



## **SDA\_OS Developer handbook**

## Basic Application structure

```
function init {
    screen = sys.gui.addScreen();
    sys.gui.addText(1, 1, 2, 6, "Hello, world!", screen);
    sys.os.gui.setMainScr(screen);
}

function update {
    # empty in this example
}
```

### ***Required functions***

Each application must implement all required functions.

#### **Init function**

```
function init {}
```

Init is called once when the app is loaded. arg0 - arg2: call arguments passed from sys.os.subProcess

#### **Update function**

```
function update
```

Update function is called each update cycle, when the app is active and in the foreground.

### ***Optional functions***

User might also implement optional functions. OS will call these functions under circumstances defined below.

#### **Exit function**

```
function exit {}
```

Exit function is called upon exiting the app.

#### **Suspend function**

```
function suspend {}
```

Suspend function is called when the app is suspended.

#### **Wakeup function**

```
function wakeup {}
```

Wakeup is called when the app is woken from suspend state. If the application process was set as a singular, and the app was woken from sys.os.subProcess call, init arguments are passed to the wakeup function.

## **SDA API Level history**

API level given by `sys.os.getVer` etc. works like this:

10510 - SDA version 1.5.1 with all its features

Note: before version 1.4 sub-version used only one digit:

1120 - SDA version 1.1.2 with all its features

## **Constants**

Constant	Description
SVP_LANG_CZ	Czech language
SVP_LANG_ENG	English language

## **Main OS functions**

### **Get redraw flag**

```
sys.os.getRedraw();
```

Gets redraw flag. *getRedraw* also works.

Return: [num] 1 if redraw flag is set, otherwise 0

### **Set redraw**

```
sys.os.setRedraw();
```

Sets redraw flag

Return: None

### **Wake the SDA from sleep**

```
sys.os.wake();
```

Wakes SDA without turning the screen on. SDA will wake in the low power mode and will sleep again after the lcd shutdown time.

Return: None

### **Pushes app to foreground**

```
sys.os.arise();
```

If called from timer or uart callback, the app is promoted to the foreground.

Return: None

### **Disable app close**

```
sys.os.noClose([num] enabled);
```

If enabled, running application is only suspended when close button is pressed. Application can still be closed from task manager.

Return: None

### **Get if running in simulator**

```
sys.os.inSim();
```

Gets if app is running in simulator. 1 - Simulator, 0 - Real hardware.

Return: [num] result

### **Show Error**

```
sys.os.error([str]errorText);
```

Throws error message

Return: None

### **Gets app path**

```
sys.os.getAppPath();
```

Gets directory path of the currently running svs app including the name of the app.

Return: [str] Path

## ***Keyboard***

### **Show keyboard**

```
sys.os.showKbd();
```

Shows system keyboard

Return: None

### **Hide keyboard**

```
sys.os.hideKbd();
```

Hides system keyboard.

Return: None

### **Get Keyboard state**

```
sys.os.kbdGetState();
```

Gets if keyboard is deployed 1 - keyboard shown, 0 - keyboard hidden

Return: [num] state

## **Misc**

### **Get random number**

```
sys.os.rnd();
```

Returns random number

Return: [num]RandomValue

### **Suspend app**

```
sys.os.suspend();
```

Suspends (minimizes) currently running app.

Return: none

### **Exit app**

```
sys.os.exit();
```

```
sys.os.exit([undef] arg0, [undef] arg1, [undef] arg2); # optional return values
```

Stops program execution after exiting *update* function and performing *exit* function.

Return: None

### **Check API level**

```
sys.os.checkVer([num] API_Level);
```

Checks for API Lvl support. If host level is below given API\_Level, error is thrown and app is terminated.

Return: None

### **Get API level**

```
sys.os.getVer();
```

Checks for API Lvl support.

Return: [num] SDA\_OS version number

### **Get system language**

```
sys.os.getLang();
```

Returns language of the running SDA\_OS build.

Return: defines SVP\_LANG\_CZ (0) or SVP\_LANG\_ENG(1)

## Subprocess

### Set process as singular

```
sys.os.setSingular();
```

Sets current process as singular.

Return: None

### Launch subprocess

```
sys.os.subProcess([str]fileName, [str/ref] callback, [undef] arg0, [undef] arg1, [undef] arg2);
```

Runs child process with given arguments. *fileName* must contain valid path to .svs file located in APPS directory. *callback* stores name of a function that will be called after child process exits. Subprocess will be launched after the current application returns from its *update* function. When strings are passed as arguments, their total size must not exceed APP\_ARG\_STR\_LEN define (2048 by default).

Return: None

### Enable launching subprocess from cwd

```
sys.os.subProcCWD([num] val);
```

Sets if subprocesses are launched from cwd or from APPS folder.

val:

Value:	Description
0	APPS folder (default)
1	CWD

Return: None

### Disable caching for launched subprocess

```
sys.os.subProcNC();
```

Disables caching for next call of sys.os.subProcess. Usefull when running modified content and precaching on launch is enabled in settings.

Return: None

### Return data to parent process

```
sys.os.subRetVal([undef] arg0, [undef] arg1, [undef] arg2);
```

Sets values that will be returned to parent process When strings are passed as arguments, their total size must not exceed APP\_ARG\_STR\_LEN define (2048 by default).

Return: None

### **Sets the clipboard string**

```
sys.os.setClipboard([str] string);
```

Sets the OS clipboard 256 chars by default

Return: [num] 1 - ok, 0 - string too long

### **Gets the clipboard string**

```
sys.os.getClipboard();
```

Gets the OS clipboard 256 chars max by default.

Return: [str] clipboard\_string

### ***OS settings functions***

#### **Reload homescreen settings**

```
sys.os.settings.homeRld();
```

Reloads homescreen settings stored in homescreen.cfg

Return: none

#### **Requests high privileges**

```
sys.os.settings.rqAuth();
```

Requests authorization form user to change system settings. Result can be retrieved with 'sys.os.settings.getAuth()';

Return: None

#### **Gets if privileges are granted**

```
sys.os.settings.getAuth();
```

Gets if high privileges are granted.

Return: [num] 1 if authorization is given

#### **Sets time and date**

```
sys.os.settings.setTime([num] year, [num] month, [num] day, [num] hour, [num] min);
```

Sets values that will be returned to parent process

Return: None

## **Gui**

### **Set main application screen**

```
sys.os.gui.setMainScr([num]id);
```

Sets main screen to screen with given id. When you wish to display overlay only, set this to 0.

Return: None

### **Get main application screen**

```
sys.os.gui.getMainScr();
```

Gets main screen id

Return: [num]id

### **Set root for redraw**

```
sys.os.gui.setRoot([num]in_apps, [str]dir);
```

Sets custom root directory for the redraw function. All paths for icons and other images passed to gui functions will use this folder as a root.

Return: None

### **Handle keypad input of a screen**

```
sys.os.gui.btnCtrl([num]screen_id, [num]back_btn_id);
```

Allows control of a given screen via buttons. Element given as back\_btn\_id will be linked with back button. When 0 is passed instead of id, back button will bring the user on the SDA\_OS main screen.

Return: None

### **Selects element for keyboard control**

```
sys.os.gui.btnSelect([num]element_id);
```

Selects element for keypad control. If the desired element is on a sub-screen of an button controlled screen, then the sub-screen must be also selected.

Return: None

### **Gets element selected by keyboard control**

```
sys.os.gui.btnGetSel([num]screen_id);
```

Gets selected element in current screen (or its sub-screens)

Return: [num] element\_id



### **Clear button control for a screen**

```
sys.os.gui.btnClear([num]screen_id);
```

Clears keypad input for entire screen.

Return: None

### ***Text field handling***

#### **Handle text input**

```
sys.os.gui.handleText([num]id, [str]text);
```

```
sys.os.gui.handleText([num]id);
```

Handles text input fields. Id is field id. Text is default text value.

Return: [str] New modified text value

Usage: `string_val = sys.os.gui.handleText([num]id, string_val)`; This is usefull if you are using `string_val` in every loop, `handleText` will keep the variable and the string in sync.

Alternative usage: `sys.os.gui.handleText([num]id)`; This will handle the text field and text value can be retrieved with `sys.gui.getString`. Note: the text value is still stored in SVS string memory.

#### **Set keyboard string**

```
sys.os.gui.setKbdStr([str] string);
```

Sets the current keyboard string (max 63 chars) Backspace code is `"\b"`, delete is `"\bd"`

Return: [num] 1 - ok, 0 - string too long

#### **Paste clipboard**

```
sys.os.gui.pasteClipboard();
```

Pastes clipboard into active text field

Return: none

#### **Get text cursor position**

```
sys.os.gui.getCPos([num] id);
```

Gets the cursor position of a text field

Return: [num]id

#### **Set text cursor position**

```
sys.os.gui.setCPos([num] id, [num]val);
```

Sets the cursor position of a text field

Return: [num]id

## Switch between landscape and portrait mode

```
sys.os.gui.setLandscape([num]val);
```

Sets the orientation of the display. 1 - Landscape 0 - Portrait Return: none

## Get display orientation

```
sys.os.gui.getLandscape();
```

Gets the orientation of the display. 1 - Landscape 0 - Portrait Return: [num]val

## Notification area icons

### Set notification area icon

```
sys.os.gui.setNotif([str] path_to_sic, [str] callback);
```

Sets the notification area icon. Path is dependent on actual CWD.

Return: [num] id (1 - 3), 0 - Error, probably no empty icon spot

### Free notification area icon

```
sys.os.gui.freeNotif([num] id);
```

Removes notification icon with given id.

Return: [num] 1 - ok, 0 - Error

## Default Icons

List of icons

Define	Description
ICON_NONE	Box with questionmark
ICON_BACK	Back arrow
ICON_FORWARD	Forward arrow

Hint: Unknown icons are drawn as ICON\_NONE

### Add an os icon to a button

```
sys.os.gui.setIcon([num] id, [num] icon_define);
```

Adds an icon to a given button. Icons are specified with system defines.

Return: none

## **Sound**

### **Beep the speaker**

```
sys.snd.beep();
```

Initiates system beep.

Return: None

### **Beep the speaker with callback**

```
sys.snd.beepC([num] frequency_hz, [num] duration_ms, [str] callback);
```

Makes sound of given frequency and duration, calls given callback afterwards. Internally calls `sys.snd.beepTime` and `sys.snd.beepFreq`, so calling `sys.snd.beep()`; will produce tone with frequency and duration of last `sys.snd.beepC` call.

If 0 is given as `frequency_hz`, only callback will be performed after given duration.

If empty string is passed as callback, no callback will be performed.

Return: None

### **Set beep param to default**

```
sys.snd.beepDef();
```

Sets beep to its default values.

Return: None

### **Set the duration**

```
sys.snd.beepTime([num]time (~ms));
```

Sets length of beep.

Return: None

### **Set the frequency**

```
sys.snd.beepFreq([num]frequency (Hz));
```

Sets frequency of the beep in Hz in range from 27 to 20000.

Return: None

### **Get if system sound is disabled**

```
sys.snd.getMute();
```

Returns system mute

Return: [num]1 if system is on mute.

## ***Date selector widget***

### **Init calendar widget**

```
sys.w.cal.init([num]year, [num]month, [num]day);
```

Creates callendar widget screen. With given year, month and day.

Return: [num]Callendar widget screen id.

### **Select date**

```
sys.w.cal.select([num]year, [num]month, [num]day);
```

Sets year, month and day to callendar widget.

Return: None

### **Update**

```
sys.w.cal.update();
```

Updates callendar widget.

Return: [num] 1 when callendar is clicked.

### **Mark day**

```
sys.w.cal.mark([num]day);
```

Marks day in callendar widget.

Return: None

### **Set highlighting**

```
sys.w.cal.highlight([num]val);
```

Enable that all buttons except marked are rendered as ghost buttons.

Return: None

### **Get selected day**

```
sys.w.cal.getDay();
```

Returns selected day.

Return: [num]day

## ***Counters***

### **Set counter**

`sys.cnt.set([num] ms);`

Sets system timer, it counts down and stops at zero.

Return: None

### **Gets counter**

`sys.cnt.get();`

Gets system timer value

Return: value of system timer

## ***Text obfuscation***

### **Unlock overlay init**

`sys.cr.unlockInit();`

Creates unlock overlay

Return: [num] overlay ID, 0 when error

### **Unlock overlay update**

`sys.cr.update([num] ovId);`

Updates unlock overlay

Return: None

### **Unlock overlay get ok**

`sys.cr.getOk([num] ovId);`

Gets if unlock was successfull

Return: [num] 1 - unlock success, 2 - unlock canceled

### **Unlock overlay clear ok**

`sys.cr.clrOk([num] ovId);`

Creates unlock overlay

Return: None

### **Get if is locked**

`sys.cr.getLock();`

Gets if crypto is unlocked

Return: [num] 1 - crypto unlocked, 0 - crypto locked

**Loads password as a key**

```
sys.cr.loadPass();
```

Loads OS password as a key

Return: 0 if success, 1 if error

**Load custom key string**

```
sys.cr.loadStr([str]key);
```

Loads custom string as a crypto key

Return: 0 if success, 1 if error

**Load custom keyfile**

```
sys.cr.loadKey([str]keyfile);
```

Loads custom keyfile as a crypto key

Return: 0 if success, 1 if error

**Load OS keyfile**

```
sys.cr.loadOSKey();
```

Loads OS keyfile as a crypto key

Return: 0 if success, 1 if error

**Generate keyfile**

```
sys.cr.genKey([str]keyfile);
```

Generates custom keyfile.

Return: 0 if success, 1 if error

**Lock**

```
sys.cr.lock();
```

Locks sda encryption

Return: None

**Encrypt file**

```
sys.cr.encrypt([str]fname);
```

Encrypts file.

Return: 0 if success, 1 if error

**Decrypt file**

```
sys.cr.decrypt([str]fname);
```

Decrypts file.

Return: 0 if success, 1 if error

## Encrypt string

```
sys.cr.encryptStr([str]source);
```

Encrypts given string.

Return: [str] encryptedString

## Decrypt string

```
sys.cr.decryptStr([str]source);
```

Decrypts given string.

Return: [str] decryptedString

## SDA OS HW functions

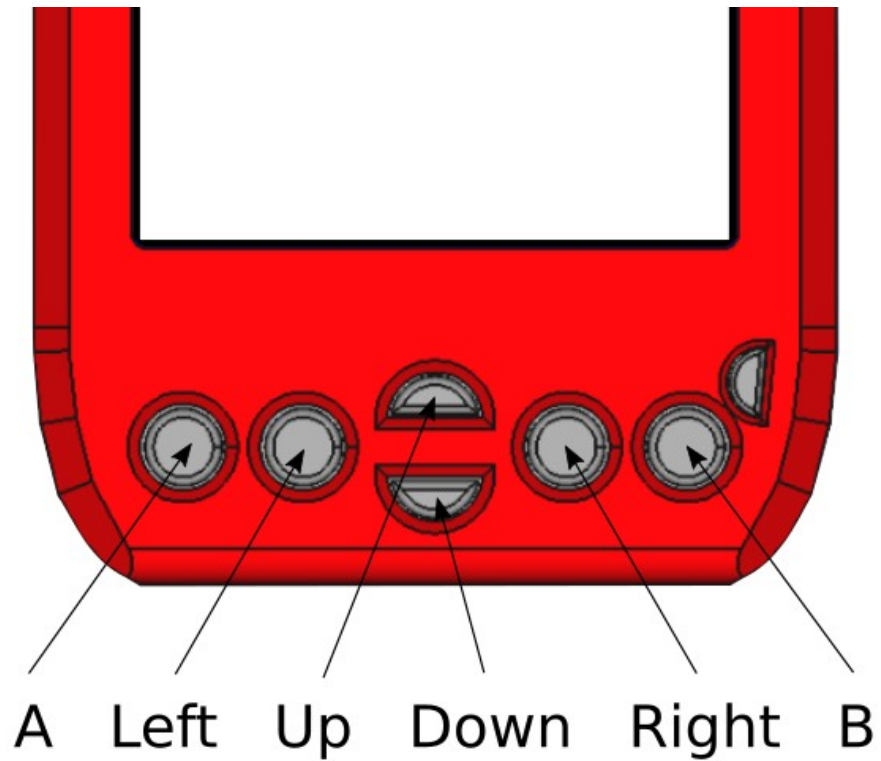
### Constants

#### Indicator LED

Constant	Description
LED_ON	Nonification led on
LED_OFF	Nonification led off
LED_BLINK	Nonification led pattern
LED_SHORTBLINK	Nonification led pattern
LED_ALARM	Nonification led pattern

#### Buttons

Constant	Description
BTN_A	Button define
BTN_LEFT	Button define
BTN_UP	Button define
BTN_DOWN	Button define
BTN_RIGHT	Button define
BTN_B	Button define



#### Expansion pin states

Constant	Description
PIN_IN	Pin set as input
PIN_OUT	Pin set as output
PIN_ALT	Pin set to its alternate function
PIN_NOPULL	Pin set as input with pulldown
PIN_PULLUP	Pin set as input with pullup
PIN_PULLDOWN	Pin set as input with no pull resistor



## **Power Functions**

### **Lock system suspend**

```
sys.hw.lockSuspend([num]val);
```

Sets sleep lock value. On 1 system won't go to any deeper sleep mode than SDA\_PWR\_MODE\_SLEEP\_LOW, regardless of the screen state.

Return: None

### **Lock LCD sleep**

```
sys.hw.lockSleep([num]val);
```

Sets sleep lock value. On 1 system won't shut down LCD automatically. User can still shut down the screen with power button, in that case, SDA will go in the SDA\_PWR\_MODE\_SLEEP\_LOW mode.

Return: None

### **Turn on the LCD**

```
sys.hw.wakeLcd();
```

Turns on the LCD screen.

Return: None

### **Get LCD state**

```
sys.hw.getLcdState();
```

Gets state of lcd.

Return: 1 if lcd is on, otherwise 0

### **Set LCD state**

```
sys.hw.setLcdState([num]lcd_state);
```

Sets the LCD state, 0 - off, 1 - on. In off state, the device will enter sleep mode.

Return: None

### **Set notification led pattern**

```
sys.hw.setLed([num]led_type);
```

Sets notification led to a given pattern, uses: LED\_ON, LED\_OFF, LED\_BLINK, LED\_SHORTBLINK, LED\_ALARM

Return: None

## ***Expansion Ports***

### **Get USB State**

`sys.hw.getUsbState();`

Gets state of usb port. Useful for determining if the SDA is connected to PC.

Return: 1 when powered from usb, otherwise 0

### ***Resource claiming***

System resources:

<b>Resource define</b>	<b>Description</b>
EXT_EXP_PORT	External expansion connector
INT_EXP_PORT	Internal expansion connector
USB_PORT	USB port (ftdi serial)
SERIAL_PORT	internal/external serial port

Locks area automatically freed on app close.

### **Claim hardware resource**

`sys.hw.claim([num]Resource);`

Claims given hardware resource for currently running app.

Return: [num] 1 - error, 0 - ok

### **Free hardware resource**

`sys.hw.free([num]Resource);`

Frees given hardware resource. Return: [num] 1 - error, 0 - ok

### **Get hardware resource state**

`sys.hw.getLock([num]Resource);`

Frees given hardware resource. Return: [num] 1 - locked, 0 - free

### **Internal expansion port**

Internal expansion connector pinout:

MCU pin	Exc pin number	Exc pin number	MCU pin
PD15	01	09	PB15
PD14	02	10	PB14
PD13	03	11	PB13
PD12	04	GND	-
PD11	05	Vcc (3.3V)	-
PD10	06	14	PE12
PD09	07	15	PE13
PD08	08	16	PE14

Dot marks the pin no. 1

### **Define direction of pins on the internal expansion**

```
sys.hw.iPinDef([num]Pin, [num]type, [num]pullUp);
```

Sets direction of internal expansion pins. Uses defines: PIN\_IN, PIN\_OUT, PIN\_ALT, PIN\_NOPULL, PIN\_PULLUP, PIN\_PULDOWN Pin number is number of pin on the connector, can be read from schematics.

Return: None

### **Set state of pins on the internal expansion**

```
sys.hw.iPinSet([num]Pin, [num]val);
```

Sets state of internal expansion pin. Value 1 sets the pin high, value 0 sets it low. Pin number is number of pin on the connector, can be read from schematics.

Return: None

### **Get state of pins on the internal expansion**

```
sys.hw.iPinGet([num]Pin);
```

Gets state of internal expansion pin. Pin number is number of pin on the connector, can be read from schematics.

Return: 1 if the pin is high, 0 if it is low.

### External expansion pins

External expansion connector pinout:

Exc pin number	MCU pin	Alt Function
1	GND	
2	PA1	ADC_IN
3	3.3V	
4	PE15	I/O
5	PB10	Tx
6	PB11	Rx

Pin no. 1 is the one closest to the charging LED

### Define direction of pins on the expansion

```
sys.hw.ePinDef([num]Pin, [num]type, [num]pullUp);
```

Sets direction of external expansion pins. Uses defines: PIN\_IN, PIN\_OUT, PIN\_ALT, PIN\_NOPULL, PIN\_PULLUP, PIN\_PULDOWN Pin number is number of pin on the connector, can be read from schematics.

Return: None

### Set state of pins on the expansion

```
sys.hw.ePinSet([num]Pin, [num]val);
```

Sets state of external expansion pin. Value 1 sets the pin high, value 0 sets it low. Pin number is number of pin on the connector, can be read from schematics.

Return: None

### Get state of pins on the expansion

```
sys.hw.ePinGet([num]Pin);
```

Gets state of external expansion pin. Pin number is number of pin on the connector, can be read from schematics.

Return: 1 if the pin is high, 0 if it is low.

### Get ADC readout

```
sys.hw.eADCRead();
```

Gets the voltage from pin 2 of the external expansion port. This function re-initializes the pin 2 to set it in the ADC mode. If you use this pin for anything else after, you need to re-init it with *sys.hw.ePinDef*.

Return: [float] measured voltage in volts.

## **Buttons**

Functions for handling hw buttons. Button defines: BTN\_A, BTN\_LEFT, BTN\_UP, BTN\_DOWN, BTN\_RIGHT, BTN\_B Events are the same as in handling GUI: EV\_NONE, EV\_PRESSED, EV\_HOLD, EV\_RELEASED

### **Get button event**

```
sys.hw.btn.getEvent([num]btn);
```

Return last button event.

Return: [num] event

### **Clears button events**

```
sys.hw.btn.clrEvent([num]btn);
```

Sets button event to EV\_NONE

Return: None

### **Enable button events with LCD off**

```
sys.hw.btn.stdbbyEn([num]val);
```

Enables button readout with LCD off. Val: 1 - enabled, 0 - disabled When this is enabled, SDA won't go in deep sleep, and button presses will be handled immediately.

Return: None

## **Communication**

### ***Serial transmit queue***

Could be used for all the transmission modes.

### **Serial expansion transmit queue**

```
sys.com.qAdd([num]data);
```

Queues given hex value to transmit buffer. Max 32 bytes.

Return: [num] 1-ok, 0-full buffer

### **Serial expansion transmit queue clear**

```
sys.com.qClr();
```

Clears transmit buffer.

Return: None

## ***USB serial interface***

### **USB serial transmit**

```
sys.com.usbTrs([str]data);
```

Sends given string to usb serial port.

Return: None

### **USB serial transmit queue**

```
sys.com.usbTrsQ();
```

Sends previously stored queue to the initialized serial port. Queue can be filled with `sys.srlTrsQAdd` and cleared with `sys.srlTrsQClr`. Max 32 bytes.

Return: None

### **USB serial set speed**

```
sys.com.usbSetBd([num] bd);
```

Sets baud rate of the usb-serial port

Return: none

### **USB serial receive**

```
sys.com.usbRcv([num]timeout);
```

Gets string (max 512 bytes) from USB serial port. If nothing is sent during timeout (in ms), empty string is returned.

Return: [str] data

### **USB serial receive init**

```
sys.com.usbRcvIT();
```

Initializes usb serial port receive operation in non-blocking mode Returns 1 if ok, 0 if error occurred

Return: [num] result

### **USB serial get ready flag**

```
sys.com.usbGetRd();
```

Gets transmission ready flag. Returns 1 if data is pending, 2 if whole line of data is pending

Return: [num] ready

### **USB serial get pending data**

```
sys.com.usbGetStr();
```

Gets the pending string and resets the serial interface for another ready flag.

Return: [str] pending

### **USB serial get pending data**

```
sys.com.usbGetBytes();
```

Gets the bytes from a serial interface and stores them in local buffer (512 Bytes max)

Return: [num] bytes used

### **USB serial get pending data**

```
sys.com.usbGetByte([num] index);
```

Reads the byte value from a serial interface local buffer (512 Bytes)

Return: [num] byte value (0 - 255, -1 when error occurs)

### ***Expansion port serial interface***

#### **Serial expansion transmit**

Code to init the internal expansion port serial interface:

```
sys.hw.iPinDef(15, PIN_ALT, PIN_NOPULL);
```

```
sys.hw.iPinDef(16, PIN_OUT, PIN_NOPULL);
```

Code to init the external expansion port serial interface:

```
sys.hw.ePinDef(5, PIN_ALT, PIN_NOPULL);
```

```
sys.hw.ePinDef(6, PIN_ALT, PIN_NOPULL);
```

#### **Serial expansion transmit**

```
sys.com.uartTrs([str]data);
```

Sends given string to serial port on internal or external expansion connector. Depends on what is initialized.

Return: None

#### **Serial expansion transmit queue**

```
sys.com.uartTrsQ();
```

Sends previously stored queue to the initialized serial port. Max 32 bytes.

Return: None

#### **Serial expansion receive**

```
sys.com.uartRcv([num]timeout);
```

Gets string (max 512 bytes) from currently initialized serial port. If nothing is sent during timeout (in ms), empty string is returned.

Return: [str] data

### **Serial expansion set speed**

```
sys.com.uartSetBd([num] bd);
```

Sets baud rate of the uart expansion port

Return: none

### ***Serial receive in non-blocking mode***

For more flexible serial interface operations

### **Serial expansion receive**

```
sys.com.uartRcvIT();
```

Initializes serial port receive operation in non-blocking mode

Return: [num] result (1 if ok, 0 if error occurred)

### **Serial expansion get ready flag**

```
sys.com.uartGetRd();
```

Gets transmission ready flag. Returns 1 if data is pending, 2 if whole line of data is pending

Return: [num] ready

### **Serial expansion get pending data**

```
sys.com.uartGetStr();
```

Gets the pending string and resets the serial interface for another ready flag.

Return: [str] pending

### **Serial expansion get pending data**

```
sys.com.uartGetBytes();
```

Gets the bytes from a serial interface and stores them in local buffer (512 Bytes)

Return: [num] bytes used

### **Serial expansion get pending data**

```
sys.com.uartGetByte([num] index);
```

Gets the byte value from a serial interface local buffer (512 Bytes)

Return: [num] byte value (0 - 255, -1 when error occurs)



### **Serial expansion register wakeup callback**

```
sys.com.uartRdClb([str] callback, [num] val);
```

Registers callback on expansion uart, when rd flag goes to *val*, application will be waken up and callback will be called. Application can fully resume by calling `sys.os.arise()`; Serial port needs to be previously configured for IT receive. Setting *val* to zero will disable the callback.

Return: [num] success

## SDA Files

Since SDA\_OS 1.0.2, more than one general purpose file is supported. Functions for basic file i/o operations now accept optional index parameter that specifies what file is used. Number of files currently opened is defined in SDA\_FILES\_OPEN\_MAX define. Default value is 10.

### Open file

```
sys.fs.open([str]fname);  
sys.fs.open([num]index, [str]fname);
```

Opens text file for read or write. If no file index is given, index 0 is used.

Return: 1 on success, 0 on failure

### Read given number of chars

```
sys.fs.readChars([num] bytes);  
sys.fs.readChars([num]index, [num] bytes);
```

Reads given number of chars from file. If no file index is given, index 0 is used.

Return: [str] result

### Writes given string to file

```
sys.fs.writeChars([str]string);  
sys.fs.writeChars([num]index, [str]string);
```

Writes given string to file.

Return: 1 - ok, 0 - fail

### Read byte from file

```
sys.fs.readByte();  
sys.fs.readByte([num]index);
```

Reads byte from file.

Return: [num] result: 0 to 255 - ok, -1 - error, -2 - EOF

### Write byte to file

```
sys.fs.writeByte([num] byte (0 - 255));  
sys.fs.writeByte([num]index, [num] byte (0 - 255));
```

Writes byte to file.

Return: [num] 0 - fail, 1 - ok

### Seek position in file

```
sys.fs.seek([num] pos_from_start);  
sys.fs.seek([num]index, [num] pos_from_start);
```

Writes byte to file.

Return: [num] 0 - fail, 1 - ok

### **Truncate file**

```
sys.fs.truncate();
```

Truncate currently opened file at the position of write pointer.

Return: [num] 0 - fail, 1 - ok

### **Tels position in file**

```
sys.fs.tell();
```

```
sys.fs.tell([num]index);
```

Returns current write pointer position in the file.

Return: [num] pos

### **Get size of file**

```
sys.fs.size();
```

```
sys.fs.size([num] index);
```

Returns size of opened file.

Return: [num] size in bytes

### **Get last modification time**

```
sys.fs.mtime([str] fname);
```

Returns last modified time of a given fname. In sda timestamp format.

Return: [num] timestamp

### **Close file**

```
sys.fs.close();
```

```
sys.fs.close([num] index);
```

Closes open file.

Return: [num] 1 - ok, 0 - error

## ***Directory functions***

### **Get if path is dir**

```
sys.fs.isDir([str] path);
```

Gets if path is a directory or not.

Return: [num] 0 - file, 1 - dir

### **Create directory**

```
sys.fs.mkdir([str] name);
```

Creates new directory

Return: [num] 1 - ok, 0 - fail

### **Change working directory**

`sys.fs.chdir([str] pathInData);`

Changes working directory. call `sys.fs.chdir(0)`; or `sys.fs.chdir()`; to get to the DATA context call `sys.fs.chdir(1)`; to get to the APPS context

Return: [num] 1 - ok, 0 - fail

### **Get current working directory**

`sys.fs.getCWD();`

Returns current working directory

Return: [str]path

### **Get current working context**

`sys.fs.getCWC();`

Returns current working context

Return: [num]context 0 - DATA, 1 - APPS

### ***File copy***

#### **File copy select source**

`sys.fs.copySource([str]source);`

Selects source file for copy operation.

Return: [num] 1 - ok, 0 - failed

#### **File copy start**

`sys.fs.copyStart([str]dest, [num]ChunkSize);`

Starts copy operation, chunksize of bytes will be copied each cycle.

Return: [num] 1 - ok, 0 - failed

#### **File copy status**

`sys.fs.copyStat([num]opt);`

opt: 0 - status ret: [num]0 - nothing, 1 - source selected, 2 - copy in progress

opt: 1 - size of source [num]bytes

opt: 2 - remaining bytes [num]bytes

#### **Stop current copy operation**

`sys.fs.copyStop();`

Stops current copy operation.

Return: None

## ***Check, remove, rename***

### **Check if file exist**

```
sys.fs.exists([str]fname);
```

Checks if file exists.

Return: 1 if file exists, otherwise 0

### **Remove file**

```
sys.fs.delete([str]fname);
```

Deletes file with filename. Can also delete empty directories.

Return: None

### **Move/rename file**

```
sys.fs.rename([str]oldPath, [str]newPath);
```

Moves/renames given file.

Return: None

## ***Find files***

### **Find begin**

```
sys.fs.find([str]extension, [str]directory);
```

Initiates file find operation, returns first result.

Return: [str]filename or "" if none

### **Find next**

```
sys.fs.findNext();
```

Next iteration of file find operation.

Return: [str]filename or "" if none

## Example

```
for(findfil = sys.fs.find("txt", "."); findfil != ""; findfil =  
sys.fs.findNext();) {  
    print("found: " + findfil);  
}
```

Find is not stateless, sys.fs.find must be re-initialized after recursive call. Example of recursive function:

```
function ls {  
    local findfil;  
    local n = 0;  
    print(arg0);  
    for(findfil = sys.fs.find("", arg0); findfil != ""; findfil =  
sys.fs.findNext();) {  
        if (sys.fs.isDir(arg0 + "/" + findfil)) {  
            ls(arg0 + "/" + findfil);  
            findfil = sys.fs.find("", arg0);  
            local x;  
            for (x = 0; x<n; x++;) {  
                findfil = sys.fs.findNext();  
            }  
        } else {  
            print(arg0 + "/" + findfil);  
        }  
        n++;  
    }  
}
```

## *Files as strings*

### Reads file as string

```
sys.fs.readStr([str]fname);
```

Reads text file to svcs string buffer.

Return: [str]FileContents

### Write string as file

```
sys.fs.writeStr([str]str, [str]fname);
```

Writes svcs string to file.

Return: None

## **SDA CSV files API**

SDA\_OS implements basic csv-like file api.

### **Open csv file**

```
sys.fs.csv.open([str]fname);
```

Opens csv file.

Return: [num]1 on succes.

### **Set separator**

```
sys.fs.csv.setSeparator([str]separator);
```

Sets the csv separator default is "|".

Return: none.

### **Close csv file**

```
sys.fs.csv.close();
```

Closes csv file.

Return: [num]1 on succes.

### **New csv line**

```
sys.fs.csv.newLine([num]numberOfCells);
```

Adds new line to csv with given number of cells.

Return: None.

### **Get csv cell**

```
sys.fs.csv.getCell([num]cellNumber, [str]default);
```

Gets data from specified cell on current line.

Return: [str]cellContents

### **Set csv cell**

```
sys.fs.csv.setCell([num]cellNumber, [str]value);
```

Sets data of specified cell on current line. Cells are counted from 0.

Return: [str]cellContents

### **Feed line**

```
sys.fs.csv.lineFeed();
```

Moves to the next lone of csv file

Return: [num] 1 - ok, 0 - end of file

### **Remove line**

```
sys.fs.csv.removeLine();
```

Removes current line from csv

Return: None

### **Rewind file**

```
sys.fs.csv.rewind();
```

Rewinds file back on the start.

Return: None

## ***Config files API***

### **Open config file**

```
sys.fs.conf.open([str]fname);
```

Opens config file.

Return: [num]1 on succes.

### **Close config file**

```
sys.fs.conf.close();
```

Close conf file.

Return: [num]1 on succes.

### **Check if key exists**

```
sys.fs.conf.exists([str]key);
```

Checks if key exists in conf file

Return: [num] 1 if key exists.

### **Read key**

```
sys.fs.conf.read([str]key);
```

```
sys.fs.conf.read([str]key, [str]default);
```

Reads key from config file as a string, 1024 chars max, when no default value is provided and the key doesn't exist, empty string is returned.

Return: [str]Value

### **Read Key as int**

```
sys.fs.conf.readInt([str]key, [num]default);
```

Reads key from config file as num (integer). To be removed.

Return: [num]Value



**Write key**

```
sys.fs.conf.write([str]key, [str]val);
```

Writes value in specified key.

Return: None

**Remove key**

```
sys.fs.conf.remove([str]key);
```

Removes given key.

Return: None

**Get if key value matche**

```
sys.fs.conf.valMatch([str]key, [str]value, [num]caseSensitive);
```

Returns 1 if value matches portion of a value in a given key. Case sensitive switch switches if the thing is case sensitive...

Return: [num] isMatch (0 - no match, 1 - match, -1 - key not found)

## ***Binary DB API***

Sda supports its own binary data format. It is quick, universal but not easily readable or recoverable when data corruption occurs.

### **Create new db file**

```
sys.fs.db.new([str]fname);
```

Creates new db file.

Return: [num]1 on succes.

### **Open db file**

```
sys.fs.db.open([str]fname);
```

Opens existing db file.

Return: [num]1 on succes.

### **Sync file**

```
sys.fs.db.sync();
```

Writes all the file changes to the card. Same as when file is closed, but can be triggered on demand.

Return: [num]1 on succes.

### **Close db file**

```
sys.fs.db.close();
```

Close db file.

Return: [num]1 on succes.

### **Create new table**

```
sys.fs.db.newTable([str]name, [num] columns);
```

Creates new table

Return: [num] 1 if ok.

### **Drop Data**

```
sys.fs.db.dropAllRows();
```

Drops all data from currently selected table.

Return: [num] 1 if ok.

### **Drop table**

```
sys.fs.db.dropTable();
```

Drops currently selected table.

Return: [num] 1 if ok.

### **Set column type**

```
sys.fs.db.setColumn([num] id, [str]name, [num] type);
```

Sets name and type of given column

Return: [num] 1 if ok.

### **Enable ID field**

```
sys.fs.db.idEnable([str]fieldName);
```

Sets given column as an id field.

Return: [num] 1 if ok.

### **Select table**

```
sys.fs.db.selectTable([str]name);
```

Selects existing table

Return: [num] 1 if ok.

### **New row**

```
sys.fs.db.newRow();
```

Adds new row to the selected table. New row is selected automatically.

Return: [num] 1 if ok.

### **Drop row**

```
sys.fs.db.dropRow();
```

Drops current row from the selected table.

Return: [num] 1 if ok.

### **Get Row count**

```
sys.fs.db.getRowCount();
```

Gets row count of the selected table.

Return: [num] row count.

### Select row

```
sys.fs.db.selectRow([num]row_n);
```

Select row with given number (not an id). Usefull for selecting row 0 and then using *sys.fs.db.nextRow()*; to read the full table line by line. Note: When app is suspended, selected row is forgotten.

Return: [num] 1 if ok.

### Select row by id

```
sys.fs.db.selectRowId([num]id);
```

Select row with given id. (Id field must be enabled)

Return: [num] 1 if ok.

### Next row

```
sys.fs.db.nextRow();
```

Selects next available row.

Return: [num] 1 if ok.

### Select next matching row

```
sys.fs.db.selectRowNum([str]column, [num]val);
```

Selectcs next row where given column has given value

Return: [num] 1 if ok.

### Select next row matching string

```
sys.fs.db.selectRowStr([str]column, [str]string, [num]partial,  
[num]case_sensitive);
```

Selectcs next row where given column has given value.

Parameter	Value	Meaning
partial	1	string can be only a part of the entry
	0	full string must be contained
case_sensitive	1	strings are matched case-sensitive
	0	strings are matched non case-sensitive

Return: [num] 1 if entry was found.

**Write text entry**

```
sys.fs.db.setEntryStr([str]col_name, [str]value);
```

Sets db entry

Return: [num] 1 if ok.

**Write numeric entry**

```
sys.fs.db.setEntryNum([str]col_name, [num]value);
```

Sets db entry

Return: [num] 1 if ok.

**Read text entry**

```
sys.fs.db.getEntryStr([str]col_name, [str]default);
```

Gets db entry

Return: [str] entry or default

**Read num entry**

```
sys.fs.db.getEntryNum([str]col_name, [num]default);
```

Gets db entry

Return: [num] entry or default

## ***Overlay API***

### **Set overlay screen**

```
sys.o.setScr([num]screen_id);
```

Sets overlay screen, returns overlay id.

Return: [num] Overlay id

### **Get overlay id**

```
sys.o.getId();
```

Gets id of current overlay.

Return: [num] OverlayId

### **Destroy overlay**

```
sys.o.destroy();
```

Destroys current overlay, also destroys its screen.

Return: None

### **Set position and size of overlay**

```
sys.o.setXXYY([num]x1, [num]y1, [num]x2, [num]y2);
```

Sets position and size of current overlay (in screen pixels).

Return: None

### **Set overlay height**

```
sys.o.setY([num]val);
```

Sets lower coordinates of current overlay (in screen pixels).

Return: None

## ***Date overlay***

### **Create date overlay**

```
sys.o.date.add([num]year, [num]month, [num]day);
```

Creates date overlay id, returns id

Return: [num]Date overlay id

### **Update date overlay**

```
sys.o.date.update([num]id);
```

Updates date overlay

Return: None

**Get ok from overlay**

```
sys.o.date.getOk([num]id);
```

Gets if overlay ok button was pressed.

Return: 1 if overlay ok button was pressed

**Clear ok from overlay**

```
sys.o.date.clrOk([num]id);
```

Clears ok flag from overlay

Return: None

**Get year from overlay**

```
sys.o.date.getYr([num]id);
```

Gets year from overlay with given id.

Return: [num]Year

**Get day from overlay**

```
sys.o.date.getDay([num]id);
```

Gets day from overlay with given id.

Return: [num]Day

**Get month from overlay**

```
sys.o.date.getMon([num]id);
```

Gets month from overlay with given id.

Return: [num]Month

***Time overlay*****Create time overlay**

```
sys.o.time.add();
```

```
sys.o.time.add([num]hr, [num]min);
```

Adds a time overlay, returns its id

Return: [num]overlay id

**Set time overlay time**

```
sys.o.time.set([num]overlay_id, [num]hr, [num]min);
```

Sets time in overlay with given id.

Return: None

### **Update time overlay**

`sys.o.time.update([num]overlay_id);`  
Updates time overlay.

Return: None

### **Get time overlay ok**

`sys.o.time.getOk([num]overlay_id);`  
Gets ok from time overlay.

Return: 1 if ok was pressed

### **Get time overlay minutes**

`sys.o.time.getMin([num]overlay_id);`  
Returns minutes from overlay.

Return: [num]Minutes

### **Get time overlay hours**

`sys.o.time.getHr([num]overlay_id);`  
Returns hours from overlay.

Return: [num]Hours

### **Clear ok**

`sys.o.time.clrOk([num]overlay_id);`  
Clears ok from time overlay

Return: None

### ***Color overlay***

#### **Create color overlay**

`sys.o.color.add([num]color);`  
Adds a color overlay, returns its id

Return: [num]overlay id

#### **Set color overlay color**

`sys.o.color.set([num]overlay_id, [num]color);`  
Sets color in overlay with given id.

Return: None



### **Update color overlay**

```
sys.o.color.update([num]overlay_id);
```

Updates color overlay.

Return: None

### **Get color overlay ok**

```
sys.o.color.getOk([num]overlay_id);
```

Gets ok from color overlay.

Return: 1 if ok was pressed

### **Get color overlay value**

```
sys.o.color.getCol([num]overlay_id);
```

Returns color from overlay.

Return: [num]Color

### **Clear ok**

```
sys.o.color.clrOk([num]overlay_id);
```

Clears ok from the overlay

Return: None

## ***Time API***

### **Get time**

```
sys.time.get();
```

Returns system time in the timestamp form. Count of seconds from 00:00 1. 1. 2007

Return: [num]Timestamp

### **Get time update flag**

```
sys.time.getUpd();
```

Returns 1 when time update flag occurs.

Return: 1 on time update (roughly 1s)

### **Get seconds**

```
sys.time.getSec();
```

Returns system time.

Return: [num]Seconds

**Get minutes**

```
sys.time.getMin();
```

Returns system time.

Return: [num]Minutes

**Get hours**

```
sys.time.getHr();
```

Returns system time

Return: [num]Hours

**Get day**

```
sys.time.getDay();
```

Returns system time.

Return: [num]Day

**Get month**

```
sys.time.getMonth();
```

Returns system time.

Return: [num]Month

**Get year**

```
sys.time.getYear();
```

Returns system time.

Return: [num]Year

**Get uptime**

```
sys.time.getUptime();
```

Returns system uptime.

Return: [num]Uptime(secs)

**Get uptime in milliseconds**

```
sys.time.getAppUptime();
```

Returns milisecond uptime of the app.

Return: [num]Uptime(milliseconds)

### **Get new timestamp**

```
sys.time.setTs([num]Year, [num]Month, [num]Day, [num]Hour, [num]Min,  
[num]Sec);
```

Returns timestamp of given time, works only for years 2007 and above.

Return: [num]Timestamp

### **Get seconds from timestamp**

```
sys.time.getTsSec([num]timestamp);
```

Returns seconds from given timestamp.

Return: [num]Seconds

### **Get minutes from timestamp**

```
sys.time.getTsMin([num]timestamp);
```

Returns seconds from given timestamp.

Return: [num]Minutes

### **Get hours from timestamp**

```
sys.time.getTsHr([num]timestamp);
```

Returns seconds from given timestamp.

Return: [num]Seconds

### **Get days from timestamp**

```
sys.time.getTsDay([num]timestamp);
```

Returns seconds from given timestamp.

Return: [num]Days

### **Get weekday from timestamp**

```
sys.time.getTswkDay([num]timestamp);
```

Returns weekday from given timestamp.

Return: [num]Weekday (starting with monday)

### **Get month from timestamp**

```
sys.time.getTsMonth([num]timestamp);
```

Returns seconds from given timestamp.

Return: [num]Months

### **Get year from timestamp**

```
sys.time.getTsYear([num]timestamp);
```

Returns seconds from given timestamp.

Return: [num]Years

### ***Timer API***

System timer will call the callback after given time, it will wake up the app if it is in the background. You can push app to foreground by calling `sys.os.arise()`; from the callback.

### **Set timer**

```
sys.time.setTimer([num]time_ms, [str]callaback);
```

Sets the timer.

Return: none

### **Clear timer**

```
sys.time.clearTimer();
```

Clears the timer if it is running.

Return: none

### ***Alarm API***

Alarm API creates system handled, repeatable alarms, that will be stored during reboots and will automatically launch the app. When alarm is created, alarm id is returned, this id should be stored for future operations with the alarm. Every alarm can have an user-defined parameter.

### **Register alarm**

```
sys.alarm.setFixed([num]timestamp, [num]param);
```

Creates new one-time alarm. Returns id of the new alarm.

Return: [num]id

### **Register repeating alarm**

```
sys.alarm.setRep([num]hour, [num]min, [num]wkday, [num]day, [num]month,  
[num]param);
```

Creates new repeatable alarm. Zero value in wkday/day/month means repeat every wkday/day/month. Returns id of the new alarm.

Return: [num]id

**Remove alarm**

```
sys.alarm.destroy([num]id);
```

Returns if alarm was deleted.

Return: [num] 0 - Ok, 1 - Fail

**Get alarm flag**

```
sys.alarm.getFlag();
```

Returns 1 when alarm has occurred.

Return: [num]flag

**Clear alarm flag**

```
sys.alarm.clrFlag();
```

Clears alarm flag.

Return: none

**Get alarm id**

```
sys.alarm.getId();
```

Returns id of the current alarm.

Return: [num]id

**Get alarm parameter**

```
sys.alarm.getPar();
```

Returns parameter of the current alarm.

Return: [num]param Automatically generated documentation for GR2 SVS wrapper, follows markdown syntax.

## GR2 Graphics library

### *Constants*

#### Element events

Constant	Value	Meaning
EV_PRESSED	1	Event: pressed
EV_RELEASED	3	Event: released
EV_HOLD	2	Event: hold
EV_NONE	0	Event: none

#### UI Colors

Constant	Value	Meaning
COL_BORDER	1	Color: Border
COL_TEXT	2	Color: Text
COL_BACKGROUND	3	Color: Background
COL_FILL	4	Color: Fill
COL_ACTIVE	5	Color: active

#### Text alignment

Constant	Value	Meaning
ALIGN_LEFT	5	Text align: Left
ALIGN_RIGHT	5	Text align: Right
ALIGN_CENTER	5	Text align: Center

## Available fonts

Constant	Value	Meaning
FONT_12	12	Font 12px
FONT_12_M	13	Font 12px, monospace
FONT_18	18	Font 18px (Default)
FONT_32	32	Font 32px
FONT_70	70	Font 70px
FONT_87	87	Font 87px

## Element constructors

### New screen

`sys.gui.addScreen()`;  
or also `sys.gui.addScreen([num]x1, [num]y1, [num]x2, [num]y2, [num]scrId)`; Creates new screen.

Return: `[num]scrId`

### New frame

`sys.gui.addFrame([num]x1, [num]y1, [num]x2, [num]y2, [num]value, [num]scrId)`;  
Creates new pscg frame. Value contains id of screen inside frame.

Return: `[num]elementId`

### New text field

`sys.gui.addText([num]x1, [num]y1, [num]x2, [num]y2, [str]str, [num]scrId)`;  
Adds a new text field.

Return: `[num]id`

### New button

`sys.gui.addButton([num]x1, [num]y1, [num]x2, [num]y2, [str]str, [num]scrId)`;  
Creates new button.

Return: `[num]id`

### New color button

`sys.gui.addCButton([num]x1, [num]y1, [num]x2, [num]y2, [str]str, [num]scrId)`;  
Adds color button, color is stored in its value.

Return: `[num]id`

### **New check box**

```
sys.gui.addCheckBox([num]x1, [num]y1, [num]x2, [num]y2, [str]str, [num]scrId);
```

Creates new checkbox. Checkbox state is stored in its value.

Return: [num]id

### **New icon**

```
sys.gui.addIcon([num]x1, [num]y1, [num]x2, [num]y2, [str]description,  
[str]image, [num]scrId);
```

Adds icon. Image must be a file in current working directory. Optimal resolution is 64x64px. Icon will try to accomodate given space by scaling itself. use sys.gui.setTextAlign to change position of the description box. When parameter of icon element is not zero, color of value param - 1 (16bit RGB565) is drawn as transparent.

Return: [num]id

### **Set icon**

```
sys.gui.setIcon([num]id, [str]image);
```

Sets image for given icon. Image must be a file in current working directory, with resolution 64x64px. When parameter of icon element is not zero, color of value param - 1 (16bit RGB565) is drawn as transparent.

Return: none

### **New image**

```
sys.gui.addImage([num]x1, [num]y1, [num]x2, [num]y2, [str]fname, [num]scrId);
```

Creates new image (ppm or p16) container. Name of image is stored in str\_value Size attribute is stored in value, (one by default) Color of value param - 1 (16bit RGB565) is drawn as transparent.

Return: [num]id

### **New vertical slider**

```
sys.gui.addSliderV([num]x1, [num]y1, [num]x2, [num]y2, [num]howMuchOverall,  
[num]howMuch, [num]scrId);
```

Adds a new vertical slider.

Return: [num]id

### **New horizontal slider**

```
sys.gui.addSliderH([num]x1, [num]y1, [num]x2, [num]y2, [num]howMuchOverall,  
[num]howMuch, [num]scrId);
```

Adds a new horizontal slider.

Return: [num]id



## **New progress bar**

```
sys.gui.addBar([num]x1, [num]y1, [num]x2, [num]y2, [num]howMuchOverall,  
[num]howMuch, [num]scrId);
```

Adds progress bar. Orientation depends on its dimensions.

Return: [num]id

## ***Destructors***

```
sys.gui.destroy([num]id);
```

Destroys element with given id.

Return: none

## **Getters & setters**

### ***Modified flag***

```
sys.gui.setModif([num]Id);
```

Sets modified flag of an element. Return: None

### ***Element property: Value***

#### **Get value**

```
sys.gui.getValue([num]Id);
```

Gets value of gr2 element.

Return: [num]value

#### **Set value**

```
sys.gui.setValue([num]Id, [num]value);
```

Sets value of pscg item.

Return: None

### ***Element property: Parameter***

#### **Get parameter**

```
sys.gui.getParam([num]Id);
```

Gets elements parameter value.

Return: [num]param

#### **Set parameter**

```
sys.gui.setParam([num]Id, [num]value);
```

Sets elements parameter value. Return: None

## ***Events Handling***

### **Get event**

`sys.gui.getEvent([num]Id);`

Gets event from an element.

Return: [num] event value, one of EV\_ defines.

### **Get event and clear**

`sys.gui.getEventC([num]Id);`

Gets event from an element and clears the event.

Return: [num] event value, one of EV\_ defines.

### **Set event**

`sys.gui.setEvent([num]Id, [num] event);`

Sets event of an element.

Return: None

### **Clear screen events**

`sys.gui.clrScrEv([num]Id);`

Clears event for whole screen and its sub-screens.

Return: None

### **Set screen**

`sys.gui.setScreen([num]Id, [num]screenId);`

Sets element screen parameter.

Return: None

## ***Element property: Grayout***

### **Get grayout**

`sys.gui.getGrayout([num]Id);`

Gets element grayout.

Return: [num] grayout, 1 if element is grey, 0 if it is normal

### **Get grayout**

`sys.gui.setGrayout([num]Id, [num]grayout);`

Sets element grayout.

Return: None

## **Element property: Visibility**

### **Get visibility**

```
sys.gui.getVisible([num]Id);
```

Gets element visibility.

Return: [num] visibility

### **Set visibility**

```
sys.gui.setVisible([num]Id, [num]visibility);
```

Sets element visibility.

Return: None

## ***Element property: Ghost***

When ghost is enabled, only button text and its outline is drawn.

### **Get ghost**

```
sys.gui.getGhost([num]Id);
```

Gets element ghost parameter.

Return: [num] isGhost

### **Set ghost**

```
sys.gui.setGhost([num]Id, [num]ghost);
```

Sets element ghost parameter.

Return: None

## ***Element property: Select***

### **Get select**

```
sys.gui.getSelected([num]Id);
```

Gets element selected parameter.

Return: [num] isSelected

### **Set select**

```
sys.gui.setSelected([num]Id, [num]select);
```

Sets element select parameter.

Return: None

## ***Slider size***

### **Set slider size**

```
sys.gui.setSliderSize([num]Id, [num]val);
```

Sets size of slider in pixels.

Return: None

### **Set rounded**

```
sys.gui.setRounded([num]Id, [num]rounded);
```

Sets element rounded parameter.

Return: None

## ***Element property: String parameter***

### **Set String**

```
sys.gui.getStr([num]Id);
```

Gets element value\_str parameter.

Return: [str]str

### **Get String**

```
sys.gui.setStr([num]Id, [str]str);
```

Sets element value\_str parameter.

Return: None

## **Keypad control**

```
sys.gui.getBtnSel([num]screenId);
```

Gets element selected by the keypad input method from the given screen.

Return: [num] Id if something is selected, otherwise 0

## ***Size and placement***

### **Set relative init**

```
sys.gui.setRelInit([num]val);
```

Sets application gr2 context to relative init mode. In relative init mode the x2 parameter is used as element width and the y2 as element height.

Return: None

### **Set rounded init**

```
sys.gui.setRndInit([num]val);
```

Sets application gr2 context rounded init mode. In rounded init mode all elements are initialized as rounded type.

Return: None

### **Set screen background redraw**

```
sys.gui.setBgRedraw([num]screen_id, [num]val);
```

Enables or disables redrawing background on a screen update. Usefull for improving redraw speed in scrolling texts.

Return: None

### **Set element size and position**

```
sys.gui.setXYXY([num]Id, [num] x1, [num] y1, [num] x2, [num] y2);
```

Sets position of element inside screen.

Return: None

```
sys.gui.setX1([num]Id, [num] x1);
```

Sets position of element inside screen.

Return: None

```
sys.gui.setX2([num]Id, [num] x2);
```

Sets position of element inside screen.

Return: None

```
sys.gui.setY1([num]Id, [num] y1);
```

Sets position of element inside screen.

Return: None

```
sys.gui.setY2([num]Id, [num] y2);
```

Sets position of element inside screen.

Return: None

### **Get element size and position**

`sys.gui.getX1([num]Id);`

Gets element position.

Return: [num] x1

`sys.gui.getX2([num]Id);`

Gets element position.

Return: [num] x2

`sys.gui.getY1([num]Id);`

Gets element position.

Return: [num] y1

`sys.gui.getY2([num]Id);`

Gets element position.

Return: [num] y2

### ***Screen spacing & cell size***

#### **Set spacing**

`sys.gui.setSpacing([num]Id, [num] left, [num] right, [num] top, [num] bottom);`

Sets element spacing attributes of given screen.

Return: None

#### **Set cell width**

`sys.gui.setXcell([num]screenId, [num] Xcell);`

Sets screen Xcell parameter. (32 by default)

Return: None

#### **Get cell width**

`sys.gui.getXcell([num]screenId);`

Gets screen Xcell parameter.

Return: [num] Xcell

#### **Set cell height**

`sys.gui.setYcell([num]screenId, [num] Ycell);`

Sets screen Ycell parameter. (32 by default)

Return: None

### **Get cell height**

```
sys.gui.getYcell([num]screenId);
```

Gets screen Ycell parameter.

Return: [num] Ycell

### **Set screen xscroll**

```
sys.gui.setXscroll([num]screenId, [num]Xscroll);
```

Sets Xscroll parameter. Determines screen horizontal offset.

Return: None

### **Get screen xscroll**

```
sys.gui.getXscroll([num]screenId);
```

Gets screen Xscroll.

Return: [num] Xscroll

### **Set screen yscroll**

```
sys.gui.setYscroll([num]screenId, [num]Yscroll);
```

Sets Yscroll parameter. Determines screen vertical offset.

Return: None

### **Get screen yscroll**

```
sys.gui.getYscroll([num]screenId);
```

Gets Yscroll parameter.

Return: [num] Yscroll

## **Colors**

### ***Sets gr2 context color***

```
sys.gui.setColor([num]Col, [num]val);
```

Sets given color to given value (16bit RGB565). Available system colors: COL\_BORDER, COL\_TEXT, COL\_BACKGROUND, COL\_FILL, COL\_ACTIVE

Return: None

### ***Gets gr2 context color***

```
sys.gui.getColor([num]Col);
```

Gets value of given color define. Available system colors: COL\_BORDER, COL\_TEXT, COL\_BACKGROUND, COL\_FILL, COL\_ACTIVE

Return: [num]val

## ***Global text modifiers***

Functions that works on all elements that display text.

### **Set text size**

```
sys.gui.setTxtSize([num]Id, [num]val);
```

Sets size of text inside buttons or text fields. Possible values are those used by LCD\_Set\_Sys\_Font By default they are: 12, 18 (default), 32, 70, 87

Return: None

### **Get text size**

```
sys.gui.getTxtSize([num]Id);
```

Gets size of text inside buttons or text fields.

Return: [num] font\_size

### **Set text alignment**

```
sys.gui.setTxtAlign([num]Id, [num]val);
```

```
sys.gui.setTexAlign([num]Id, [num]val); # TBR
```

Sets text alignment. (uses consts: ALIGN\_LEFT, ALIGN\_RIGHT, ALIGN\_CENTER)

Return: None

### **Set text alignment**

```
sys.gui.getTxtAlign([num]Id);
```

```
sys.gui.getTexAlign([num]Id); # TBR
```

Gets text alignment.

Return: [num]alignment (uses consts: ALIGN\_LEFT, ALIGN\_RIGHT, ALIGN\_CENTER)

## ***Misc***

### **Set default text size for a screen**

```
sys.gui.defTxtSize([num]screenId, [num] val);
```

```
sys.gui.setDefFont([num]screenId, [num] val); # TBR
```

Sets default screen text size.

Return: None

## ***Text element modifiers***

Functions that work only on text element.



### **Set text field as password**

```
sys.gui.setTxtPwd([num]Id, [num]val);  
sys.gui.setTexPwd([num]Id, [num]val); # TBR
```

Sets text field as password field. Draws stars instead of characters.

Return: None

### **Get password value**

```
sys.gui.getTxtPwd([num]Id);  
sys.gui.getTexPwd([num]Id); # TBR
```

Gets if text field is a password field.

Return: [num]isPassword

### **Set text fit**

```
sys.gui.setTxtFit([num]Id, [num]val);  
sys.gui.setTexFit([num]Id, [num]val); # TBR
```

Sets automatic line-breaking. val: 1 - enabled, 0 - disabled

Return: None

### **Set selected text inversion**

```
sys.gui.setTxtInvert([num]Id, [num]val);
```

When inverted text is selected, fill color is used for text field background.

Return: None

### **Get selected text inversion**

```
sys.gui.getTxtInvert([num]Id);
```

Get the text inversion value.

Return: [num] Text invert value

### **Get text width**

```
sys.gui.getTxtWidth([num]Id, [num]pos);
```

Get max width of string drawn in a given text element. Works with text field params like font size and text fit. pos: cursor position (0 for full string)

Return: Text width in px

### **Get text height**

```
sys.gui.getTxtHeight([num]Id, [num]pos);
```

Get max height of string drawn in a given text element. Works with text field params like font size and text fit. pos: cursor position (0 for full string)

Return: Text height in px

### **Set text editable**

```
sys.gui.setTxtEd([num]Id, [num]val);  
sys.gui.setTexEd([num]Id, [num]val); # TBR
```

Sets text field as editable.

Return: None

### ***Text element activation/deactivation***

#### **Set text active**

```
sys.gui.setTxtAct([num]Id);  
sys.gui.setTexAct([num]Id); # TBR
```

Sets given editable text field as currently active.

Return: None

#### **Get text active**

```
sys.gui.getTxtAct([num]Id);  
sys.gui.getTexAct([num]Id);
```

Gets if given editable text field is currently active.

Return: [num]isActive

#### **Deactivate active text**

```
sys.gui.txtDeact();  
sys.gui.texDeact(); # TBR
```

Deactivates currently active text field.

Return: None

### ***Text block functions***

#### **Enable block select mode**

```
sys.gui.setTxtBlk([num]Id, [num]val);  
sys.gui.setTexBlk([num]Id, [num]val); # TBR
```

Enables block selection in a text field. This is enabled by default, so it's mainly for disabling block mode.

Return: None

#### **Set text block**

```
sys.gui.setBlk([num] id, [num] start, [num] stop);  
Sets start and stop of a block in active text field
```

Return: None

### **Get block start**

```
sys.gui.getBlkStart([num] id);
```

Gets text field block start.

Return: [num] block\_start

### **Get block end**

```
sys.gui.getBlkEnd([num] id);
```

Gets text field block end.

Return: [num] block\_end Automatically generated documentation on wrap\_directS.c

## **Direct Screen draw functions**

### ***Color & Areas***

#### **Get color from RGB**

```
sys.ds.mixColor([num]r, [num]g, [num]b);
```

Mixes the right color from red, green and blue values (0 - 255)

Return: [num] Color (16bit RGB565)

#### **Set draw area**

```
sys.ds.setArea([num] x1, [num] y1, [num] x2, [num] y2);
```

Sets the draw area. Uses hardware coordinates. For example: sys.ds.setArea(0, 32, 319, 479);

This will init all app available space as a draw area.

Return: None

#### **Draw rectangle**

```
sys.ds.drawRect([num] x1, [num] y1, [num] x2, [num] y2, [num]col);
```

Draws rectangle outline

Return: None

#### **Draw filled rectangle**

```
sys.ds.fillRect([num] x1, [num] y1, [num] x2, [num] y2, [num]col);
```

Draws filled rectangle

Return: None

#### **Draw circle**

```
sys.ds.drawCircle([num] x1, [num] y1, [num] radius, [num]col);
```

Draws circle

Return: None

### **Draw filled circle**

```
sys.ds.fillCircle([num] x1, [num] y1, [num] radius, [num]col);
```

Fills circle

Return: None

### **Draw portion of a circle**

```
sys.ds.drawCPart([num] x1, [num] y1, [num] radius, [num] part, [num]col);
```

Draws part of a circle

Parts: | 0 | 1 | | 2 | 3 | Return: None

### **Fill portion of a circle**

```
sys.ds.fillCPart([num] x1, [num] y1, [num] radius, [num] part, [num]col);
```

Fills a part of a circle

Return: None

### **Draw line**

```
sys.ds.drawLine([num] x1, [num] y1, [num] x2, [num] y2, [num]col);
```

Draws line

Return: None

### **Draw text**

```
sys.ds.drawText([num] x1, [num] y1, [str] txt, [num]col);
```

Draws text

Return: None

### **Set text to fit specified width**

```
sys.ds.setTextFit([num] enable, [num] width);
```

Sets max width for next drawn text. (Redraw of UI elements might reset it)

Return: None

### **Get text width**

```
sys.ds.getTextWidth([str] txt);
```

Gets width of a string, when drawn with current font.

Return: [num] width (px)

### **Get text height**

```
sys.ds.getTextHeight([str] txt);
```

Gets height of a string, when drawn with current font.

Return: [num] height (px)

### Fill area with color

```
sys.ds.clearArea([num]col);
```

Clears draw area with given color

Return: None

### *P16 image tools*

#### Draws P16 image

```
sys.ds.drawImage([num]x, [num]y, [num]scale_w, [num]scale_h, [str]name);
```

Draws p16 image from the working directory. Supports upscaling, and downscaling.

Scale table:

Scale value	Image size
-3	1/16
-2	1/8
-1	1/4
0	1/2
1	1
2	2x
n	n*x

Return: None

#### Get P16 image width

```
sys.ds.getImageW([str]name);
```

Gets width of given p16 file.

Return: [num] width (px)

#### Get P16 image height

```
sys.ds.getImageH([str]name);
```

Gets height of given p16 file.

Return: [num] height (px)

#### Draws PPM

```
sys.ds.drawPPM([num]x, [num]y, [num]scale, [str]name);
```

Draws ppm image (To be removed).

Return: None

## ***Touch API***

### **Get if screen is touched**

`sys.ds.touchEv();`

Gets if screen is touched, returns last touch event

Return: [num] Touch event (from event defines)

### **Clears touch event**

`sys.ds.touchClr();`

Clears touch event

Return: none

### **Get touch y**

`sys.ds.touchY();`

Gives y coordinate of touch event

Return: [num] Touch y coordinate

### **Get touch x**

`sys.ds.touchX();`

Gives x coordinate of touch event

Return: [num] Touch y coordinate

## Contents:

SDA_OS Developer handbook.....	1
Version 2025-02-08 SDA_OS v.1.5.0.....	1
Basic Application structure.....	2
Required functions.....	2
Init function.....	2
Update function.....	2
Optional functions.....	2
Exit function.....	2
Suspend function.....	2
Wakeup function.....	2
SVP API Level history.....	3
Constants.....	3
Main OS functions.....	3
Get redraw flag.....	3
Set redraw.....	3
Wake the SDA from sleep.....	3
Pushes app to foreground.....	3
Disable app close.....	3
Get if running in simulator.....	4
Show Error.....	4
Gets app path.....	4
Keyboard.....	4
Show keyboard.....	4
Hide keyboard.....	4
Get Keyboard state.....	4
Misc.....	5
Get random number.....	5
Suspend app.....	5
Exit app.....	5
Check API level.....	5
Get API level.....	5
Get system language.....	5
Subprocess.....	6
Set process as singular.....	6
Launch subprocess.....	6
Enable launching subprocess from cwd.....	6
Disable caching for launched subprocess.....	6
Return data to parent process.....	6
Sets the clipboard string.....	7
Gets the clipboard string.....	7
OS settings functions.....	7
Reload homescreen settings.....	7

Requests high privileges.....	7
Gets if privileges are granted.....	7
Sets time and date.....	7
Gui.....	8
Set main application screen.....	8
Get main application screen.....	8
Set root for redraw.....	8
Handle keypad input of a screen.....	8
Selects element for keyboard control.....	8
Gets element selected by keyboard control.....	8
Clear button control for a screen.....	9
Text field handling.....	9
Handle text input.....	9
Set keyboard string.....	9
Paste clipboard.....	9
Get text cursor position.....	9
Set text cursor position.....	9
Switch between landscape and portrait mode.....	10
Get display orientation.....	10
Notification area icons.....	10
Set notification area icon.....	10
Free notification area icon.....	10
Default Icons.....	10
Add an os icon to a button.....	10
Sound.....	11
Beep the speaker.....	11
Beep the speaker with callback.....	11
Set beep param to default.....	11
Set the duration.....	11
Set the frequency.....	11
Get if system sound is disabled.....	11
Date selector widget.....	12
Init calendar widget.....	12
Select date.....	12
Update.....	12
Mark day.....	12
Set highlighting.....	12
Get selected day.....	12
Counters.....	13
Set counter.....	13
Gets counter.....	13
Text obfuscation.....	13
Unlock overlay init.....	13
Unlock overlay update.....	13



Unlock overlay get ok.....	13
Unlock overlay clear ok.....	13
Get if is locked.....	13
Loads password as a key.....	14
Load custom key string.....	14
Load custom keyfile.....	14
Load OS keyfile.....	14
Generate keyfile.....	14
Lock.....	14
Encrypt file.....	14
Decrypt file.....	14
Encrypt string.....	15
Decrypt string.....	15
SDA OS HW functions.....	15
Constants.....	15
Indicator LED.....	15
Buttons.....	15
Expansion pin states.....	16
Power Functions.....	17
Lock system suspend.....	17
Lock LCD sleep.....	17
Turn on the LCD.....	17
Get LCD state.....	17
Set LCD state.....	17
Set notification led pattern.....	17
Expansion Ports.....	18
Get USB State.....	18
Resource claiming.....	18
Claim hardware resource.....	18
Free hardware resource.....	18
Get hardware resource state.....	18
Internal expansion port.....	19
Define direction of pins on the internal expansion.....	19
Set state of pins on the internal expansion.....	19
Get state of pins on the internal expansion.....	19
External expansion pins.....	20
Define direction of pins on the expansion.....	20
Set state of pins on the expansion.....	20
Get state of pins on the expansion.....	20
Get ADC readout.....	20
Buttons.....	21
Get button event.....	21
Clears button events.....	21
Enable button events with LCD off.....	21

Communication.....	21
Serial transmit queue.....	21
Serial expansion transmit queue.....	21
Serial expansion transmit queue clear.....	21
USB serial interface.....	22
USB serial transmit.....	22
USB serial transmit queue.....	22
USB serial set speed.....	22
USB serial receive.....	22
USB serial receive init.....	22
USB serial get ready flag.....	22
USB serial get pending data.....	22
USB serial get pending data.....	23
USB serial get pending data.....	23
Expansion port serial interface.....	23
Serial expansion transmit.....	23
Serial expansion transmit.....	23
Serial expansion transmit queue.....	23
Serial expansion receive.....	23
Serial expansion set speed.....	24
Serial receive in non-blocking mode.....	24
Serial expansion receive.....	24
Serial expansion get ready flag.....	24
Serial expansion get pending data.....	24
Serial expansion get pending data.....	24
Serial expansion get pending data.....	24
Serial expansion register wakeup callback.....	25
SDA Files.....	26
Open file.....	26
Read given number of chars.....	26
Writes given string to file.....	26
Read byte from file.....	26
Write byte to file.....	26
Seek position in file.....	26
Truncate file.....	27
Tels position in file.....	27
Get size of file.....	27
Get last modification time.....	27
Close file.....	27
Directory functions.....	27
Get if path is dir.....	27
Create directory.....	27
Change working directory.....	28
Get current working directory.....	28

Get current working context.....	28
File copy.....	28
File copy select source.....	28
File copy start.....	28
File copy status.....	28
Stop current copy operation.....	28
Check, remove, rename.....	29
Check if file exist.....	29
Remove file.....	29
Move/rename file.....	29
Find files.....	29
Find begin.....	29
Find next.....	29
Example.....	30
Files as strings.....	30
Reads file as string.....	30
Write string as file.....	30
SDA CSV files API.....	31
Open csv file.....	31
Set separator.....	31
Close csv file.....	31
New csv line.....	31
Get csv cell.....	31
Set csv cell.....	31
Feed line.....	31
Remove line.....	32
Rewind file.....	32
Config files API.....	32
Open config file.....	32
Close config file.....	32
Check if key exists.....	32
Read key.....	32
Read Key as int.....	32
Write key.....	33
Remove key.....	33
Get if key value matche.....	33
Binary DB API.....	34
Create new db file.....	34
Open db file.....	34
Sync file.....	34
Close db file.....	34
Create new table.....	34
Drop Data.....	34
Drop table.....	35

Set column type.....	35
Enable ID field.....	35
Select table.....	35
New row.....	35
Drop row.....	35
Get Row count.....	35
Select row.....	36
Select row by id.....	36
Next row.....	36
Select next matching row.....	36
Select next row matching string.....	36
Write text entry.....	37
Write numeric entry.....	37
Read text entry.....	37
Read num entry.....	37
Overlay API.....	38
Set overlay screen.....	38
Get overlay id.....	38
Destroy overlay.....	38
Set position and size of overlay.....	38
Set overlay height.....	38
Date overlay.....	38
Create date overlay.....	38
Update date overlay.....	38
Get ok from overlay.....	39
Clear ok from overlay.....	39
Get year form overlay.....	39
Get day from overlay.....	39
Get month from overlay.....	39
Time overlay.....	39
Create time overlay.....	39
Set time overlay time.....	39
Update time overlay.....	40
Get time overlay ok.....	40
Get time overlay minutes.....	40
Get time overlay hours.....	40
Clear ok.....	40
Color overlay.....	40
Create color overlay.....	40
Set color overlay color.....	40
Update color overlay.....	41
Get color overlay ok.....	41
Get color overlay value.....	41
Clear ok.....	41

Time API.....	41
Get time.....	41
Get time update flag.....	41
Get seconds.....	41
Get minutes.....	42
Get hours.....	42
Get day.....	42
Get month.....	42
Get year.....	42
Get uptime.....	42
Get uptime in milliseconds.....	42
Get new timestamp.....	43
Get seconds from timestamp.....	43
Get minutes from timestamp.....	43
Get hours from timestamp.....	43
Get days from timestamp.....	43
Get weekday from timestamp.....	43
Get month from timestamp.....	43
Get year from timestamp.....	44
Timer API.....	44
Set timer.....	44
Clear timer.....	44
Alarm API.....	44
Register alarm.....	44
Register repeating alarm.....	44
Remove alarm.....	45
Get alarm flag.....	45
Clear alarm flag.....	45
Get alarm id.....	45
Get alarm parameter.....	45
GR2 Graphics library.....	46
Constants.....	46
Element events.....	46
UI Colors.....	46
Text alignment.....	46
Available fonts.....	47
Element constructors.....	47
New screen.....	47
New frame.....	47
New text field.....	47
New button.....	47
New color button.....	47
New check box.....	48
New icon.....	48

Set icon.....	48
New image.....	48
New vertical slider.....	48
New horizontal slider.....	48
New progress bar.....	49
Destructors.....	49
Getters & setters.....	49
Modified flag.....	49
Element property: Value.....	49
Get value.....	49
Set value.....	49
Element property: Parameter.....	49
Get parameter.....	49
Set parameter.....	49
Events Handling.....	50
Get event.....	50
Get event and clear.....	50
Set event.....	50
Clear screen events.....	50
Set screen.....	50
Element property: Grayout.....	50
Get grayout.....	50
Get grayout.....	50
Element property: Visibility.....	51
Get visibility.....	51
Set visibility.....	51
Element property: Ghost.....	51
Get ghost.....	51
Set ghost.....	51
Element property: Select.....	51
Get select.....	51
Set select.....	51
Slider size.....	52
Set slider size.....	52
Set rounded.....	52
Element property: String parameter.....	52
Set String.....	52
Get String.....	52
Keypad control.....	52
Size and placement.....	52
Set relative init.....	52
Set rounded init.....	53
Set screen background redraw.....	53
Set element size and position.....	53

Get element size and position.....	54
Screen spacing & cell size.....	54
Set spacing.....	54
Set cell width.....	54
Get cell width.....	54
Set cell height.....	54
Get cell height.....	55
Set screen xscroll.....	55
Get screen xscroll.....	55
Set screen yscroll.....	55
Get screen yscroll.....	55
Colors.....	55
Sets gr2 context color.....	55
Gets gr2 context color.....	55
Global text modifiers.....	56
Set text size.....	56
Get text size.....	56
Set text alignment.....	56
Get text alignment.....	56
Misc.....	56
Set default text size for a screen.....	56
Text element modifiers.....	56
Set text field as password.....	57
Get password value.....	57
Set text fit.....	57
Set selected text inversion.....	57
Get selected text inversion.....	57
Get text width.....	57
Get text height.....	57
Set text editable.....	58
Text element activation/deactivation.....	58
Set text active.....	58
Get text active.....	58
Deactivate active text.....	58
Text block functions.....	58
Enable block select mode.....	58
Set text block.....	58
Get block start.....	59
Get block end.....	59
Direct Screen draw functions.....	59
Color & Areas.....	59
Get color from RGB.....	59
Set draw area.....	59
Draw rectangle.....	59

Draw filled rectangle.....	59
Draw circle.....	59
Draw filled circle.....	60
Draw portion of a circle.....	60
Fill portion of a circle.....	60
Draw line.....	60
Draw text.....	60
Set text to fit specified width.....	60
Get text width.....	60
Get text height.....	60
Fill area with color.....	61
P16 image tools.....	61
Draws P16 image.....	61
Get P16 image width.....	61
Get P16 image height.....	61
Draws PPM.....	61
Touch API.....	62
Get if screen is touched.....	62
Clears touch event.....	62
Get touch y.....	62
Get touch x.....	62

Version 2024-09-14