Flextech AKT Automation API Specification

Version 1.3
April 2026

This document is for information and instruction purposes. Flextech AKT reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Flextech AKT to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Flextech AKT products are set forth in written agreements between Flextech AKT and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Flextech AKT whatsoever.

FLEXTECH AKT MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

FLEXTECH AKT SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF FLEXTECH AKT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

U.S. GOVERNMENT LICENSE RIGHTS: The software and documentation were developed entirely at private expense and are commercial computer software and commercial computer software documentation within the meaning of the applicable acquisition regulations. Accordingly, pursuant to FAR 48 CFR 12.212 and DFARS 48 CFR 227.7202, use, duplication and disclosure by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in the license agreement provided with the software, except for provisions which are contrary to applicable mandatory federal laws.

TRADEMARKS: The trademarks, logos and service marks ("Marks") used herein are the property of Flextech AKT Corporation or other parties. No one is permitted to use these Marks without the prior written consent of Flextech AKT or the owner of the Mark, as applicable. The use herein of a third- party Mark is not an attempt to indicate Flextech AKT as a source of a product, but is intended to indicate a product from, or associated with, a particular third party.

Flextech AKT
24613 S 220th St.
Queen Creek, AZ 85142

Website: www.flextechakt.com

Table of Contents

AKT Automation Powered by Lua

The Flextech AKT Automation environment provides a powerful way to customize and expand the functionality of all AKT products. The AKT Automation environment is based on the Lua programming language with access to internal subsystems provided by built-in Lua modules. This environment allows for completely autonomous operation enabling a wide variety of demo, bench and in-vehicle test and measurement setups.

The hardware can be expanded through the onboard qwiic and I/O ports. Using the qwiic and I/O Lua modules, it is possible to add displays, actuators, sensors, real-time clocks, and other peripheral devices to create user interfaces and interact with the local environment.

Refer to your product's User Guide for details downloading and running AKT Automation scripts.

When developing AKT Automation scripts, the following techniques can speed up script development:

- Use XMODEM to transfer scripts to the product instead of copying to the SD card from the PC
- Running Lua with no arguments starts an interactive Lua interpreter. Lua code "chunks" can be copied and pasted from the PC into the interpreter for quick prototyping of logic or code blocks.
- Use the on-board 'edit' command for quick bug fixes or script modifications.

Most modules include a brief 'help()' method to assist script development.

Product Support

Automation Module	Function	Industrial A²B Bridge	Pocket A²B Bridge	Pocket Bus Monitor	Pocket TDM
rtos	RTOS	✔	✔	✔	✔
term	Terminal	✔	✔	✔	✔
system	System	✔	✔	✔	✔
qwiic	Qwiic		✔	✔	✔
pio	Pocket I/O		✔	✔	✔
led	Pocket LED		✔	✔	✔
drive	Filesystem	✔	✔	✔	✔
cbuf	Audio Circular Buffer	✔	✔	✔	✔
gen	Audio Signal Generator	✔	✔	✔	✔
route	Audio Routing	✔	✔	✔	✔
bm	Bus Monitor			✔	
tdm	TDM				✔
watchdog	Lua watchdog	✔	✔	✔	✔
irq*	A²B Bridge interrupt queue	✔	✔		
api*	A²B Bridge API locking	✔	✔		
comm*	A²B Bridge communication protocols	✔	✔		
master*	A²B Bridge master mode control	✔	✔		
setup*	A²B Bridge setup	✔	✔		
streaming*	A²B Bridge streaming	✔	✔		
util*	A²B Bridge utility	✔	✔		

*Refer to the A²B Bridge API Specification for details about the highlighted modules.

RTOS

The RTOS module provides access to basic RTOS services.

Require

rtos = require('rtos')

Methods

ret = rtos.delay(seconds)

ret = rtos.sleep(seconds)

Suspend script execution for the desired number of seconds

Parameters

Parameter	Type	Optional	Description
seconds	number	no	Number of seconds to suspend. Fractional seconds down to 0.001 are supported.

Return Values

Return	Type	Optional	Description
ret	number	N/A	Number of seconds suspended

---

sec = rtos.time()

sec = rtos.clock()

Return the time elapsed in seconds since power on.

***NOTE**: These methods are not the same as the Lua 'os' module methods of the same name. It is preferred to use the Lua os.clock() method as a monotonic time base for benchmarking and os.time() for the current time.*

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
sec	number	N/A	Number of seconds elapsed since power on. This value has a granularity of 0.001 seconds

Terminal

The Terminal module provides access to terminal services. These services can be used to build terminal based user interfaces.

Require

term = require('term')

Methods

Refer to the eLua term module reference manual for a full description of this module.

System

The System module provides access to certain system services and information. Methods exist to query basic system information, execute shell commands, set the system time, and place messages in the system log.

Require

system = require('system')

Methods

ok = system.shell(cmd)

Execute a shell command

Parameters

Parameter	Type	Optional	Description
cmd	string	no	Shell command to execute

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok, rfc3339 = system.date('rfc3339')

Set or get the current date and time in RFC3339 format. An example for April 1, 2025 at 20:00:00 would be '2025-04-01T20:00:00Z'. The terminating 'Z' is optional as the device has no concept of timezones or daylight savings time.

Parameters

Parameter	Type	Optional	Description
rfc3339	string	yes	Current date and time in RFC3339

Parameter	Type	Optional	Description
			format

Return Values

Return	Type	Optional	Description
ok	boolean string	N/A	True if successful setting the time. Current date and time in RFC3339 format when no arguments are supplied.
rfc3339	string	N/A	Current date and time in RFC3339 format when setting the time.

---

ok = system.syslog(msg)

Adds a message to the system log

Parameters

Parameter	Type	Optional	Description
msg	string	no	Message to add to the system log

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

info = system.sysinfo()

Returns basic system information

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
info	table	N/A	System information table **Parameter** / **Type** / **Value** name / string / System name id / number / System Product ID version / table / System version table. Includes 'str' version string and 'major', 'minor', 'release' version numbers.

---

ok = system.reset(type)

Reset a variety of subsystems

Parameters

Parameter	Type	Optional	Description
type	string	no	Reset the full system or a subsystem. Valid resets are: Type / Reset soft / Soft reset to power on reset values hard / Full power on system reset gen / Reset signal generator

Parameter	Type	Optional	Description
			<table><tr><td></td><td>subsystem (when available)</td></tr><tr><td>routes</td><td>Reset route subsystem</td></tr><tr><td>cbuf</td><td>Reset circular buffer subsystem</td></tr><tr><td>wav</td><td>Reset WAV file subsystem</td></tr><tr><td>tdm</td><td>Reset TDM settings (Pocket TDM only)</td></tr></table>

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

Qwiic

The Qwiic module provides I²C access to the qwiic port for expansion.

Require

qwiic = require('qwiic')

Methods

ok, ret = qwiic.i2c(i2cAddr,wBuf[,nRead])

Execute an I²C transaction on the qwiic port.

Parameters

Parameter	Type	Optional	Description
i2cAddr	number	no	7-bit I²C address of the qwiic device
wBuf	table	no	Data bytes to write. Can be an empty table with no data.
nRead	number	yes	Number of bytes to read. Omit for write only transactions.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false
ret	table	N/A	Bytes read from the device when nRead is greater than zero.

Pocket I/O

The Pocket I/O module provides access to hardware assets available on the Pocket I/O connector. These include I^2C, SPI, and GPIO accesses (where available).

Require

pio = require('pio')

Methods

ok, ret = pio.i2c(i2cAddr,wBuf[,nRead])

Execute an I^2C transaction on the Pocket device I/O connector I^2C port

Parameters

Parameter	Type	Optional	Description
i2cAddr	number	no	7-bit I^2C address of the qwiic device
wBuf	table	no	Data bytes to write. Can be an empty table with no data.
nRead	number	yes	Number of bytes to read. Omit for write only transactions.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false
ret	table	N/A	Bytes read from the device when nRead is greater than zero.

***NOTE**: This method is only available on the Pocket TDM and Pocket A^2B Bridge devices*

---

ok, ret = pio.spi(cmd,val)

Execute a SPI transaction on the Pocket device I/O connector SPI port

***NOTE**: This method is only available on the Pocket TDM device*

Parameters

<table>
<tr><th>Parameter</th><th>Type</th><th>Optional</th><th>Description</th></tr>
<tr><td>cmd</td><td>string</td><td>no</td><td>Execute a SPI operation
<table><tr><th>cmd</th><th>val</th></tr><tr><td>‘speed’</td><td>SPI speed in Hz. The value should be less than or equal to 50MHz. Due to divisor constraints, the speed may be less than the value specified.</td></tr><tr><td>‘mode’</td><td>SPI transfer mode. Valid values are 0-3.</td></tr><tr><td>‘xfer’</td><td>SPI data transfer. Supply an array of bytes to write for ‘val’. The SPI bytes read will be returned in ‘ret’</td></tr></table></td></tr>
<tr><td>val</td><td>number
table</td><td>no</td><td>See ‘cmd’</td></tr>
</table>

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false
ret	table	N/A	Bytes read from the device during a ‘xfer’

ok = pio.gpio([num[,cmd[,val]]])

Configures GPIO pins on the I/O connector. No arguments will return a table containing the current level of all GPIO pins.

***NOTE**: This method is only available on the Pocket Bus Monitor and Pocket A²B Bridge devices*

Parameters

Parameter	Type	Optional	Description
num	number	yes	GPIO number to operate on
cmd	string	yes	Operate on a GPIO **cmd** / **val** dir — Set the direction of the GPIO pin. Valid values are 'in', and 'out'. Default is 'out'. set — Sets the value of the GPIO output pin. Valid values are 'high', 1, true and 'low', 0, false. Default is 0. If no 'cmd' is given, the current level of the GPIO pin is returned.
val	string number boolean	yes	See 'cmd'. If no value is given, the default will be used.

Return Values

Return	Type	Optional	Description
ok	boolean table	N/A	True for success otherwise false Current values if no arguments

Pocket LED

The Pocket LED module provides access to the three multi-color LEDs available on the Pocket products.

Require

led = require('led')

Enumerations

LEDs

led.<LED>

LED Enumeration	Description
ALL_LED	All LEDs
STATUS	Status LED
USB	USB LED
IO	I/O LED

Colors

led.<COLOR>

Color Enumeration	Description
ALL_COLOR	All colors
RED	Red
GREEN	Green
BLUE	Blue

Methods

led.override()

Give control of the LEDs to the script. All LEDs will be initialized to off.

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This method requires no arguments

Return Values

Return	Type	Optional	Description
N/A	N/A	N/A	This method has no return value

---

led.release()

Returns LED control to the firmware.

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This method requires no arguments

Return Values

Return	Type	Optional	Description
N/A	N/A	N/A	This method has no return value

---

ok, msg = led.set(led,color,state)

***NOTE**: The 'Status' LED does not support BLUE*

Parameters

Parameter	Type	Optional	Description
led	number string	no	Valid LED strings are ‘usb’, ‘status’, ‘io’. Valid LED enumerations are ALL_LED, USB, IO, and STATUS. Enumerations can be or’d together to control more than one LED.
color	number string	no	Valid LED color strings are ‘red’, ‘green’, ‘blue’. Valid LED color enumerations are ALL_COLOR, RED, GREEN, BLUE. Enumerations can be or’d together to mix more than one color.
state	number string boolean	no	Set the state of the selected LEDs. Value values are ‘on’, true, 1 and ‘off’, false, 0.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false
msg	string	N/A	Result string

Filesystem

The Filesystem module provides access to certain file operations not included in the standard Lua io or os libraries. These include drive and directory operations.

Require

drive = require('drive')

Enumerations

DIRENT

The DIRENT enumerations can be used to decode the ‘fdate’ and ‘ftime’ values returned by the ls() / dir() method. The BITP and BITM values can be used to isolate the bit fields. OFFSET should be added to the returned value.

Alternatively, use the hms() and yyyymmdd() methods to decode these fields.

drive.<DIRENT>

LED Enumeration	**Description**
DIRENT_BITM_YEAR	Year bit mask for the ‘fdate’ bitfield
DIRENT_BITM_MONTH	Month bit mask for the ‘fdate’ bitfield
DIRENT_BITM_DAY	Day bit mask for the ‘fdate’ bitfield
DIRENT_BITM_HOUR	Hour bit mask for the ‘ftime’ bitfield
DIRENT_BITM_MIN	Minute bit mask for the ‘ftime’ bitfield
DIRENT_BITM_SEC	Second bit mask for the ‘ftime’ bitfield
DIRENT_BITP_YEAR	Year bit position for the ‘fdate’ bitfield
DIRENT_BITP_MONTH	Month bit position for the ‘fdate’ bitfield
DIRENT_BITP_DAY	Day bit position for the ‘fdate’ bitfield

DIRENT_BITP_HOUR	Hour bit position for the ‘ftime’ bitfield
DIRENT_BITP_MIN	Minute bit position for the ‘ftime’ bitfield
DIRENT_BITP_SEC	Second bit position the for ‘ftime’ bitfield
DIRENT_OFFSET_YEAR	Year offset for the ‘fdate’ bitfield
DIRENT_OFFSET_MONTH	Month offset for the ‘fdate’ bitfield
DIRENT_OFFSET_DAY	Day offset for the ‘fdate’ bitfield
DIRENT_OFFSET_HOUR	Hour offset for the ‘ftime’ bitfield
DIRENT_OFFSET_MIN	Minute offset for the ‘ftime’ bitfield
DIRENT_OFFSET_SEC	Second offset for the ‘ftime’ bitfield

Methods

ok, drives = drive.drive([drive,attribute])

Returns a list of support filesystem drives with attributes if no arguments are given. Otherwise sets a drive attribute. Common drives are ‘sd:’ for the SD card and ‘sf:’ for the internal flash filesystem.

Parameters

Parameter	Type	Optional	Description
drive	string	yes	Drive to modify
attribute	string	yes	Attribute to set on the drive. Accepted attributes are ‘default’ to set the drive as the default drive. Required if ‘drive’ is specified.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false
drives	table	N/A	List of supported drives and attributes

ok, full = drive.df([drive])

Disk full command. Returns the disk full status of all drives if no parameters are given. Otherwise returns the attributes for a specific drive. Common drives are ‘sd:’ for the SD card and ‘sf:’ for the internal flash filesystem.

Parameters

Parameter	Type	Optional	Description
drive	string	yes	Drive to query

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false
full	table	N/A	Drive full attributes **Attribute** / **Definition** available — Space available in 1k blocks blocks — Total space available in 1k blocks percent — Percentage full used — Space used in 1k blocks

yyyy, mm, dd = drive.yyyymmdd(fdate)

Returns the decoded year, month, and day from the ‘fdate’ bitfield returned by the ls() / dir() iterator.

Parameters

Parameter	Type	Optional	Description
fdate	number	no	‘fdate’ bitfield value

Return Values

Return	Type	Optional	Description
yyyy	number	N/A	Year
mm	number	N/A	Month
dd	number	N/A	Day

---

h, m, s = drive.hms(ftime)

Returns the decoded hour, minute, and second from the ‘ftime’ bitfield returned by the ls() / dir() iterator.

Parameters

Parameter	Type	Optional	Description
ftime	number	no	‘ftime’ bitfield value

Return Values

Return	Type	Optional	Description
h	number	N/A	Hour
m	number	N/A	Minute
s	number	N/A	Second

---

iterator = drive.ls([drive])

iterator = drive.dir([drive])

Returns a directory list iterator for the specified drive. If no drive is specified, the default drive is used.

Parameters

Parameter	Type	Optional	Description
drive	string	yes	Drive name. Normally ‘sd:’ for the SD card or ‘sf:’ for the internal filesystem.

Return Values

Return	Type	Optional	Description
iterator	function	N/A	Directory iterator. Returns a table with the following fields. **Attribute** / **Contents** fdate — File date bitfield ftime — File time bitfield fsize — File size in bytes flags — 0 = file, 1 = directory fname — File name

Example

The following example iterates over all files on the SD card showing a simple directory listing.

```
drive = require('drive')
sd_dir = drive.dir('sd:')
while true do
   local dir_entry = sd_dir()
   if dir_entry == nil then break end
   local yyyy, mm, dd = drive.yyyymmdd(dir_entry.fdate)
   local h, m, s = drive.hms(dir_entry.ftime)
   print(string.format(
     '%04d-%02d-%02d %02d:%02d:%02d %9s %s',
     yyyy, mm, dd, h, m, s,
     dir_entry.flags == 1 and '<DIR>' or string.format('%d', dir_entry.fsize),
     dir_entry.fname
   ))
end
```

Bus Monitor

The Bus Monitor module provides access to the Bus Monitor event subsystem on the Pocket Bus Monitor. This can be used as a basis for in-situ autonomous A2B event and data logging.

Require

bm = require('bm')

General Operation

1. Require the Bus Monitor module
2. Subscribe to desired Bus Monitor events
3. Poll for events

Refer to the Bus Monitor event logger on Github for a comprehensive example of this module.

Enumerations

Bus Monitor Events

Events containing "A2B2" are specific to A2B 2.0 networks.

bm.event.<EVENT>

I2C
I2C_A2B2

I2C register or peripheral event

Parameter	Type	Value
tid	number	I2C transaction ID
type	number	I2C event type. See I2C events.
nodeAddr	number	Sub node address
addr	number	Register or peripheral I2C address
rw	boolean	True for read, false for write
data	number	I2C data byte
condition	number	I2C peripheral

<table>
<tr><td></td><td>
<table>
<tr><td></td><td></td><td>condition. See I2C conditions</td></tr>
<tr><td>src</td><td>number</td><td>I2C source (A2B 2.0 only). See I2C sources.</td></tr>
</table>
</td></tr>
<tr><td>SPI</td><td>SPI event
<table>
<tr><th>Parameter</th><th>Type</th><th>Value</th></tr>
<tr><td>respNode</td><td>number</td><td>SPI node</td></tr>
<tr><td>error</td><td>boolean</td><td></td></tr>
<tr><td>type</td><td>string</td><td>SPI transaction type
"ATOMIC_READ"
"ATOMIC_WRITE"
"FULL_DUPLEX"
"BULK_WRITE"</td></tr>
<tr><td>ss</td><td>string</td><td>Slave select
"ADR1"
"SIO2"
"ADR2"</td></tr>
<tr><td>wLen</td><td>number</td><td>Write data length</td></tr>
<tr><td>rLen</td><td>number</td><td>Read data length</td></tr>
<tr><td>wr</td><td>number array</td><td>Write data</td></tr>
<tr><td>rd</td><td>number array</td><td>Read data</td></tr>
</table>
</td></tr>
<tr><td>BUS_LOCK_LOCKED</td><td>Bus Locked event</td></tr>
<tr><td>BUS_LOCK_UNLOCKED</td><td>Bus Unlocked event</td></tr>
<tr><td>BIAS_OK_DETECTED</td><td>Bias OK detected event</td></tr>
<tr><td>BIAS_OK_REMOVED</td><td>Bias OK removed event</td></tr>
<tr><td>BIAS_REV_DETECTED</td><td>Bias Reverse detected event</td></tr>
<tr><td>BIAS_REV_REMOVED</td><td>Bias Reverse removed event</td></tr>
<tr><td>DISCOVERY_MODE</td><td>Discovery mode event
<table>
<tr><th>Parameter</th><th>Type</th><th>Value</th></tr>
<tr><td>status</td><td>string</td><td>"START"
"STOP"</td></tr>
<tr><td>respCycles</td><td>number</td><td>Response cycles</td></tr>
</table>
</td></tr>
<tr><td>IRQ</td><td>IRQ event</td></tr>
</table>

	<table><tr><th>Parameter</th><th>Type</th><th>Value</th></tr><tr><td>nodeAddr</td><td>number</td><td>Sub node address. First submode is zero.</td></tr><tr><td>status</td><td>number</td><td>IRQ status</td></tr></table>
IRQ_QUERY_A2B2	IRQ query event (A2B 2.0) <table><tr><th>Parameter</th><th>Type</th><th>Value</th></tr><tr><td>nodeAddr</td><td>number</td><td>Sub node address. First submode is zero. Broadcast is 0x1F.</td></tr><tr><td>bprio</td><td>number</td><td>Bus Priority</td></tr></table>
IRQ_ACK_A2B2	IRQ event (A2B 2.0) <table><tr><th>Parameter</th><th>Type</th><th>Value</th></tr><tr><td>nodeAddr</td><td>number</td><td>Sub node address. First submode is zero.</td></tr><tr><td>ack</td><td>boolean</td><td>IRQ Acknowledge</td></tr><tr><td>active</td><td>boolean</td><td>IRQ Active</td></tr><tr><td>invalid</td><td>boolean</td><td>IRQ Invalid</td></tr><tr><td>type</td><td>number</td><td>IRQ Type</td></tr></table>
DOWNSTREAM_SCF_ERROR **UPSTREAM_SRF_ERROR**	Downstream SCF error event <table><tr><th>Parameter</th><th>Type</th><th>Value</th></tr><tr><td>type</td><td>string</td><td>"SCF" "SRF"</td></tr><tr><td>missed</td><td>number</td><td>Control frames missed in last block.</td></tr><tr><td>hdcnt</td><td>number</td><td>Headers missed in last block</td></tr><tr><td>crc</td><td>number</td><td>Bad CRCs in last block</td></tr></table>
SLAVE_ERROR	Slave error / ack event <table><tr><th>Parameter</th><th>Type</th><th>Value</th></tr></table>

	<table><tr><td>nodeAddr</td><td>number</td><td>Sub node address. First submode is zero.</td></tr><tr><td>error</td><td>number</td><td>Error number</td></tr><tr><td>errorStr</td><td>string</td><td>"SRFMISSED_ERROR" "BROADCAST_ACK" "DISCOVERY_ERROR" "DOWNSTREAM_CRC_ERROR" "UNSPECIFIED_ERROR"</td></tr></table>
SEQUENCE_ERROR	Sequence error event <table><tr><th>Parameter</th><th>Type</th><th>Value</th></tr><tr><td>seq</td><td>Number</td><td>Sequence errors detected in last block</td></tr></table>
DSH_ERROR_A2B2 **USH_ERROR_A2B2**	Sequence error event (A2B 2.0) <table><tr><th>Parameter</th><th>Type</th><th>Value</th></tr><tr><td>crc</td><td>Number</td><td>CRC</td></tr></table>
NEWSTRCT	NEWSTRCT detected event <table><tr><th>Parameter</th><th>Type</th><th>Value</th></tr><tr><td>dn</td><td>table</td><td>Downstream slot info</td></tr><tr><td>up</td><td>table</td><td>Upstream slot info</td></tr><tr><td>[dn/up].slots</td><td>number</td><td>Total active downstream slots</td></tr><tr><td>[dn/up].dtOffs</td><td>number</td><td>SPI data tunnel slot offset</td></tr><tr><td>[dn/up].dtSlots</td><td>number</td><td>SPI data tunnel slots</td></tr></table>

I2C Events

bm.i2c.types.<TYPE>

I2C_REG	Register transaction
I2C_PERIPHERAL	Peripheral transaction
I2C_PERIPHERAL_CONDITION	Peripheral condition

I2C Peripheral Conditions

bm.i2c.conditions.<CONDITION>

I2C_RPTSTART	repeat start
I2C_ACK	ack
I2C_NACK	nack
I2C_NORMAL	data
I2C_STOP	stop
I2C_ERROR	error
I2C_UNKNOWN	unknown

I2C Sources (A2B 2.0 Only)

bm.i2c.src.<SOURCE>

PE	Protocol Engine
I2C	I2C
SPI0	SPI0
SPI1	SPI1
BSD	Bus Self Discovery

Methods

ok = bm.subscribe(event)

Subscribe to a Bus Monitor event

Parameters

Parameter	Type	Optional	Description
event	number	no	Event enumeration (i.e. bm.event.I2C)

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok = bm.unsubscribe(event)

Unsubscribe from a Bus Monitor event

Parameters

Parameter	Type	Optional	Description
event	number	no	Event enumeration (i.e. bm.event.I2C)

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

status = bm.status()

Returns the current status of the Bus Monitor event queue

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
status	table	N/A	<table><tr><th>Parameter</th><th>Type</th><th>Value</th></tr></table>

Return	Type	Optional	Description
			<table><tr><td>size</td><td>number</td><td>Size in bytes of the event buffer</td></tr><tr><td>fill</td><td>number</td><td>Fill level in bytes of the event buffer</td></tr><tr><td>events</td><td>number</td><td>Number of events in the event buffer</td></tr><tr><td>max</td><td>number</td><td>Maximum fill size of the event buffer</td></tr><tr><td>dropped</td><td>number</td><td>Number of dropped events</td></tr></table>

---

ok = bm.reset(event)

Reset the Bus Monitor event queue

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

event = bm.getEvent()

Retrieve an event from the Bus Monitor event queue

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
event	Table	N/A	**Parameter** / **Type** / **Value** event / number / Event enumeration timeStamp / number / Event timestamp EVENT SPECIFIC / / See event enumerations

Notes

1. All events are automatically unsubscribed and cleared from the queue when the module is garbage collected.

Audio Circular Buffer

The Audio Circular Buffer module can be used to monitor and save a snapshot of audio to a file on demand.

Require

cbuf = require('cbuf')

General Operation

4. Require the circular buffer module
5. Reset
6. Configure
7. Enable
8. Save, clear, disable, enable

Methods

ok = cbuf.reset()

Resets the circular buffer configuration and all data

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok = cbuf.clear()

Clears all data but maintains current configuration

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok = cbuf.enable()

Enable / start the circular buffer

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok = cbuf.disable()

Disable / stop the circular buffer

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok = cbuf.save(fname)

Save the contents of the circular buffer to a file

Parameters

Parameter	Type	Optional	Description
fname	string	no	Filename

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok = cbuf.config(channels, wsize, size, rate)

Configures the circular buffer

Parameters

Parameter	Type	Optional	Description
channels	number	yes	Number of channels. Default 2. The maximum number of channels is 32.
wsize	number	yes	Audio sample word size in bytes.

Parameter	Type	Optional	Description
			Default 2 (16-bit)
size	number	yes	Size of circular buffer in bytes. Default 1,920,000 bytes (20 seconds @ 1 Channel @ 48 kHz @ 16-bit). The maximum size is 32 MB.
rate	number	yes	Audio sample rate in Hz. Default 48000 (48 KHz)

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok = cbuf.domain(domain)

Set the circular buffer clock domain.

Parameters

Parameter	Type	Optional	Description
domain	string	no	Audio clock domain

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

status = cbuf.status()

Reports the status of the circular buffer

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
status	table	N/A	**Parameter** / **Type** / **Value** configured / boolean / Configure status enabled / boolean / Enable status channels / number / Channels wordSize / number / Sample word size sampleRate / number / Sample rate size / number / Buffer size in bytes fill / number / Buffer fill level in bytes

Audio Signal Generator

The Audio Signal Generator module provides access to the onboard signal generators (where available)

Require

gen = require('gen')

Constants

gen.<CONSTANT>

Constant	Type	Description
MAX_SIG_GENS	number	Maximum number of signal generators

Methods

ok = gen.clear()

Clears all signal generators

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

info = gen.get(idx)

Retrieves the settings of the signal generator at the specified index

Parameters

Parameter	Type	Optional	Description
idx	number	no	Signal generator index. Valid values are zero to gen.MAX_SIG_GENS - 1

Return Values

Return	Type	Optional	Description
info	boolean table	N/A	False for fail or table containing signal generator details. The table will always contain ‘idx’ and ‘type’ fields along with the following **Type** / **Additional Fields** TONE: amplitude, frequency PINK: amplitude WHITE: amplitude HEX: value UNKNOWN: N/A

---

ok = gen.set(idx,type[,args])

Configures the settings of the signal generator at the specified index

Parameters

Parameter	Type	Optional	Description
idx	number	no	Signal generator index. Valid values are zero to gen.MAX_SIG_GENS - 1
type	string	no	Valid types are ‘tone’, ‘pink’, ‘white’, ‘hex’, and ‘off’
args		Yes	Default values shown below are used if

Parameter	Type	Optional	Description
			not specified. **Type** / **Additional Arguments** tone — amplitude (number, 1.0), frequency (number, 1000) pink — amplitude (number, 1.0) white — amplitude (number, 1.0) hex — value (number, 0x00000000) off — N/A

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

Audio Routing

The Audio Routing module provides access to the onboard audio routing engine.

Require

route = require('route')

Constants

route.<CONSTANT>

Constant	Type	Description
MAX_ROUTES	number	Maximum number of routes

Methods

ok = route.clear()

Clears all routes

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

info = route.get(idx)

Retrieves the settings of the audio route at the specified index

Parameters

Parameter	Type	Optional	Description
idx	number	no	Route index. Valid values are zero to route.MAX_ROUTES - 1

Return Values

Return	Type	Optional	Description
info	boolean table	N/A	False for fail or table containing route details. See the product's User Guide for specific route information.

Example Routing Table Entry

```
{
  attenuation = -0.0,
  channels = 2.0,
  dst = {
    name = "A2B",
    offset = 0.0
  },
  idx = 0.0,
  mix = "set",
  src = {
    name = "GEN",
    offset = 0.0
  }
}
```

---

ok = route.set(idx,src,srcOffset,dst,dstOffset,channels[,attenuation[,'mix'|'set']])

Configures the settings of the route at the specified index. Refer to the product User Guide for route source and destination details.

Parameters

Parameter	Type	Optional	Description
idx	number	no	Route index. Valid values are zero to route.MAX_ROUTES - 1
src	string	no	Route source
srcOffset	number	no	Route source offset
dst	string	no	Route destination
dstOffset	number	no	Route destination offset
channels	number	no	Route channels
attenuation	number	yes	Attenuation in dB. Rounded down to the nearest 6dB. Default 0dB.
‘mix’ | ‘set’	string	yes	Mix with other overlapping routes or set to this route. Default ‘set’.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

TDM

The TDM module provides access to the TDM subsystem on the Pocket TDM.

Require

tdm = require('tdm')

Methods

ok = tdm.reset()

Resets TDM settings to power on reset values.

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok = tdm.start()

Starts the TDM interface

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok = tdm.stop()

Stops the TDM interface

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok = tdm.clk(dir[,options])

Sets the TDM bit clock direction and options

Parameters

Parameter	Type	Optional	Description
dir	string	no	Sets the TDM clock pin direction. Valid values are ‘in’ and ‘out’.
options	table	yes	Optional array of clock options. Valid options are ‘none’, ‘rising’, ‘falling’. Using ‘none’ as the first option clears any previously defined options.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok = tdm.sync(dir[,options])

Sets the TDM sync clock direction and options

Parameters

Parameter	Type	Optional	Description
dir	string	no	Sets the TDM clock pin direction. Valid values are ‘in’ and ‘out’.
options	table	yes	Optional array of clock options. Valid options are ‘none’, ‘rising’, ‘falling’, ‘early’, ‘pulse’, and ‘50%’. Using ‘none’ as the first option clears any previously defined options.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok = tdm.data(pair[,dir[,pins]])

Sets the TDM data pin pair direction and pins

Parameters

Parameter	Type	Optional	Description
pair	string	no	Select the TDM data pin pair. Valid values are ‘tdm01’, ‘tdm23’, ‘tdm45’, ‘tdm67’, and ‘tdm89’.

Parameter	Type	Optional	Description
dir	string	yes	Sets the TDM data pin pair direction. Valid values are ‘in’ and ‘out’.
pins	string	yes	Sets the active TDM data pin pair pins. Valid values are ‘primary’, ‘secondary’, and ‘both’.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok = tdm.size(word[,slot])

Sets the TDM word and slot size

Parameters

Parameter	Type	Optional	Description
word	number	no	Sets the TDM data word size in bits. Valid values are 16 and 32.
slot	number	yes	Sets the number of TDM slots. Valid values are 2, 4, 8, 16, and 32.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

Watchdog

The Watchdog module provides access to a simple Lua watchdog to protect AKT automation scripts from infinite loops.

Require

watchdog= require('watchdog')

Methods

ok = watchdog.enable(sec)

Enables and starts the watchdog

Parameters

Parameter	Type	Optional	Description
sec	number	No	Seconds elapsed until the watchdog expires. Fractional seconds down to 0.001 are supported.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok = watchdog.disable()

Disables and stops the watchdog

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false

---

ok = watchdog.pet()

Pets the watchdog and resets the watch expiration counter

Parameters

Parameter	Type	Optional	Description
N/A	N/A	N/A	This command requires no parameters.

Return Values

Return	Type	Optional	Description
ok	boolean	N/A	True for success otherwise false