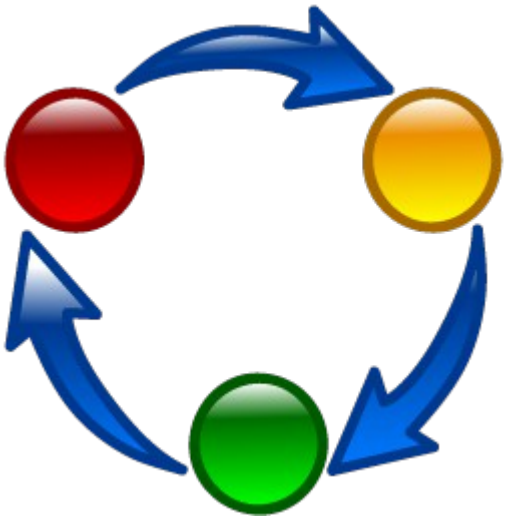
Cardocket Programming



How to
Track and Control anything

Revision 2.506

Introduction

The Cardocket programming language is based on the BASIC programming language, making it a superb tool for learning to program. Experienced programmers will quickly be able to write complex programs in a style similar to more powerful languages, such as C or Java.

If you're new at programming, it will be easiest to first look at the examples and then read the more technical stuff that follows.

Disclaimer

Overview of the Cardocket Programming language

The Cardocket programming language has a very loose syntax, making it easy to write code in the style you feel comfortable with. For example, the following code (in typical BASIC style):

```
IF NAME$ = "JOHN" THEN BEGIN
match = match + 1
.....
END
```

can also be written as follows (C style):

```
if (name$ == "John")
{
 match++;
 .....;
}
```

Variables

Variables are not declared and can consist of a combination of letter, numbers and underscores, e.g. variable1 and My_name.

Variables are not case-sensitive, thus apple and APple will point to the same variable.

All variables are floating point, but can be treated as integers.

To declare a variable as a string variable, put a dollar sign after it, e.g. string_variable$

String variables can be compared, e.g. If name$ = "PETER"

and added together, e.g. full_names$ = "John " + "Doe"

String and number variables can also be added, e.g.

```
age = 10
string$ =  "I am " + age + " years of age"
```

will result in: *"I am 10 years of age"*

If the number is a decimal, the string will output up to 3 decimal characters, e.g.

pi = 3.14159265359

string$ = "pi = " + pi

will result in: "pi = 3.141"

or

value = 1.400287

string$ = "Value = " + value

will result in "Value = 1.4"

Code execution

A typical program will consist of blocks of code that executes on specified triggers, e.g.

STARTUP: code to execute once when the module starts up

TIMER: code that executes every x milliseconds, where the interval can be specified and changed during execution of the program.

INCOMING CALL: Code to execute when receiving a call

INCOMING SMS: Code to execute when receiving an SMS

MOVEMENT: code to execute when detecting movement of the module.

Examples

Hello World Example

The first example replies to an incoming SMS with the message: "Hi there -Thanks for your message:" and the original message.

* Click on "Incoming SMS" on the debugger and insert the following code:

if (stringlength(SMS_MESSAGE$)>0) sendSMS(SMS_NUMBER$, "Hi There - Thanks for your message: "+SMS_MESSAGE$)

else sendSMS(SMS_NUMBER$, "Why so quiet?")

* Now click on “Debug” and make sure it doesn't show any errors.

* If a Cardocket unit is connected via USB, you can download the code to the unit by clicking on “Send code to unit”. Or you can test it in the debugger – click on the “Send” button and see what happens on the notification window.

That's it!

When receiving an SMS, the system string SMS_MESSAGE$ will contain the incoming message and SMS_MESSAGE$ will contain the message.

The command: sendSMS(number, message) sends an SMS back to the incoming number with the new message.

Note that all strings ends with a dollar sign ($) whereas number variables don't.

Detect Movement Example

This example will switch on the LED when it detects movement.

* First we need to set-up the unit once, so click on "Start-up" and insert the following code (remember, code after // is comments to make the program easier to understand and will be ignored by the program):

```
OVERRIDE_LED = 1   //we want full control of the LED
// execute code under "Movement" when it detects a change of 0.2G or more in any direction
ACC_THRESHOLD = 0.2
TIMER_INTERVAL = 10  //execute code under "Timer" every 10 milliseconds
```

* Click on "Movement" and add the following:

```
LED_timer=10 //our own variable. Every time it detects movement it will set LED_timer to 10
```

* Lastly, click on "Timer" and add the following to execute every 10 milliseconds

```
if (LED_timer>0) then begin
LED=1   //switch the LED on, (1=green, 2=red)
LED_timer = LED_timer – 1 //decrement the timer to make it switch off again
end
else LED=0  //switch off LED
```

* Click on Debug to run it in the simulator, or send the code to the unit for a live demo!

Programming Reference

Conditional statements		
Command	**Description**	**Usage**

IF
 	
THEN
 	
ELSE
 	
BEGIN
 	
END
	If [condition] then [statement]
	else [other statement]
	
	Alternative notation:
	
	{ (begin)
	
	} (end)
	
	Note: else is optional
		IF x<10 THEN x=x+1
		ELSE x=0
		
		IF x<10 THEN BEGIN
		x=x+1
		y=y+1
		END
		ELSE BEGIN
		x=0
		y=0
		END

AND OR	Logical operators. Used when testing for more than one condition, e.g. If apples>10 AND pears>10 Alternative notation: && (AND) || (OR)	IF apples>10 AND pears>10 THEN BEGIN ..... END IF apples>10 && pears>10 THEN ....

Comparison		
Command	**Description**	**Usage**
=	Comparison and assignment Alternative notation: ==	If a=1 then b=1 if a>1 then ....
>	Greater than	if a>=1 then ...
>=	Greater than or equal	if a<1 then...
<	Less than	if a<=1 then...
<=	Less than or equal	if a<>1 then .... (if a!=1 then ...)
<>	not equal Alternative notation: !=	if a!=1 then

Operations		
Command	**Description**	**Usage**
*	Multiply	a = 3*2
/	Divide	a = 3/2
+	Add	a = a+1
-	Subtract	a = a-1
++	increment variable (x=x+1)	if a<10 then a++
--	decrement variable (x=x-1)	if a>10 then a--

Bitwise Operations		
Command	**Description**	**Usage**
&	AND	a = a & 7
|	OR	a = a | 8
^	XOR	a = a^1
%	Modulus (remainder)	a = a%5

Functions		
Command	**Description**	**Usage**
wait(milliseconds)	Pause execution of code for specified number of milliseconds	LED=1 wait(5000) LED=0
return	Stops execution of code. Note: this command doesn't terminate the whole program, e.g. If running the timer task, it will stop execution of currently running code, but will still restart when next timer is due.	return
makeCall(number)	Makes a call to specified number. The call will be terminated as soon as it's answered or after ringing for more than 20 seconds. Can also be used for USSD queries, e.g. makeCall("*100#")	makeCall("08212345")
replyUSSD(string);	Reply to a USSD string, usually an option from a menu	replyUSSD(“4”);
sendSMS(number, message)	Sends an SMS to the specified number with message up to 160 characters long	SendSMS("08212345","This is the SMS text")
accelCalibrate()	Calibrates the accelerometer Make sure the unit is flat on a rigid surface before running this command. It's only necessary to run it once as calibration values are stored in memory.	accelCalibrate()
connectWeb(address)	Connect to a website, usually to send data. Note: Also set OVERRIDE_GPRS=1	connectWeb(“http://www.cardocket.com/update.php?data=12345”)
sendData(dataString)	Send a data string to the cardocket website. String consists of numbers, seperated by commas. Maximum length of string = 200	sendData(“123,0.01,55.4”)
random(max)	Generates a random number between 0 and max	random(10) will return a value between (and including) 0 and 10
randomize()	Seeds the random number generator. Run this once after startup to prevent the same random sequence after each startup.	randomize()
distanceFrom(latitude,longitude)	Calculates the current distance in meters from specified latitude and longitude	d=distanceFrom(-32.768,29.567)
deviceID()	Returns device ID as a string.	id$ = deviceID()

String Functions		
Command	**Description**	**Usage**
subString(string,startPos,length)	Returns part of a string, from position startPos, with specified length. Note: startPos starts at 1, not zero.	string$ = subString(“Hi there”,4,5) will set string$ to “there”
stringPos(haystack,needle)	Returns the position of string “needle” in string “haystack”. Returns 0 if not found. Note: index starts at 1, not zero	pos=stringPos(“eat me”,”me”) will return pos=5
stringLength(string)	Returns the length of a string	stringLength(“Hello there!”) will return 12

Flash functions Saving variables to flash will be stored permanently. Thus even after removing power they will keep their values.		
Command	**Description**	**Usage**
flashWriteValue(position,value)	Save a value to flash at specified position. Position = 0 .. 255	flashWriteValue(0,10000)
flashReadValue(position)	Read a value from flash at specified position. Position = 0 .. 255	saved = flashReadValue(0)
flashWriteString(position,string)	Save a string to flash at specified position. Position = 0 .. 7. Each string can be up to 128 characters	flashWriteString(2,”Hi There”)
flashReadString(position)	Read a string from flash at specified position. Position = 0 .. 7. Each string can be up to 128 characters	saved$ = flashReadString(2)

I^2C Functions Used to communicate with a device via the I^2C protocol. Connect IO1 to SCK (clock) and IO2 to SDA (data) of the device.		
Command	**Description**	**Usage**
I2C_START()	Send I^2C start condition	I2C_START()
I2C_STOP()	Send I^2C stop condition	I2C_STOP()
I2C_WRITE(data)	Send 8 bits (1 byte) to the device	I2C_WRITE(22)
I2C_READ(acknowledge)	Read 8 bits (1 byte) from the device, send acknowledge (1) or not (0)	data = I2C_READ(1)

1-Wire Functions Used to communicate with a device via the 1-Wire protocol. Data is send on IO3. Typical device: DS18B20 for temperature measurement		
Command	**Description**	**Usage**
OW_RESET()	Do a 1-Wire reset	OW_RESET()
OW_SEARCH_ROM(first_device)	Returns the ROM codes of devices on the 1-wire bus. Set first_device=1 to get first device, set first_device=0 to get rest of the devices	device1$ = OW_FIND_ID(1) device2$ = OW_FIND_ID(0) device3$ = OW_FIND_ID(0)
OW_WRITE(data)	Send 8 bits (1 byte) to the device	OW_WRITE(20)
OW_READ()	Read 8 bits (1 byte) from the device	data = OW_READ()
CRCDOW(crc,data)	Calculates an 8 bit CRC DOW value. CRC DOW is used by 1-wire products.	crc = 0 crc = CRCDOW(crc,byte1) crc = CRCDOW(crc,byte2)

Single Bus Functions Used to communicate with a device via the Single Bus protocol. Data is send on IO3. Typical device: AM2301 for temperature and humidity measurement		
Command	**Description**	**Usage**
SB_read(bytes_to_read)	Read bytes from a device Returns the number of bytes read from device.	if (SB_read(5) != 5) return
SB_next_value()	Returns next value read from device	SB_read(2) val1 = SB_next_value() val2 =SB_next_value()

Miscellaneous		
Command	**Description**	**Usage**
//	Comment	a=a+1 //this is a comment

;
 	
()
	Semi-colons at the end of each line is optional
	
	brackets when using if statements is optional. “Then“ can also be left out.
		a=a+1;
		a=a+1
		
		if a=1 then ...
		if (a=1) ...

System Variables

Name	Type	Description
PROGRAM_ERRORS	integer	Number of times the interpreter encountered an error in the code. It should be zero for a bug-free program.
SECONDS_SINCE_STARTUP	integer	Seconds elapsed since the unit started up, or since the new program was loaded.
SECONDS_SINCE_GPRS_UPDATE	integer	Seconds since the last successful GPRS connection
TIMER_INTERVAL	integer	Duration in milliseconds between code that execute on a timer. E.g. Set to 1000 to execute timer code every 1 second.
OVERRIDE_LED	integer	Set to 1 to gain full control over the status LED. Note: When switching an LED, (e.g.LED=1) OVERRIDE_LED will automatically be set to 1
OVERRIDE_GPS	integer	Control GPS status. 0: GPS on/off controlled by Cardocket Defaults 1: Force GPS to switch on 2: Force GPS to switch off
OVERRIDE_GSM	integer	Control GSM status. 0: GSM on/off controlled by Cardocket Defaults 1: Force GSM to switch on 2: Force GSM to switch off
OVERRIDE_TRIP	integer	Control trip start/stop 0: Start/stop controlled by Cardocket defaults 1: Force trip to start 2: Force trip to stop
OVERRIDE_LOW_POWER	integer	Set low power mode 0: Low power mode controlled by Cardocket defaults 1: Force unit to not to go to low power mode, unless battery is very low 2+ Force unit to go to low power mode for number of seconds. e.g. OVERRIDE_LOW_POWER=600 unit will go to low power mode for 600 seconds. Thereafter OVERRIDE_LOW_POWER will be set to 1. Note: if unit is in low power mode, it will be waken if movement detected that's greater than ACC_THRESHOLD
OVERRIDE_GPRS	integer	Control GPRS settings 0: GPRS connection handled by cardocket defaults 1: Do not connect to cardocket website. Use connectWeb() to send and receive data from custom website.

PULSE_THRESHOLD1 PULSE_THRESHOLD2	float	Voltage threshold to detect pulses. PULSES_IO1 will be incremented if input on IO1 goes from a voltage lower than PULSE_THRESHOLD1 to a voltage higher than PULSE_THRESHOLD1. The same goes for PULSE_THRESHOLD2 and IO2 Note: Input 3 is a digital input (0..3.3V and the threshold for pulses is fixed)
ACC_THRESHOLD	float	Threshold for accelerometer before triggering a movement event. Thus is ACC_THRESHOLD=0.5, the difference in X, Y and Z (ACC_DETECT) must be greater then 0.5G's to trigger movement.
LED	integer	Read or set the LED status. 0: LED off 1: LED Green 2: LED Red
IO1 IO2	float	Read or set voltage on IO1 and IO2: Read: Voltage on IO (0 to 32V) Write: 0: Force output to GND / 0V 1: Disconnect output (High Impedance state)
IO3	integer	Read or set voltage on IO3 Read: Voltage on digital IO3 (0=low/0V, 1=high/3.3V) Write: 0: Force to GND (0V) 1: Pull-up High (1k ohm to 3.3V) 2: Force High (3.3V)
PULSES_IO1 PULSES_IO2 PULSES_IO3	integer	Pulses counted on IO1 ... IO3. IO3 is a digital input and will count pulses between 0 and 3.3V. Threshold is around 2 Volts
VOLT_BATT	float	Voltage measured on internal battery. A fully charged battery should read 4.2V and a depleted battery below 3.3V
VOLT_SUPPLY	Float	Voltage measured on supply pin. Can be anything between 0V and 32V
VOLT_USB	float	Voltage measured on USB. Should be around 5V if USB connected.
VOLT_GSM	float	Voltage measured on GSM supply. Should be between 3.3V and 4.5V
GPS_SPEED	integer	Speed (in Km/h) measured by GPS
GPS_DIR	integer	Direction of travel as measured by GPS. Between 0 and 360 degrees.

GPS_HDOP	float	Reliability of GPS lock. Lower is better.
GPS_FIX	integer	Does GPS have a fix? 0: No fix 2: 2D fix 3: 3D fix
GPS_SAT	integer	Number of satellites in GPS view
GPS_LOCK	integer	Number of satellites GPS has locked on
GPS_LAT GPS_LNG	float	GPS latitude and Longitude coordinates
GSM_STATUS	integer	Status of GSM module: 0: switched off 1: not registered 2: registration denied 3: no SIM card detected 4: PIN on SIM 5: PUK on SIM 6: Registered 7: Making a call 8: Ringing 9: Talking 10: call terminated 11: incoming SMS 12: incoming call
GSM_SIGNAL	integer	Signal strength of GSM. Scale of 0 ...100 (100 = strongest signal)
GSM_ROAMING	integer	Indicates whether the modem is roaming on a network or not. 0 = Registered to Home network 1 = Registered to other network (National or International)
TRIP_STATUS	integer	Status of trip: 0 = Stopped 1 = Driving
TRIP_DISTANCE	integer	Distance in meters of current trip
TRIP_BUSINESS	integer	Sets the current trip type: 0 = Private Trip 1 = Business trip As soon as a trip is marked as stopped, the current value of TRIP_BUSINESS will determine the trip type.
AXX_X	float	Accelerometer value in X axis, 1=1G
AXX_Y	float	Accelerometer value in Y axis, 1=1G
AXX_Z	float	Accelerometer value in Z axis, 1=1G
ACC_DETECT	float	Intensity of movement detected. Sum of difference in acceleration in X, Y and Z direction

RTC_SECOND RTC_MINUTE RTC_HOUR RTC_DAY RTC_MONTH RTC_YEAR	integer	Values of internal clock. The unit gets the time from the GPS.
RTC_DAY_OF_WEEK	Integer	Day of week. 0: Sunday 1: Monday 2: Tuesday 3: Wednesday 4: Thursday 5: Friday 6: Saturday