieferten Arrays ist exakt ausreichend um die mit length angegebene Anzahl an Bits aufnehmen zu können. Ungenutzte Bits sind auf 0 gesetzt.
- Sollte das Source Array nicht die mit length angegebene Anzahl an Bits beinhalten, dann werden entsprechend weniger Bits kopiert. Dieses hat keinen Einfluss auf die Größe des gelieferten Arrays.

**Vorbedingung:**

- keine

**Parameter:**

**sourceArray**

Array (Array of char, Array of int oder Array of long), aus dem die Daten von der angegebenen Position an kopiert werden sollen.

**position**

Position im SourceArray, von wo aus die Daten kopiert werden sollen. Die Position muss sich innerhalb des SourceArrays befinden.

**length**

Anzahl der Bits. Diese wird gegebenenfalls soweit verringert, dass das SourceArray genügend Daten liefern kann.

**Rückgabewert:**

destArray mit den kopierten Daten. Die Größe richtet sich nach der angegebenen length. Der Typ entspricht dem Typ vom sourceArray (Array of char, Array of int oder Array of long).

**Exceptions:**

OutOfMemory

Es steht nicht genug Speicher zur Verfügung, um das Basic Array anzulegen

apiValueRange

Unzulässiger Wert für position oder length

apiTypeFault

Unzulässiger Typ

## 9.12 chain

chain <fileSystem>, <fileName>, <cryptSpec>

Beendet das jetzige Basic-Skript und startet ein anderes compliliertes Basic-Skript mit der Dateiendung ".BIN".

**Vorbedingung:**

- keine

**Parameter:**

**fileSystem**

Gibt an, auf welchem Dateisystem die Prozedur ausgeführt werden soll. Mögliche Werte sind:

**SDCARD, FLASHVARS und FLASHDISK**

Aus Kompatibilitätsgründen wird auch eine 0 für die SD-Karte akzeptiert.

**fileName**

Die angegebene Datei muß eine compilierte Basic-Datei sein.

Für SD-Karte: Der Dateiname muss einem gültigen Dateinamen für

die SD-Karte entsprechen, siehe „Vorgaben für Dateinamen“.  
Für Flash-Data: Der Dateiname muss einer gültigen id für Flash-Data entsprechen, siehe „Vorgaben für I“.

### cryptSpec

optionaler Parameter zum Ausführen einer verschlüsselten Datei.  
Für den Parameter padding innerhalb der enthaltenden dataSpec gilt:

- **0-15:** Anzahl der Bytes, die nach der Entschlüsselung ignoriert werden sollen.
- **SEC\_NOPADDING:** Nach der Entschlüsselung werden die Daten nicht gekürzt. Dieser Mode ist nur bei SEC\_CTR anwendbar und dafür empfohlen.
- **SEC\_PKCS7:** Nach der Entschlüsselung werden die Daten gemäß PKCS7-Padding gekürzt. Sollten die Daten nicht PKCS7 entsprechen, dann wird eine Exception "secPaddingError" erzeugt.
- 

### Rückgabewert:

- keiner (Prozedur)

### Hinweis:

Um in dem neu geöffneten compilierten Skript alle Ressourcen wie Busse, UART etc. zur Verfügung zu haben, wird empfohlen vor der Anwendung von chain bereits belegte Ressourcen vorab frei zu geben.

### Exceptions:

apiTypeFault  
dataTypeError

<diverse Exceptions des  
Dateisystems>

cryptError  
secParamError  
secPaddingError

Unzulässiger Typ  
Versuchter Zugriff auf Daten, die nicht vom Typ  
"array of bytes" sind ( FlashVars bzw. FlashDisk)  
Siehe Kapitel „Exceptions des Dateisystems“.

Allgemeiner Fehler bei der Berechnung.  
Die angegeben CryptSpec ist fehlerhaft.  
PKCS7 ist angegeben, die entschlüsselten  
Daten entsprechen nicht der PKCS7 Kodierung.

## VII Exceptions

Im Handbuch für das roloBasic finden Sie die genaue Beschreibung, wie Exceptions ausgelöst und wieder gefangen werden können. Wird eine Exception nicht gefangen, wird die Exception mittels der LEDs angezeigt.

Falls dabei die Exception keine Zahl darstellt, wird die Exception „exceptionNotANumber“ ausgegeben. Details dazu finden Sie unter Kapitel „Exception aufgetreten“. Nur vom Benutzer ausgelöste Exceptions können nicht-numerisch sein.

Es gibt verschiedene Arten von Exceptions, die alle gleich behandelt werden:

- Exceptions des roloBasic
- Exceptions des Dateisystems
- Exceptions des roloFlash
- Vom Benutzer ausgelöste Exceptions

---

### 1 Exceptions des roloBasic

Diese Exceptions treten bei Fehlern auf, die nicht speziell etwas mit roloFlash zu tun haben, sondern mit der Abarbeitung des roloBasic. Ein typisches Beispiel dazu ist eine valueRange-Exception.

Sie finden diese Exceptions auch nochmal im Handbuch zu roloBasic.

Treten Fehler, wie bei den Exceptions valueRange, argumentFault und typeFault beschrieben, beim Aufruf an eine API-Funktion / -Prozedur auf, dann werden stattdessen die Exceptions apiValueRange, apiArgumentFault oder apiTypeFault erzeugt. Die jeweilige Nummer der Exceptions ist exakt 200 größer als die entsprechenden Exceptions des roloBasic.



Name	Nummer	Bedeutung
outOfMemory	1	Zu wenig freier Speicher vorhanden
rootstackOverflow	2	Interner Systemfehler
nullpointerAccess	3	Interner Systemfehler
valueRange	4	Wertbereichsüberschreitung, z.B. bei Zuweisung von Werten an Arrays
divisionByZero	5	Division durch 0. Kann bei div oder mod auftreten
argumentFault	6	Ungültige Anzahl Argumente beim Aufruf einer roloBasic-Funktion / Prozedur.
illegalFunction	7	Eine Variable wurde wie eine Funktion oder Prozedur aufgerufen, enthält jedoch keine gültige Funktion oder Prozedur.
indexRange	8	Indexbereichsüberschreitung bei Arrayzugriff
typeFault	9	Ein übergebener Parameter hat einen nicht passenden Typen.

## 2 Exceptions des Dateisystems

Diese Exceptions treten bei Fehlern im Zusammenhang mit dem Dateisystem oder mit der microSD-Karte auf.

Name	Nummer	Bedeutung
deviceError	101	Es konnte nicht von der microSD-Karte gelesen bzw. auf sie geschrieben werden.
badCluster	102	Probleme innerhalb des Dateisystems. Das Dateisystem sollte auf dem PC auf Konsistenz geprüft werden.
notMounted	103	Es wurde versucht, auf die microSD-Karte zuzugreifen, obwohl sie nicht angemeldet ist. Dieses deutet auf ein Problem mit der microSD-Karte hin.
removeError	104	Die microSD-Karte wurde entfernt.
createError	105	Das Erzeugen einer Datei oder eines Verzeichnisses ist fehlgeschlagen.
fileNotOpen	106	Die Datei ist nicht geöffnet.
fileNotFound	107	Die angegebene Datei oder das Verzeichnis konnte nicht gefunden werden.
diskFull	108	Die microSD-Karte ist voll.

truncateError	109	Das Kürzen einer Datei mittels <code>fs_truncate</code> ist fehlgeschlagen.
illegalCluster	110	Probleme innerhalb des Dateisystems. Das Dateisystem sollte auf dem PC auf Konsistenz geprüft werden.
fileLocked	111	Es wurde versucht, eine bereits geöffnete Datei ein zweites Mal zu öffnen. Evtl. wurde ein <code>fs_close</code> vergessen.
outOfFileHandles	112	Die Anzahl der maximal geöffneten Dateien ist auf 3 begrenzt. Es wurde versucht, eine weitere Datei zu öffnen.
loaderNotFound	113	Der benötigte Loader wurde auf der microSD-Karte nicht gefunden
fileIsOpen	114	Es wurde versucht, eine Datei zu löschen oder umzubenennen, die noch geöffnet ist (siehe <code>fs_remove</code> , <code>fd_remove</code> oder <code>fs_rename</code> )
renameError	115	Das Umbenennen einer Datei hat nicht funktioniert. Eventuell existiert bereits eine Datei mit dem neuen Namen (siehe <code>fs_rename</code> oder <code>fd_rename</code> )
dataTypeError	116	Versuchter Zugriff auf Daten, die nicht dem erwarteten Datentyp entsprechen (siehe <code>fd_readArrayElem</code> , <code>fd_readSubArray</code> , <code>fd_getArraySize</code> , <code>fs_fileSize</code> , <code>fs_open</code> ).

### 3 Vom Benutzer ausgelöste Exceptions

- Der Benutzer kann mittels `throw` selbst Exceptions auslösen. Diese können numerisch sein und bestehende Werte mitnutzen, z. B.:  
`throw rangeError`
- Um vom Benutzer erzeugte Exceptions von den anderen Exceptions besser unterscheiden zu können, können andere Exceptionnummern verwendet werden. Es wird hierzu die Konstante `userException` mit dem Wert 1000 zur Verfügung gestellt. Der Vorteil dieses Wertes ist, dass sie im Blinkcode besonders gut zu erkennen ist, wenn die Exception nicht mehr gefangen wird. Die Konstante ist als Offset für eigenen Exceptions nutzbar, z.B.:  
`throw userException + 1`
- Es können auch nicht-numerische Exceptions erzeugt werden. Falls eine solche Exception nicht mehr gefangen wird, wird sie zum Schluss in

die Exception `exceptionIsNotANumber` umgewandelt und der Code als Blinkcode ausgegeben, z. B.:  
`throw "error"`

## 4 Exceptions des roloFlash

Name	Nummer	Bedeutung
<code>exceptionIsNotANumber</code>	200	Es wurde eine Exception erzeugt, die keine Zahl ist, und innerhalb des roloBasic nicht mehr gefangen wurde. In diesem Fall wird die Original-Exception verworfen und durch diese Exception ersetzt.  Das kann nur durch vom Benutzer erzeugte Exceptions auftreten, da alle anderen Funktionen ausschließlich die hier beschriebenen numerischen Exceptions nutzen. Beispiel: <code>throw "Fehler"</code>
<code>imageTooLarge</code>	201	Das roloBasic-Skript ist zu groß. Es können maximal circa 65.000 Bytes geladen werden. Bitte überprüfen Sie die Dateigröße der vom Compiler erzeugten Datei.
<code>imageWrongVersion</code>	202	Der verwendete Compiler passt nicht zur Firmware auf dem roloFlash. Es wird empfohlen, immer jeweils den neuesten Compiler und neueste Firmware für den roloFlash zu verwenden.
<code>productWrongVersion</code>	203	Es wurde versucht, ein Image eines anderen Produktes auf den roloFlash zu laden. Beispielsweise wurde versucht, ein Image für roloFlash 1 auf einen roloFlash 2 zu laden.
<code>apiValueRange</code>	204	Wertbereichsüberschreitung eines Parameters beim Aufruf einer Api-Funktion / Prozedur . z.B. <code>ledOn 6, COLOR_GREEN</code> ! Es gibt nur 5 LEDs ( <b>Hinweis:</b> die Fehlernummer ist exakt 200 größer als die entsprechende Exception des roloBasic: <code>valueRange</code> )
<code>imageNotFound</code>	205	Die Datei <code>RUN_V07.BIN</code> konnte weder in der internen Flash-Disk noch auf der microSD-Karte (falls eingelegt) gefunden werden.
<code>apiBadArgumentCount</code>	206	Ungültige Anzahl Argumente beim Aufruf einer Api-Funktion / Prozedur. ( <b>Hinweis:</b> die Fehlernummer ist exakt 200 größer

		als die entsprechende Exception des roloBasic: badArgumentCount)
apiTypeFault	209	Ein an eine API-Funktion / Prozedur übergebener Parameter hat einen nicht passenden Typen. ( <b>Hinweis:</b> die Fehlernummer ist exakt 200 größer als die entsprechende Exception des roloBasic: typeFault)
targetWrongMode	210	Die aufgerufene Prozedur oder Funktion erfordert einen bestimmten Mode des Targets. Z. B. setzt die Prozedur setProgrammingSpeed den ProgramMode voraus.
targetCommunication	211	Ein Kommunikationsfehler mit dem Target.
targetMemoryLayout	212	Das Speicherlayout des Controllers ist nicht angegeben worden (target_setMemoryMap).
eraseError	213	Das Löschen des Targets hat nicht funktioniert.
targetVerify	214	Beim Zurücklesen von Daten wurde ein Unterschied festgestellt.
targetAlignment	215	Das Speicheralignment des Targets wurde nicht eingehalten. So darf bei STM32H7 nur an einer 32 Byte-Grenze begonnen werden ins Flash zu schreiben.
hexFileSize	230	Die Größe der angegebenen Hex-Datei ist nicht plausibel. Eventuell ist die Hex-Datei nicht in Ordnung oder leer.
hexFileCRC	231	Beim Parsen der Hex-Datei ist ein Prüfsummenfehler aufgetreten. Eventuell ist die Hex-Datei nicht in Ordnung.
hexFileSyntax	232	Beim Parsen der Hex-Datei ist ein Syntaxfehler aufgetreten. Eventuell ist die Hex-Datei nicht in Ordnung.
srecRecordTypeTooSmall	233	Zum Schreiben eines SREC-Files wurde ein Dateiformat ausgewählt, welches die tatsächlich verwendeten Adresse nicht kodieren kann. Bitte wechseln Sie zu einem größeren Format (Fileformat S28, S37) oder überlassen sie roloFlash die Auswahl des kleinstmöglichen Formates (Fileformat SREC)
badLoader	234	Der angegebene Loader kann nicht verwendet werden.
invalidHandle	250	Das benutzte Handle ist nicht gültig. Das Handle wurde bereits geschlossen oder ein falscher Parameter wurde anstatt des Handles verwendet.
resourceUnavailable	251	Die angeforderte Ressource ist nicht verfügbar.

		Dieses kann beim Öffnen eines Busses passieren, falls schon ein anderer Bus geöffnet hat, der zumindest teilweise auf die gleichen Ressourcen zugreift. Insbesondere tritt diese Exception auf, wenn der gleiche Bus zweimal geöffnet wird.
unknownTarget	252	In der Datenbank ist der angefragte Controller nicht aufgeführt (siehe <code>db_getHandle</code> )
propertyNotFound	253	Das angefragte Property ist für den angegebenen Controller nicht verfügbar (siehe <code>db_get</code> )
familyNotSupported	254	Die angegebene Controller-Familie wird nicht unterstützt (siehe <code>target_open</code> )
functionNotSupported	255	Es wurde eine für das aktuelle Target nicht unterstützte Funktion aufgerufen. Z.B. wird die Prozedur <code>target_writeBits</code> nur für Atmel-Controller unterstützt.
valueUnknown	256	Es wurde versucht, einen Wert auszulesen, der nicht ermittelt werden kann (siehe <code>target_getMemoryMap</code> ).
valueNotAllowed	257	Es wurde versucht, einen unzulässigen Wert zu verwenden (siehe <code>target_setMemoryMap</code> ).
timeoutError	258	Die aufgerufene Funktion dauert zu lange. Evtl. liegt ein Problem mit dem Target vor. Falls nach einem solchen Fehler mit dem Target weitergearbeitet werden soll, kann es nötig sein, das Target-Handle vorab zu schließen und neu anzufordern.
targetError	260	Das Target hat einen nicht weiter spezifizierten Fehler gemeldet. Bei ARM-Targets können das gesetzte Sticky-Bits sein.
writeProtectError	261	Der angesprochene Speicherbereich des Targets ist schreibgeschützt.
readProtectError	262	Der angesprochene Speicherbereich des Targets ist lesegeschützt.
writeError	263	Es gab einen Fehler beim Beschreiben des angesprochenen Speicherbereichs.
readError	264	Es gab einen Fehler beim Auslesen des angesprochenen Speicherbereichs.
targetMissingProperty	265	Ein benötigter Wert wurde nicht gesetzt.
targetResourceAccessConflict	266	Bei einem Zugriff gibt es einen Konflikt. Dieses kann bei STM32WB auftreten, wenn das PESD-Bit im Register <code>FLASH_SR</code> gesetzt ist.
fdCRCmissingError	290	Bei einem Zugriff auf ein Element aus Flash-Data

		ist ein CRC gefordert, aber das Element wurde ohne CRC angelegt (Element wurde mit <code>crcMode = NOCRC</code> angelegt, aber mit <code>crcMode = USECRC</code> abgefragt).
<code>fdCRCnotSupported</code>	291	Bei Flash-Data wird für den Inkrementtyp ein CRC gefordert. Der Inkrementtyp unterstützt keinen CRC (siehe <code>fd_write</code> ).
<code>fdCRCalreadySetError</code>	292	Bei Flash-Data kann auch bei einem Arraytypen ein CRC vorgesehen werden. Dieser ist anfangs nicht gesetzt und kann nur einmalig gesetzt werden. Anschließend kann in das Array nicht mehr geschrieben werden. Es wurde hier ein <code>fd_setCrc</code> aufgerufen und im Anschluss <code>fd_writeArrayElem</code> oder <code>fd_writeSubArray</code> .
<code>fdCRCmismatch</code>	293	Bei einem Lesezugriff in Flash-Data wurde der CRC berechnet und entsprach nicht dem gespeicherten Wert.
<code>fdWriteError</code>	294	Ein Schreibzugriff in Flash-Data hat nicht funktioniert. Evtl. sind nun Daten unter der aktuellen ID oder das gesamte Dateisystem korrupt.
<code>fdCorruptedFile</code>	295	Das Element in Flash-Data ist korrupt. Eventuell ist das gesamte Dateisystem korrupt.
<code>fdCorruptedError</code>	296	Bei Flash-Data ist das gesamte Dateisystem korrupt.
<code>secParamError</code>	340	<code>sec_encrypt/sec_decrypt/chain/target_writeFromFile</code> : Es ist ein Fehler in den übergebenen <code>CryptSpec</code>
<code>secError</code>	341	Es trat ein Verarbeitungsfehler bei Ver- oder Entschlüsselung oder bei der Berechnung eines Hashes oder CRC auf. Tritt auch auf, wenn Blockgrößen (AES: 16 Bytes) nicht eingehalten wurden.
<code>secPaddingError</code>	342	Bei Ver-/Entschlüsselung kann ein Padding (PKCS7) in den <code>CryptSpec</code> angegeben werden. Bei der Anwendung von PKCS7 ist ein Fehler aufgetreten.

# VIII Bedeutungen von LED-Codes

---

## 1 Normaler Betrieb

### 1.1 Keine microSD-Karte gefunden

**LEDs:**

- 1: rot
- 2:
- 3:
- 4:
- 5:

**Bedeutung:**

Keine microSD-Karte gefunden, bzw. die Karte ist nicht mit FAT32 formatiert.

**Hinweis:**

Für den normalen Betrieb ist es Voraussetzung, daß beim Anschließen des roloFlash die microSD-Karte bereits eingelegt ist. Der Fall, die microSD-Karte erst später einzustecken, ist für die Firmware-Aktualisierung des roloFlash reserviert. Falls Sie den roloFlash normal verwenden wollen und lediglich die microSD-Karte nicht eingesteckt hatten, dann entfernen Sie bitte roloFlash, legen die microSD-Karte ein und schließen Sie roloFlash erneut an.

### 1.2 Exception aufgetreten

Wenn eine Exception aufgetreten ist und diese nicht aufgelöst (gefangen) wurde, wird die Nummer der Exception durch einen Blinkcode angezeigt.

**LEDs:**

- 1: rot: geht am Anfang des Blinkcodes kurz aus und wieder an
- 2: rot: blinkend, Anzahl entspricht 1000-er der Exception
- 3: rot: blinkend, Anzahl entspricht 100-er der Exception
- 4: rot: blinkend, Anzahl entspricht 10-er der Exception
- 5: rot: blinkend, Anzahl entspricht 1-er der Exception

**Bedeutung:**

Dieser Code kann entstanden sein, indem

- im Skript eine entsprechende „throw“-Anweisung ausgeführt wurde. Beispiel:  

```
if getVoltage() > 4000  
    throw 1234 !Exception 1234 erzeugen  
endif
```
- eine Funktion / Prozedur Ihre Aufgabe nicht erfüllen konnte und eine Exception erzeugt hat.

---

## 2 roloFlash-Aktualisierung

Die Aktualisierung der roloFlash-Firmware ist im Kapitel „Aktualisieren von roloFlash“ beschrieben.

### 2.1 Warten auf microSD-Karte für Aktualisierung

**LEDs:**

- 1: rot
- 2:
- 3:
- 4:
- 5:

**Bedeutung:**

Wenn beim Start des roloFlash keine microSD-Karte eingelegt ist und keine Firmware auf dem roloFlash ist, dann wird auf das Einstecken der microSD-Karte gewartet, um anschließend die Aktualisie-



rung zu starten.

## 2.2 Aktualisierung läuft

### LEDs:

- 1: rot
- 2: grün \ im Wechsel blinkend
- 3: grün /
- 4:
- 5:

### Bedeutung:

Die Aktualisierung läuft. Diese benötigt circa 10-15 Sekunden, bitte nicht abbrechen.

## 2.3 Aktualisierung mit Erfolg abgeschlossen

### LEDs:

- 1: rot
- Hinweis: bei dem vorherigen Bootloader leuchtet die LED grün.
- 2: grün
- 3:
- 4:
- 5:

### Bedeutung:

Die Aktualisierung wurde mit Erfolg abgeschlossen. Nach dem Abziehen steht die neue Firmware bei der nächsten Nutzung zur Verfügung.

## 2.4 Aktualisierung fehlerhaft: Dateifehler

### LEDs:

- 1: rot
- 2: rot
- 3:
- 4:
- 5:

**Bedeutung:**

Die Aktualisierung schlug mit einem Dateifehler fehl. Die alte Firmware steht evtl. noch zur Verfügung.

**Mögliche Abhilfe:**

- Aktualisierung nochmals versuchen.
- Aktualisierung mit einer anderen Firmware durchführen.

## 2.5 Aktualisierung fehlerhaft: Datei nicht gefunden

**LEDs:**

1: rot  
2:  
3: rot  
4:  
5:

**Bedeutung:**

Die Aktualisierung konnte nicht gestartet werden, da keine Datei für die Aktualisierung gefunden wurde. Die alte Firmware steht noch zur Verfügung.

Es kann auch sein, dass die Datei beschädigt ist (Hash-Wert stimmt nicht).

**Mögliche Abhilfe:**

Datei für Firmware-Update auf die microSD-Karte kopieren, dann Aktualisierung nochmals versuchen.

## 2.6 Aktualisierung fehlerhaft: Mehrere Dateien gefunden

**LEDs:**

1: rot  
2:  
3:  
4: rot  
5:

**Bedeutung:**

Die Aktualisierung konnte nicht gestartet werden, da mehrere Dateien für die Aktualisierung gefunden wurde und dadurch nicht eindeutig ist, welche Datei verwendet werden soll. Die alte Firmware steht noch zur Verfügung.

**Mögliche Abhilfe:**

Es darf nur eine Datei vorhanden sein, die für eine Aktualisierung geeignet ist. Überflüssige Dateien bitte entfernen und dann Aktualisierung nochmals versuchen.

## 2.7 Aktualisierung fehlerhaft: Sonstiges

**LEDs:**

- 1: rot
- 2:
- 3:
- 4:
- 5: rot

**Bedeutung:**

Bei der Aktualisierung schlug etwas fehl. Die alte Firmware steht evtl. noch zur Verfügung.

**Mögliche Abhilfe:**

- Aktualisierung nochmals versuchen.
- Aktualisierung mit einer anderen Firmware durchführen.

# IX Spezifikationen

## 1 Unterstützte Controller von Atmel

Folgende Controller sind in der Datenbank bekannt. Die hier angegebenen Namen können mit `db_getHandle` verwendet werden.

### 1.1 AVR (ISP-Interface)

Anschluß über ISP-Interface.

Unterstützte Controller:

AT90CAN128,	AT90CAN32,	AT90CAN64,
AT90PWM1,	AT90PWM2,	AT90PWM216,
AT90PWM2B,	AT90PWM3,	AT90PWM316,
AT90PWM3B,	AT90PWM81,	AT90S1200,
AT90S2313,	AT90S2323,	AT90S2343,
AT90S4414,	AT90S4433,	AT90S4434,
AT90S8515,	AT90S8535,	AT90SCR100H,
AT90USB1286,	AT90USB1287,	AT90USB162,
AT90USB646,	AT90USB647,	AT90USB82,
ATmega103,	ATmega128,	ATmega1280,
ATmega1281,	ATmega1284,	ATmega1284P,
ATmega1284RFR2,	ATmega128A,	ATmega128RFA1,
ATmega128RFR2,	ATmega16,	ATmega161,
ATmega162,	ATmega163,	ATmega164A,
ATmega164P,	ATmega164PA,	ATmega165,
ATmega165A,	ATmega165P,	ATmega165PA,
ATmega168,	ATmega168A,	ATmega168P,

ATmega168PA,	ATmega168PB,	ATmega169,
ATmega169A,	ATmega169P,	ATmega169PA,
ATmega16A,	ATmega16HVA,	ATmega16HVA2,
ATmega16HVB,	ATmega16HVBrevB,	ATmega16M1,
ATmega16U2,	ATmega16U4,	ATmega2560,
ATmega2561,	ATmega2564RFR2,	ATmega256RFR2,
ATmega32,	ATmega323,	ATmega324A,
ATmega324P,	ATmega324PA,	ATmega324PB,
ATmega325,	ATmega3250,	ATmega3250A,
ATmega3250P,	ATmega3250PA,	ATmega325A,
ATmega325P,	ATmega325PA,	ATmega328,
ATmega328P,	ATmega328PB,	ATmega329,
ATmega3290,	ATmega3290A,	ATmega3290P,
ATmega3290PA,	ATmega329A,	ATmega329P,
ATmega329PA,	ATmega32A,	ATmega32C1,
ATmega32HVB,	ATmega32HVBrevB,	ATmega32M1,
ATmega32U2,	ATmega32U4,	ATmega32U6,
ATmega48,	ATmega48A,	ATmega48P,
ATmega48PA,	ATmega48PB,	ATmega64,
ATmega640,	ATmega644,	ATmega644A,
ATmega644P,	ATmega644PA,	ATmega644RFR2,
ATmega645,	ATmega6450,	ATmega6450A,
ATmega6450P,	ATmega645A,	ATmega645P,
ATmega649,	ATmega6490,	ATmega6490A,
ATmega6490P,	ATmega649A,	ATmega649P,
ATmega64A,	ATmega64C1,	ATmega64HVE,
ATmega64HVE2,	ATmega64M1,	ATmega64RFR2,
ATmega8,	ATmega8515,	ATmega8535,
ATmega88,	ATmega88A,	ATmega88P,
ATmega88PA,	ATmega88PB,	ATmega8A,

ATmega8HVA,	ATmega8U2,	ATtiny12,
ATtiny13,	ATtiny13A,	ATtiny15,
ATtiny1634,	ATtiny167,	ATtiny22,
ATtiny2313,	ATtiny2313A,	ATtiny24,
ATtiny24A,	ATtiny25,	ATtiny26,
ATtiny261,	ATtiny261A,	ATtiny4313,
ATtiny43U,	ATtiny44,	ATtiny441,
ATtiny44A,	ATtiny45,	ATtiny461,
ATtiny461A,	ATtiny48,	ATtiny80,
ATtiny828,	ATtiny84,	ATtiny840,
ATtiny841,	ATtiny84A,	ATtiny85,
ATtiny861,	ATtiny861A,	ATtiny87,
ATtiny88		

## 1.2 AVR (TPI-Interface)

Anschluß über TPI-Interface.

Unterstützte Controller:

ATtiny10,	ATtiny102,	ATtiny104,
ATtiny20,	ATtiny4,	ATtiny40,
ATtiny5,	ATtiny9	

## 1.3 AVR (PDI-Interface)

Anschluß über PDI-Interface.

Unterstützte Controller:

ATxmega128A1,	ATxmega128A1U,	ATxmega128A3,
ATxmega128A3U,	ATxmega128A4U,	ATxmega128B1,
ATxmega128B3,	ATxmega128C3,	ATxmega128D3,
ATxmega128D4,	ATxmega16A4,	ATxmega16A4U,
ATxmega16C4,	ATxmega16D4,	ATxmega16E5,
ATxmega192A3,	ATxmega192A3U,	ATxmega192C3,

ATxmega192D3,	ATxmega256A3,	ATxmega256A3B,
ATxmega256A3BU,	ATxmega256A3U,	ATxmega256C3,
ATxmega256D3,	ATxmega32A4,	ATxmega32A4U,
ATxmega32C3,	ATxmega32C4,	ATxmega32D3,
ATxmega32D4,	ATxmega32E5,	ATxmega384C3,
ATxmega384D3,	ATxmega64A1,	ATxmega64A1U,
ATxmega64A3,	ATxmega64A3U,	ATxmega64A4U,
ATxmega64B1,	ATxmega64B3,	ATxmega64C3,
ATxmega64D3,	ATxmega64D4,	ATxmega8E5

## 1.4 AVR (UPDI-Interface)

Anschluß über UPDI-Interface.

Unterstützte Controller:

ATmega3208,	ATmega3209,	ATmega4808,
ATmega4809,	ATtiny1604,	ATtiny1606,
ATtiny1607,	ATtiny1614,	ATtiny1616,
ATtiny1617,	ATtiny202,	ATtiny204,
ATtiny212,	ATtiny214,	ATtiny3214,
ATtiny3216,	ATtiny3217,	ATtiny402,
ATtiny404,	ATtiny406,	ATtiny412,
ATtiny414,	ATtiny416,	ATtiny417,
ATtiny804,	ATtiny806,	ATtiny807,
ATtiny814,	ATtiny816,	ATtiny817

## 1.5 AVR32 (aWire-Interface)

Anschluß über aWire-Interface.

Unterstützte Controller:

AT32UC3C0128C,	AT32UC3C0256C,	AT32UC3C0512C,
AT32UC3C064C,	AT32UC3C1128C,	AT32UC3C1256C,

---

AT32UC3C1512C,	AT32UC3C164C,	AT32UC3C2128C,
AT32UC3C2256C,	AT32UC3C2512C,	AT32UC3C264C,
ATUC128D3,	ATUC128D4,	ATUC64D3,
ATUC64D4,	AT32UC3L0128,	AT32UC3L016,
AT32UC3L0256,	AT32UC3L032,	AT32UC3L064,

---

## 2 Technische Daten

- Unterstützte Controller der Atmel-AVR-Serie mit ISP-Interface:
  - AT90
  - ATtiny
  - ATmega
- Unterstützte Controller der Atmel-AVR-Serie mit TPI-Interface:
  - alle Derivate
- Unterstützte Controller der Atmel-AVR-XMega-Serie mit PDI-Interface:
  - alle Derivate
- Unterstützte Controller der Atmel-AVR-Serie mit UPDI-Interface:
  - alle Derivate
- Unterstützte Controller der Atmel-AVR32-Serie mit aWire-Interface:
  - alle Derivate
- Programmierung des Mikrocontrollers über 6-polige ISP-/TPI-/PDI-/UPDI/aWire-Buchse. Diese kann direkt auf den 6-poligen ISP-, TPI-, PDI-, UPDI- oder aWire-Stecker aufgesteckt werden. Alternativ steht ein Adapter auf den 10-poligen ISP-Stecker, sowie ein 1:1-Adapter für den Einsatz von Flachbandkabeln zur Verfügung.
- Spannungsversorgung über den zu programmierenden Mikrocontroller (2,0 - 5,5 Volt).
- Schreiben und Lesen von:
  - Flash
  - EEPROM
  - Fusebits
  - Lockbits



- Unterstütztes Dateisystem auf der microSD-Karte: FAT32
- Interne Flash-Disk mit 640 kByte und interne Flash-Vars mit 16 kByte
- Unterstützte Dateiformate:
  - Intel HEX („HEX“) (I8HEX, I16HEX, I32HEX) (ASCII-Datei)
  - Motorola SREC (S19, S28, S37) (ASCII-Datei)
  - RAW (Binärdatei mit Rohdaten ohne Adreßangabe)
- Unterstützte Speicherkarten-Formate: microSD, microSDHC
- Gehäusegröße (ohne Universal-Connector): Höhe 46 mm, Breite 22 mm, Tiefe 10 mm