TI-nspire™

Lua Scripting API Reference Guide

This reference guide applies to TI-Nspire™ software version 3.2. To obtain the latest version of the documentation, go to education.ti.com/nspire/scripting.

Important Information

Except as otherwise expressly stated in the License that accompanies a program, Texas Instruments makes no warranty, either express or implied, including but not limited to any implied warranties of merchantability and fitness for a particular purpose, regarding any programs or book materials and makes such materials available solely on an "as-is" basis. In no event shall Texas Instruments be liable to anyone for special, collateral, incidental, or consequential damages in connection with or arising out of the purchase or use of these materials, and the sole and exclusive liability of Texas Instruments, regardless of the form of action, shall not exceed the amount set forth in the license for the program. Moreover, Texas Instruments shall not be liable for any claim of any kind whatsoever against the use of these materials by any other party.

The TI-Nspire™ software uses Lua as scripting environment. For copyright and license information, see http://www.lua.org/license.html.

The TI-Nspire™ software uses Chipmunk Physics as simulation environment. For license information, see http://chipmunk-physics.net/release/ChipmunkLatest-Docs/.

Microsoft® Windows®, and Mac® OS X® are trademarks of their respective owners.

Contents

Chapter 1

Standard Libraries

The TI-Nspire™ software integrates most Lua standard libraries that come with the Lua distribution. See the (Lua 5.1 Reference Manual) for definitions of the standard functions.

1.1 Basic Library Functions

For further details, please follow this link to the "Basic Functions" section in the Lua 5.1 Reference Manual.

assert	**collectgarbage**	**error**	**_G**	**getfenv**	**getmetatable**
ipairs	**load**	**loadstring**	**next**	**pairs**	**pcall**
print	**rawequal**	**rawget**	**rawset**	**select**	**setfenv**
setmetatable	**tonumber**	**tostring**	**type**	**unpack**	**_VERSION**
xpcall					

Note about load() and loadstring()

Please be cautious with the use of **load** and **loadstring**. Lua source code loaded by the use of these functions is not supported in the TI-Nspire™ Editor. This source code cannot be debugged and error messages resulting from functions loaded using **load** and **loadstring** might cause confusing results.

1.1.1 Coroutine Sub-Library

For further details, please follow this link to the "Coroutine Manipulation" section in the Lua 5.1 Reference Manual. The following functions are defined inside the **coroutine** table. Heavy use of coroutines might be difficult to debug inside the TI-Nspire™ Editor.

create **resume** **running** **status** **wrap** **yield**

1.2 Module Library

The implementation of this module is very limited. Please consult the Module Library section for more details.

1.3 String Library

For further details, please follow this link to the "String Manipulation" section in the Lua 5.1 Reference Manual.

String routines lower and upper are not tailored to the current locale. The conversion of strings to **upper** and **lower** case letters operates only on the 26 letters of the Latin alphabet. This restriction also applies to the alphabetic matching patterns (%a, %l, %u, and %w) employed by the **find**, **gmatch**, and **match** functions.

byte **char** **dump** **find** **format** **gmatch** **gsub** **len**
lower **match** **rep** **reverse** **sub** **upper**

1.4 Table Library

For further details, please follow this link to the "Table Manipulation" section in the Lua 5.1 Reference Manual.

concat **insert** **maxn** **remove** **sort**

1.5 Math Library

For further details, please follow this link to the "Mathematical Functions" section in the Lua 5.1 Reference Manual. The following functions are defined inside the **math** table. Infinite and undefined results will convert to the appropriate TI-Nspire™ representations

and cooperate with the TI-Nspire™ math extensions. The reverse conversion of string representation (infinite and undefined) to numerical representation is not supported.

abs	**acos**	**asin**	**atan**	**atan2**	**ceil**	**cos**	**cosh**
deg	**exp**	**floor**	**fmod**	**frexp**	**huge**	**ldexp**	**log**
log10	**max**	**min**	**modf**	**pi**	**pow**	**rad**	**random**
randomseed	**sin**	**sinh**	**sqrt**	**tan**	**tanh**		

1.6 Unimplemented Libraries and Functions

The following standard Lua libraries are not available in the TI-Nspire™ software:

file **io** **os** **debug**

The following standard functions and standard table entries are not available in the TI-Nspire™ software:

dofile **loadfile** **module** **package.cpath** **package.loadlib** **package.path** **package.seeall**

Chapter 2

2D Editor

The Lua 2D editor bindings enable 2D rich text editors to be created and manipulated within the TI-Nspire™ product. 2D rich text editors are created using newRichText().

Info

Rich text editors embed formatting information in the text string to indicate the presence of a Math Box (Expression Box) or Chem Box. The functions getExpression, getExpressionSelection, and getText return the below described embedded formatting information if a Math Box or Chem Box is inside the editor.

"\0el {...}" - Denotes a Math Box (Expression Box). Evaluated Math Box expressions result in a pair of "\0el {...}" separated by a filled in '>'. See the Guidebook for a list of valid math expressions for the Math Box.

"\0chem {...}" - Denotes a Chem Box.

Note

Rich text editors embed other formatting information in the text string. This information may change in future releases, so using it is not recommended. It is delimited by "\1 ...\".

2.1 newRichText

D2Editor.newRichText()

Creates and returns a new 2D rich text editor.

Note

The program must resize the 2D editor before the text editor widget is painted the first time.

Default 2D Editor Setup

A new 2D rich text editor is created with the following defaults:

```
:move(0, 0)
:setBorder(0)
:setBorderColor(0x000000)
:setColorable(false)
:setDisable2DinRT(false)
:setFontSize(<default system size>)
:setMainFont(<default system font>)
:setReadOnly(false)
:setSelectable(true)
:setTextColor(0x000000)
:setVisible(true)
```

Introduced in platform.apiLevel = '1.0'

2.2 createChemBox

D2Editor:createChemBox()

Inserts a Chem Box in the current cursor position of the editor.

Returns the text editor object.

Introduced in platform.apiLevel = '2.0'

2.3 createMathBox

D2Editor:createMathBox()

Inserts a Math Box (Expression Box) in the current cursor position of the editor.

Returns the text editor object.

Introduced in platform.apiLevel = '2.0'

2.4 getExpression

```
D2Editor : getExpression()
```

Returns the contents of the text editor as a UTF-8 encoded string.

Introduced in platform.apiLevel = '2.0'

2.5 getExpressionSelection

```
D2Editor : getExpressionSelection()
```

Returns three values: the contents of the text editor as a UTF-8 encoded string, the cursor position as an integer, and the selection start as an integer.

Usage

Cursor and selection positions are the borders between characters, not the position of the characters. The following code snippets serve as examples.

```
str = 'This is a test string to see it working.'
d2e, error = D2Editor.newRichText()
result, error = d2e:setText(str, 16, 28)
str, pos, sel, error = d2e:getExpressionSelection()
```

The above code results in:
str = 'This is a test string to see it working.'
pos = 16 (right before the "s" in "string")
sel = 28 (between the two e's in "see")

```
str = 'This is a test string to see it working.'
d2e, error = D2Editor.newRichText()
result, error = d2e:setText(str, 28, 16)
str, pos, sel, error = d2e:getExpressionSelection()
```

The above code results in:
str = 'This is a test string to see it working.'
pos = 28 (between the two e's in "see")
sel = 16 (right before the 's' in "string")

Introduced in platform.apiLevel = '2.0'

2.6 getText

D2Editor: getText()

Returns the contents of the text editor as a UTF-8 encoded string.

Introduced in platform.apiLevel = '1.0'

2.7 hasFocus

D2Editor: hasFocus()

Returns true if the editor has focus; otherwise returns false.

Introduced in platform.apiLevel = '2.0'

2.8 isVisible

D2Editor: isVisible()

Returns true if the editor is visible; otherwise returns false.

Introduced in platform.apiLevel = '2.0'

2.9 move

D2Editor: move(x, y)

Sets the parent-relative location of the upper left corner of the text editor. Both **x** and **y** must be between -32767 and 32767.

Returns the text editor object.

Introduced in platform.apiLevel = '1.0'

2.10 registerFilter

D2Editor:registerFilter(handlerTable)

This routine registers a table of handler functions that can filter events before they are sent to the 2D editor widget, or unregisters if nil is passed.

Returns the text editor object.

The **handlerTable** is a table of event handler functions. Any event described in the section on Event Handling can be filtered by a function in the handler table.

In the example code below, if the user presses Tab in the text editor **ed**, the **tabKey** filter function moves the focus to text editor **ed2**. Events **charIn** and **arrowKey** simply report which key was pressed and then allow the event to pass on through to the text editor.

```
-- Create an editor
ed = D2Editor.newRichText()

-- Register filters for events
ed:registerFilter {
    tabKey =    function()
                    ed2:setFocus()
                    return true
                end,
    charIn =    function(ch)
                    print(ch)
                    return false
                end,
    arrowKey =  function(key)
                    print(key)
                    return false
                end,
}
```

Introduced in platform.apiLevel = '2.0'

2.11 resize

D2Editor: resize(width, height)

Changes the width and height of the text editor. Both **width** and **height** must be > 0 and < 32768.

Returns the text editor object.

Introduced in platform.apiLevel = '1.0'

2.12 setBorder

D2Editor: setBorder(thickness)

Sets the editor's border thickness. The thickness value must be between 0 and 10.

Returns the text editor object.

Introduced in platform.apiLevel = '2.0'

2.13 setBorderColor

D2Editor: setBorderColor(color)

Sets the editor's border color. The color value must be between 0 and 16777215 (0x000000 and 0xFFFFFF).

Returns the text editor object.

Introduced in platform.apiLevel = '2.0'

2.14 setColorable

D2Editor: setColorable(true or false)

Makes the expression colorable or uncolorable.

Returns the text editor object.

Introduced in platform.apiLevel = '2.0'

2.15 setDisable2DinRT

```
D2Editor:setDisable2DinRT(true or false)
```

Turns off 2D layout of math input to the text box.

Returns the text editor object.

Introduced in platform.apiLevel = '2.0'

f

2.16 setExpression

```
D2Editor:setExpression(text[, cursor[, selection
                                        [, full-precision]]])
```

Sets the text content of the text editor. The cursor position is set to 1 (beginning of text), -1 (end of text), or a value from 1 to the text length plus 1. Text can be selected by specifying a selection index that indicates the end of the selection. If the **selection** = -1, no text is selected. If the cursor < -1 or **selection** < -1, an error is returned. If unspecified, both the cursor and the selection start default to -1. If true, the final optional parameter, full-precision, indicates that all digits of the calculated results should be displayed. If false, full-precision indicates that calculated results should be rounded using the editor's precision setting.

Note

All backslashes sent to the editor must be doubled. This is in addition to the standard escape rule for special characters. As a result, the string required to get the editor to show **home\stuff\work** is "home\\\\stuff\\\\work".

Usage

Cursor and selection positions are the borders between characters, not the character positions. The following code snippet highlights the characters "string to se" and places the cursor before the 's' in "string".

```
str = 'This is a test string to see it working.'
d2e, error = D2Editor.newRichText()
result, error = d2e:setText(str, 16, 28)
```

The following code snippet highlights the characters "string to se" and places the cursor before the second 'e' in "see".

```
str = 'This is a test string to see it working.'
d2e, error = D2Editor.newRichText()
result, error = d2e:setText(str, 28, 16)
```

Returns the text editor object.

Introduced in platform.apiLevel = '2.0'

2.17 setFocus

```
D2Editor:setFocus(true or false)
```

Sets the user input focus on the editor if true (the default). This is usually called from the on.getFocus event handler.

Returns the text editor object.

Introduced in platform.apiLevel = '2.0'

2.18 setFontSize

```
D2Editor:setFontSize(size)
```

Sets the text font size in the editor. The point size is restricted on the TI-Nspire™ CX and older handheld devices. Choose one of these sizes: 7, 9, 10, 11, 12, or 24. Any font size supported by Windows® or Mac® OS X® can be used on the desktop software.

Returns the text editor object.

Introduced in platform.apiLevel = '2.0'

2.19 setMainFont

```
D2Editor:setMainFont(family, style)
```

Sets the main font family ("serif" or "sansserif") and style ("r", "b", "i", "bi").

Style	Description
r	Regular
b	Bold
i	Italic
bi	Bold and Italic

Returns the text editor object.

Note

This function affects only previously set text. Subsequent calls to setText, setExpression, or setFormattedExpression use the default font.

Introduced in platform.apiLevel = '2.0'

2.20 setReadOnly

```
D2Editor:setReadOnly(true or false)
```

Makes the text editor content modifiable (false) or unmodifiable (true) by the user. If a Boolean value is not specified, defaults to true.

Returns the text editor object.

Introduced in platform.apiLevel = '1.0'

2.21 setSelectable

```
D2Editor:setSelectable(true or false)
```

Makes the text editor content selectable (true) or unselectable (false) by the user. If a Boolean value is not specified, defaults to true.

Returns the text editor object.

Introduced in platform.apiLevel = '1.0'

2.22 setSizeChangeListener

D2Editor:setSizeChangeListener(function(editor, w, h))

Sets the callback function for when the editor contents exceed the current editor size, when the contents fit on fewer lines, or when the contents fit on a single line of smaller width. This function can then resize the editor appropriately. The callback function should be a void function. It will be passed into the following parameters:

Parameter	Description
editor	Editor in which the expression changed size.
w	Optimal widget width to fit the expression.
h	Optimal widget height to fit the expression.

Returns the text editor object.

Info

To remove the listener, call D2Editor:setSizeChangeListener(nil)

Introduced in platform.apiLevel = '2.0'

2.23 setText

D2Editor:setText(text[, cursor[, selection[, full-precision]]])

See **setExpression**() for details.

Returns the text editor object.

Introduced in platform.apiLevel = '1.0'

2.24 setTextChangeListener

D2Editor:setTextChangeListener(function(editor))

Sets the callback function for when the text expression changes. This function will be passed into the editor object. This allows for processing text input as it occurs.

Returns the text editor object.

Info

To remove the listener, call D2Editor:setTextChangeListener(nil)

Introduced in platform.apiLevel = '2.0'

2.25 setTextColor

```
D2Editor:setTextColor(color)
```

Sets the editor text color. The color value must be between 0 and 16777215 (0x000000 and 0xFFFFFF).

Returns the text editor object.

Introduced in platform.apiLevel = '2.0'

2.26 setVisible

```
D2Editor:setVisible(true or false)
```

Sets the visibility of the text editor.

Returns the text editor object.

Introduced in platform.apiLevel = '2.0'

2.27 setWordWrapWidth

```
D2Editor:setWordWrapWidth(width)
```

Sets the rich text editor word-wrapping width in pixels. Ignored if the editor is in 2D mode. To indicate widget width, sets to 0. To disable wrapping, sets to < 0. The width must be -32767 to 32767.

Note

When word wrapping is disabled, that is the width is < 0, and ellipses are added to cut words, the negative value of the width specifies the margin from the right of the widget before ellipses are used.

Returns the text editor object.

Introduced in platform.apiLevel = '2.0'

Chapter 3

Class Library

The class library implements basic object-oriented class definitions.

3.1 class

```
class([parent_class])
```

Returns a new class. If a parent class is specified, the new class inherits the methods of the parent class.

```
Widget = class()
function Widget:init() ... end

Button = class(Widget)
function Button:init() ... end
```

With these definitions, when the script calls Button(), a new Button is created. The Button:init() function is called to initialize the button, and the newly minted Button object is returned as the function result of the call.

Class Button in this example inherits all the methods and class variables defined in class Widget. Class Button can override any methods of its parent class.

Introduced in platform.apiLevel = '1.0'

Chapter 4

Clipboard Library

4.1 addText

```
clipboard.addText(string)
```

This routine adds the contents of **string** to the clipboard as plain text, MIME type "text/plain".

Introduced in platform.apiLevel = '1.0'

4.2 getText

```
clipboard.getText()
```

This routine returns the contents of the clipboard as a string of plain text. If the clipboard does not contain any text (MIME type "text/plain"), this routine returns nil.

Introduced in platform.apiLevel = '1.0'

Chapter 5

Cursor Library

This cursor library controls the appearance of the mouse pointer. The visibility of the cursor can only be controlled on a handheld.

A good practice is to request the expected cursor appearance within **on.activate()**. Calls on the cursor library are ignored while deactivated (after **on.deactivate()** is received).

5.1 set

```
cursor.set(cursorname)
```

Parameter **cursorname** is a string that contains the name of the cursor shape to use for the mouse pointer. It can be one of the following strings:

"default", "interrogation", "crosshair", "text", "pointer", "link select", "diag resize", "wait busy", "hollow pointer", "rotation", "pencil", "zoom box", "hide", "arrow", "zoom out", "dotted arrow", "clear", "animate", "excel plus", "mod label", "writing", "unavailable", "resize row", "resize column", "drag grab", "hand open", "hand closed", "hand pointer", "zoom in", "dilation", "translation", "show"

Introduced in platform.apiLevel = '1.0'

5.2 hide

```
cursor.hide()
```

This routine hides the mouse pointer on a handheld.

Note: Calls to this routine are ignored if not executed on a handheld.

Introduced in platform.apiLevel = '1.0'

5.3 show

```
cursor.show()
```

This routine makes the mouse pointer visible on a handheld.

Note: Calls to this routine are ignored if not executed on a handheld.

Introduced in platform.apiLevel = '1.0'

Chapter 6

Document Library

6.1 markChanged

```
document.markChanged()
```

This routine marks the current document as changed. The user is prompted to save the TI-Nspire™ document before closing.

Introduced in platform.apiLevel = '1.0'

Chapter 7

Event Handling

Script applications respond to external stimuli by implementing event handlers. All the event handlers are grouped in the "on" module.

Example

For example, the application script implements on.paint(gc) to be notified when it is time to redraw its window. on.paint is passed a graphics context that it can use to call drawing routines on its window.

```
function on.paint(gc)
    gc:drawLine(...)
    :
end
```

Set Script Event Sequence

The following sequence of events are generated when 'Set Script' is selected.

API Level 1.0	API Level 2.0	Comment
	on.construction()	on.construction() is new in 2.0
on.restore	on.restore	Only if opening a document and something was saved by on.save()
on.resize	on.resize	
...	...	Other calls depending on other active cards or scripts
on.activate	on.activate	When the script is active on the active card
...	...	Other calls depending on other applications or scripts
on.getFocus	on.getFocus	When the script receives user input focus
...	...	Other calls depending on other applications or scripts
on.create()		on.create() was obsoleted in 2.0
on.paint()	on.paint()	

7.1 activate

```
on.activate()
```

This routine is called when the script application is activated. The dimensions of the drawing window cannot be initialized at this point so it is not a good place to create and position graphical elements if they depend on the window size.

Introduced in platform.apiLevel = '1.0'

7.2 arrowDown

```
on.arrowDown()
```

This routine is called when the user presses the down arrow key.

Introduced in platform.apiLevel = '1.0'

7.3 arrowKey

```
on.arrowKey(key)
```

This routine is called when the user presses an arrow key. The **key** parameter may be "up," "down," "left," or "right." This routine is not called if the script implements a specific arrow key handler (on.arrowDown for instance) for the particular arrow key type.

Introduced in platform.apiLevel = '1.0'

7.4 arrowLeft

```
on.arrowLeft()
```

This routine is called when the user presses the left arrow key.

Introduced in platform.apiLevel = '1.0'

7.5 arrowRight

```
on.arrowRight()
```

This routine is called when the user presses the right arrow key.

Introduced in platform.apiLevel = '1.0'

7.6 arrowUp

```
on.arrowUp()
```

This routine is called when the user presses the up arrow key.

Introduced in platform.apiLevel = '1.0'

7.7 charIn

```
on.charIn(char)
```

This routine is called when the user types a letter, digit, or other characters. The parameter char is normally a one-byte string but because it can contain a UTF-8 encoded character,

it may be two or more bytes long. It may also contain the letters of a function name from one of the short-cut keys, such as "sin" from the trig menu.

Introduced in platform.apiLevel = '1.0'

7.8 backspaceKey

```
on.backspaceKey()
```

This routine is called when the user presses Backspace on the desktop keyboard or the Del key on the handheld device keypad.

Introduced in platform.apiLevel = '1.0'

7.9 backtabKey

```
on.backtabKey()
```

This routine is called when the user presses Shift + Tab.

Introduced in platform.apiLevel = '1.0'

7.10 clearKey

```
on.clearKey()
```

This routine is called when the user presses the Clear key on the handheld keypad.

Introduced in platform.apiLevel = '1.0'

7.11 construction

```
on.construction()
```

This function is guaranteed to fire first before any other event.

Introduced in platform.apiLevel = '2.0'

7.12 contextMenu

```
on.contextMenu()
```

This routine is called when the user presses the context Menu key.

Introduced in platform.apiLevel = '1.0'

7.13 copy

```
on.copy()
```

This routine is called when the user selects the Copy command either from a menu or by pressing Ctrl + C.

Note

Copy is enabled/disabled by toolpalette.enableCopy(enable).

Introduced in platform.apiLevel = '1.0'

7.14 create

```
on.create()
```

Tip

For scripts with platform.apiLevel $\geq$ '2.0' use on.construction() instead.

This routine is called after resize and before paint when the script application is created. The window size and graphics context are valid at this point. The on.paint event handler will be called soon after this routine finishes.

It is best to think of this function as an initialization method that fires once automatically.

Introduced in platform.apiLevel = '1.0'
Removed in platform.apiLevel = '2.0'

7.15 createMathBox

```
on.createMathBox()
```

This routine is called when the the user presses Ctrl + M or inserts a Math Box (Expression Box). The implementation for this callback should call the corresponding 2d editor to insert a math box if applicable.

Introduced in platform.apiLevel = '2.0'

7.16 cut

```
on.cut()
```

This routine is called when the user selects the Cut command either from a menu or by pressing Ctrl + X.

Note

Cut is enabled/disabled by toolpalette.enableCut(enable).

Introduced in platform.apiLevel = '1.0'

7.17 deactivate

```
on.deactivate()
```

This routine is called when the script is deactivated. This happens when the user moves the focus to another page or to another application on the same page.

Introduced in platform.apiLevel = '1.0'

7.18 deleteKey

```
on.deleteKey()
```

This routine is called when the user presses the Delete key on the desktop keyboard. This is not the Del key on the handheld keypad.

Introduced in platform.apiLevel = '1.0'

7.19 destroy

```
on.destroy()
```

This routine is called just before the script application is deleted. A script app is deleted when it is cut to the clipboard and when the document that contains it is closed.

Introduced in platform.apiLevel = '1.0'

7.20 enterKey

```
on.enterKey()
```

This routine is called when the user presses the Enter key.

Introduced in platform.apiLevel = '1.0'

7.21 escapeKey

```
on.escapeKey()
```

This routine is called when the user presses the Esc key.

Introduced in platform.apiLevel = '1.0'

7.22 getFocus

```
on.getFocus()
```

This routine is called when the script receives user input focus.

Introduced in platform.apiLevel = '2.0'

7.23 getSymbolList

```
on.getSymbolList()
```

This routine is called when the script app symbol list is being serialized to the clipboard. The script app returns a list of names of variables in the symbol table that it needs to copy with it to the clipboard. The TI-Nspire^{TM} system copies the names and values of the variables along with the script app. Then when the user pastes the script app in another problem, the system adds the companion variables to the problem symbol table.

Note

> on.getSymbolList() is called when a page containing a script app is copied, but not when a problem containing a script app is copied. This is because the entire symbol table is copied when the problem is copied.

For example, the following function indicates that it needs variable **f1** to be copied with the app to the clipboard. The value of **f1** will be added to the symbol table when it is pasted into another problem even in another TNS document.

```
function on.getSymbolList()
    return {"f1"}
end
```

Introduced in platform.apiLevel = '2.0'

7.24 grabDown

```
on.grabDown(x, y)
```

This routine is called in these situations:

- When the user presses and holds the Select key on a device
- When the user presses Ctrl + Select on a device
- When the user presses the middle mouse button over an active card on the desktop

x &y are always zero

The grabDown and grabUp events prevent the generation of a mouseUp event in all cases. They will be preceded by a mouseDown event when generated by pressing and holding the Select key on a device.

Introduced in platform.apiLevel = '1.0'

7.25 grabUp

```
on.grabUp(x, y)
```

This routine is called when the mouse button is released while grab is in effect.

x &y are always zero

Introduced in platform.apiLevel = '1.0'

7.26 help

```
on.help()
```

This routine is called when the user presses the Help key. On the desktop, the Help key is Ctrl + Shift + ?. On the handheld device, it is Ctrl + ?, the control key over the Trig button.

Introduced in platform.apiLevel = '1.0'

7.27 loseFocus

```
on.loseFocus()
```

This routine is called when the script loses user input focus.

Introduced in platform.apiLevel = '2.0'

7.28 mouseDown

```
on.mouseDown(x, y)
```

This routine is called when the user clicks the mouse. **x** and **y** are in window-relative pixel coordinates.

Note

This event will NOT be generated if the right mouse button is being held down.

Introduced in platform.apiLevel = '1.0'

7.29 mouseMove

```
on.mouseMove(x, y)
```

This routine is called when the user moves the mouse pointer. The mouse button does not have to be pressed to receive these events.

Introduced in platform.apiLevel = '1.0'

7.30 mouseUp

```
on.mouseUp(x, y)
```

This routine is called when the user releases the mouse button.

Note

This event will NOT be generated in the following cases:

- The preceding mouseDown event was blocked because the right mouse button was down already.
- The preceding mouseDown event was not handled.

Introduced in platform.apiLevel = '1.0'

7.31 paint

```
on.paint(gc)
```

This routine is called when the script application window needs to be painted. The gc graphics context is used in the script code to draw on the window.

Introduced in platform.apiLevel = '1.0'

7.32 paste

```
on.paste()
```

This routine is called when the user selects the Paste command either from a menu or by pressing Ctrl + V.

Note

Paste is enabled/disabled by toolpalette.enablePaste(enable).

Introduced in platform.apiLevel = '1.0'

7.33 resize

```
on.resize(width, height)
```

This routine is called when the script application window changes size. This is a good place to initialize (or reinitialize) graphical objects based on the window size.

Introduced in platform.apiLevel = '1.0'

7.34 restore

```
on.restore(state)
```

This routine is called when the script application is restored from its saved state in a document or when the app is pasted into a document. It is called only if the state was saved with the application when it was previously copied to the clipboard or saved in a document. See the on.save handler.

The parameter **state** is the table that the on.save event handler returned.

Warning

Functionality that is not available during initialization is also not usable within on.restore. Among the functions that cannot be called are math.eval and platform.isDeviceModeRendering.

Introduced in platform.apiLevel = '1.0'

7.35 returnKey

```
on.returnKey()
```

This routine is called when the user presses the Return key (?) on the handheld keypad.

Introduced in platform.apiLevel = '1.0'

7.36 rightMouseDown

```
on.rightMouseDown(x, y)
```

This routine is called when the user clicks the right mouse button. **x** and **y** are in window-relative pixel coordinates.

Note

Only available on the desktop version.

Mouse events are exclusive, which means that a rightMouseDown event cannot occur while the left mouse button is being held down and vice versa.

Introduced in platform.apiLevel = '1.0'

7.37 rightMouseUp

```
on.rightMouseUp(x, y)
```

This routine is called when the user releases the right mouse button.

Note

Only available on the desktop version.

This event will NOT be generated in the following cases:

- The preceding rightMouseDown event was blocked because the left mouse button was already down.
- The preceding rightMouseDown event was not handled.

Introduced in platform.apiLevel = '1.0'

7.38 save

```
on.save()
```

This routine is called when the script app is saved to the document or copied to the clipboard. The script should return a table of whatever data it needs to properly restore when the on.restore event handler is called.

Introduced in platform.apiLevel = '1.0'

7.39 tabKey

```
on.tabKey()
```

This routine is called when the user presses the Tab key.

Introduced in platform.apiLevel = '1.0'

7.40 timer

```
on.timer()
```

If the script application implements on.timer, the system calls this routine each time the timer ticks.

Introduced in platform.apiLevel = '1.0'

7.41 varChange

```
on.varChange(varlist)
```

This routine is called when a monitored variable is changed by another application. The **varlist** is a list of variable names whose values were changed. This handler must return a value to indicate if it accepts the new value(s) or vetoes the change.

Valid return values are:

Value	Brief Description	Comment
0	Success	The script application accepts the change.
-1	Veto range	The new value is unsatisfactory because it is outside the acceptable range, that is too low or too high.
-2	Veto type	The new value is unsatisfactory because its type cannot be used by the script application.
-3	Veto existence	Another application deleted the variable, and this application needs it.

Introduced in platform.apiLevel = '1.0'

Chapter 8

Graphics Library

A graphics context is a module that has a handle to the script's graphics output window and a library of graphics routines that are used to draw on the window. A graphics context is supplied to the script "on.paint" event handler each time the window needs to be redrawn.

The graphics context employs a pixel-based coordinate system with the origin in the upper left corner of the drawing window.

8.1 clipRect

```
gc:clipRect(op[, x, [y, [width, [height]]]])
```

Sets the clipping rectangle for subsequent graphics operations.

Parameter **op** takes one of the strings "set," "reset," "intersect," or "null."

Operation	Description
reset	Sets the clipping rectangle to include the entire window. The remaining parameters are ignored and can be left out.
set	Sets the clipping rectangle to the x, y coordinates with the specified width and height. Unspecified parameters default to the system window location and size.
intersect	Removed in platform.apilevel = '2.0'.
null	Sets the clipping rectangle to empty. All subsequent graphics commands are ignored.

Typically the "set" operation is called before drawing, such as for a text string. It is important to call the "reset" operation after drawing the last clipped graphic so that you do not leave a lingering clipping rectangle as a side effect.

Introduced in platform.apiLevel = '1.0'

8.2 drawArc

```
gc:drawArc(x, y, width, height, startAngle, arcAngle)
```

Draws an arc in the rectangle with upper left corner (x,y) and pixel width and height. Both the width and height must be ≥ 0. The arc is drawn beginning at startAngle degrees and continues for endAngle degrees. Zero degrees points to the right, and 90 degrees points up (standard mathematical practice but worth mentioning since the y axis is inverted).

To draw a circle, the width and height must be equal in length, and the start and end angles must be 0 and 360. If the width and height are different lengths, this routine draws an oval.

Introduced in platform.apiLevel = '1.0'

8.3 drawImage

```
gc:drawImage(image, x, y)
```

Draws an image at (x, y). The image must have been created by a previous call to image.new(...).

Introduced in platform.apiLevel = '1.0'

8.4 drawLine

```
gc:drawLine(x1, y1, x2, y2)
```

Draws a line from (x1,y1) to (x2,y2).

Introduced in platform.apiLevel = '1.0'

8.5 drawPolyLine

```
gc:drawPolyLine({x1, y1, x2, y2, ..., xn, yn})
```

Draws a series of lines connecting the (x, y) points. The polygon is not closed automatically. The first x-y coordinate pair must be repeated at the end of the array of points to draw a closed polygon.

Introduced in platform.apiLevel = '1.0'

8.6 drawRect

```
gc:drawRect(x, y, width, height)
```

Draws a rectangle at (x, y) with the given pixel width and height. Both width and height must be ≥ 0.

Introduced in platform.apiLevel = '1.0'

8.7 drawString

```
gc:drawString("text", x, y [,vertalignment])
```

Draws text on the window beginning at pixel location (x,y). Vertical alignment may be "baseline," "bottom," "middle," or "top." This aligns the text in the height of the characters' bounding rectangle. If the vertical alignment is unspecified, it defaults to "none."

Returns the x pixel position after the text.

Introduced in platform.apiLevel = '1.0'

8.8 fillArc

```
gc:fillArc(x, y, width, height, startAngle, endAngle)
```

Fills an arc with the preset color. Both width and height must be ≥ 0. See setColorRGB to set the fill color.

Introduced in platform.apiLevel = '1.0'

8.9 fillPolygon

```
gc:fillPolygon({x1, y1, x2, y2, ... xn, yn})
```

Fills a polygon with the preset color. The array of points bounds the polygon. To set the fill color, see setColorRGB.

Introduced in platform.apiLevel = '1.0'

8.10 fillRect

```
gc:fillRect(x, y, width, height)
```

Fills a rectangle with the preset color. Both the width and height must be ≥ 0. To set the fill color, see setColorRGB.

Introduced in platform.apiLevel = '1.0'

8.11 getStringHeight

```
gc:getStringHeight("text")
```

Returns the pixel height of the text. The pixel height is determined by the font setting previously set by a call to setFont.

Introduced in platform.apiLevel = '1.0'

8.12 getStringWidth

```
gc:getStringWidth("text")
```

Returns the pixel width of text. The pixel width is calculated using the font setting previously set by a call to setFont.

Introduced in platform.apiLevel = '1.0'

8.13 setColorRGB

```
gc:setColorRGB(red, green, blue)
gc:setColorRGB(0xRRGGBB) -- platform.level = '2.0' only
```

Sets the color for subsequent draw and fill routines. The red, green, and blue components of the color are values in the range of 0 to 255. Black is 0,0,0 and white is 255,255,255. Alternately, a single value can be passed in. The components of this single value are blue + 255 * (green + 255 * red).

Introduced in platform.apiLevel = '1.0'

Extended in platform.apiLevel = '2.0'

8.14 setFont

```
gc:setFont(family, style, size)
```

Sets the font for drawing text and measuring text size. Family may be "sansserif" or "serif". Style may be "r" for regular, "b" for bold, "i" for italic, or "bi" for bold italic.

The point size of the font is restricted on the TI-Nspire™ CX and older handheld devices. Choose one of these sizes: 7, 9, 10, 11, 12, or 24. Any font size supported by Windows® or Mac® OS X® can be used on the desktop software.

Returns the font family, style, and size previously in effect.

Introduced in platform.apiLevel = '1.0'

8.15 setPen

```
gc:setPen([thickness[, style]])
```

Sets the pen for drawing lines and borders. Thickness may be "thin," "medium," or "thick." If the thickness is not specified, it defaults to "thin." The style can be "smooth," "dotted," or "dashed." If the style is not specified, it defaults to "smooth."

Introduced in platform.apiLevel = '1.0'

Chapter 9

Image Library

An "image" object is a container for graphical images, typically small GUI objects such as buttons, arrowheads, and other such graphical adornments.

9.1 new

```
img = image.new(str)
```

This function returns a new image object from a string input. The string consists of the image header followed by the binary representation of the image pixels.

The header consists of 20 bytes of data arranged as presented in the following table. All fields are little endian integers.

Offset	Width (bytes)	Contents
0	4	Pixel width of image
4	4	Pixel height of image
8	1	Image alignment (0)
9	1	Flags (0)
10	2	Pad (0)
12	4	The number of bytes between successive raster lines
16	2	The number of bits per pixel (16)
18	2	Planes per bit (1)

The image pixel data immediately follows the header. Pixels are arranged in rows. Each pixel is a little endian 16-bit integer with five bits for each color red, green, and blue. The

top bit determines if the pixel is drawn. If it is zero (0), the pixel is not drawn. If it is one (1), the pixel is drawn in the RGB color of the remaining 15 bits.

0x8000 is black, 0x801F is blue, 0x83E0 is green, 0xFC00 is red, and 0xFFFF is white.

Introduced in platform.apiLevel = '1.0'

9.2 copy

```
cimage = image:copy(width, height)
```

Returns a copy of the input image scaled to fit the specified pixel width and height.

The width and height default to the size of the input image.

Introduced in platform.apiLevel = '1.0'

9.3 height

```
h = image:height()
```

Returns the pixel height of the image.

Introduced in platform.apiLevel = '1.0'

9.4 rotate

```
rimage = image:rotate(angle)
```

Returns a copy of the input image rotated counterclockwise by **angle** degrees.

Introduced in platform.apiLevel = '2.0'

9.5 width

```
w = image:width()
```

Returns the pixel width of the image.

Introduced in platform.apiLevel = '1.0'

Chapter 10

Locale Library

10.1 name

```
locale.name()
```

Returns the name of the current locale. The locale name is a two-letter language code. The language code may be followed by an underscore and two-letter country code.

Introduced in platform.apiLevel = '1.0'

Chapter 11

Math Library Extension

In addition to the functions that come with the standard Lua math library, there is an interface to the TI-Nspire™ math server that allows access to the advanced mathematical features of the TI-Nspire™ product.

Note

The TI-Nspire™ math server uses a number of unicode characters. For example, the math server uses Unicode character U+F02F, *i*, UTF-8 character "\239\128\175", for imaginary numbers and another special character for the exponent for a scientific notation, small capital letter "E".
(See http://en.wikipedia.org/wiki/UTF-8 for a description of how to convert unicode to UTF-8 and vice versa. See TI-Nspire™ Reference Guide for a list of unicode characters used in TI-Nspire™ software.

All results from the TI-Nspire™ math server are returned as full-precision expressions. To limit the precision of the result to the display digits, retrieve the current display digits via math.getEvalSettings() and apply the appropriate precision before displaying the value returned by the TI-Nspire™ math server.

11.1 eval

```
math.eval(math_expression)           -- platform.apiLevel = '2.0'
math.eval(math_expression [,exact]) -- platform.apiLevel = '1.0'
```

This function sends an expression or command to the TI-Nspire™ math server for evaluation. The input expression must be a string that the TI-Nspire™ math server can

interpret and evaluate.

The second parameter, **exact**, (platform.apiLevel = '1.0 only) is meaningful only with the Computer Algebra System. If true, it instructs the math server to calculate and return exact numerical results when it can. The default value of exact is false, in which case the math server attempts to calculate an approximate result.

Beginning with platform.apiLevel = '2.0', the evaluation is performed using the current document settings, except that all evaluations are performed at full precision in approximate mode. The current document settings can be overriden by **math.setEvalSettings**.

If the math server evaluates the expression successfully, it returns the results as a fundamental Lua data type. If the math server cannot evaluate the expression because of a syntax, simplification, or semantic error, **eval** returns two results: nil and an error number meaningful to the math server. (The error numbers are documented in the TI-Nspire™ Reference Guide - Error Codes and Messages for math.eval.) If the math server calculates a symbolic result, it cannot be represented as a fundamental Lua type, so **eval** returns nil and the string "incompatible data type."

Example

To evaluate **f1** for a given value in x, the parameter x must be converted to a string, and then any embedded "e" must be replaced with Unicode character U+F000.

```
local mx = tostring(x):gsub("e", string.uchar(0xF000))
local expr = "f1(" .. mx .. ")"
return math.eval(expr)
```

Note

Because math.eval always does calculations in approximate mode, things like Boolean logic and some conversions will throw an error:

Boolean logic:
r,e = math.eval('1 and 2') returns "Argument must be a Boolean expression or integer" error

Convert to base 10
r,e = math.eval("0@>Base10") returns "Domain Error"

math.evalStr works fine in such cases.

Warning

math.eval is not available during script initialization.

Introduced in platform.apiLevel = '1.0'
Revised to remove the optional argument **exact** and use current document settings, approximate mode, and full precision in **platform.apilevel = '2.0'**

11.2 evalStr

```
math.evalStr(math_expression)
```

This function sends an expression or command to the TI-Nspire™ math server for evaluation. The input expression must be a string that the TI-Nspire™ math server can interpret and evaluate. The evaluation is performed using the current document settings, which can be overriden by **math.setEvalSettings**. NOTE: All evaluations are performed at full precision regardless of the document settings or overrides.

If the math server evaluates the expression successfully, it returns the results as a string. The **evalStr** function returns no result if the math server does not return a calculated result. If the math server cannot evaluate the expression because of a syntax, simplification, or semantic error, **evalStr** returns two results: nil and an error number meaningful to the math server.

Examples

The evaluation of "10^19" in exact mode returns "1. 19". A closer look at the result string indicates that it contains "\049\046\239\128\128\49\57". "\239\128\128" is Unicode character U+F000, which is small capital letter "E".

```
result, error = math.evalStr('10^19')
t, u, v, w, x, y, z = string.byte(result, 1, 7)
print (result, #result, t, u, v, w, x, y, z)

->1.?19   7   49   46   239   128   128   49   57
```

The evaluation of "2-3" returns "-1". The result string will be encoded as "\226\136\146\49". "\226\136\146" is Unicode character U+2212, which is a minus sign.

```
result, error = math.evalStr('2-3')
v, w, x, y, z = string.byte(result, 1, 5)
print (result, #result, v, w, x, y, z)

->?1.   5   226   136   146   49   46
```

Introduced in platform.apiLevel = '2.0'

11.3 getEvalSettings

```
math.getEvalSettings()
```

Returns a table of tables with the document settings that are currently being used by **math.eval**. These settings are equivalent to the current document settings unless a call has been made to **setEvalSettings**.

Example

This example serves to demonstrate the structure of the table returned by **getEvalSettings**.

```
{
    {'Display Digits', 'Fixed12'},
    {'Angle Mode', 'Gradian'},
    {'Calculation Mode', 'Approximate'},
    {'Real or Complex Format', 'Polar'},
    {'Exponential Format', 'Engineering'},
    {'Vector Format', 'Cylindrical'},
    {'Base', 'Binary'},
    {'Unit System', 'Eng/US'}, }
}
```

Introduced in platform.apiLevel = '2.0'

11.4 setEvalSettings

```
math.setEvalSettings(settingStructure)
```

This function is used to override one or more of the current document settings for all subsequent math evaluations performed by **math.eval** and **math.evalStr**. It does not change the document context settings. The setting structure is a table of tables. Each inner table consists of the name of the document setting to override and the name of the value to use instead.

Example

Sample call to math.setEvalSettings()

```
settings = {
    {'Unit System', 'Eng/US'},
    {'Calculation Mode', 'Approximate'},
    {'Real or Complex Format', 'Polar'},
    {'Exponential Format', 'Engineering'}
}

math.setEvalSettings(settings)
```

For user convenience, **setEvalSettings** also accepts the ordinal number of the setting to override and the ordinal number of the value to use instead. The ordinal numbers to use correspond to the order of the settings and their values found at File > Settings > Document Settings.

Example

Sample call to math.setEvalSettings() using a table with ordinal numbers

```
settingsTable = {
    {2, 3},
    {4, 3},
    {6, 3},
    {8, 2}
}

math.setEvalSettings(settingsTable)
```

In fact, **setEvalSettings** accepts any combination of names and ordinal numbers. So the following is also valid.

Example

Sample call to math.setEvalSettings() using a table with combined names and numbers

```
settings = {
    {3, 'Exact'},
    {'Angle Mode', 2},
    {'Real or Complex Format', 'Polar'},
    {8, 2}
}

math.setEvalSettings(settings)
```

The function **math.setEvalSettings** may be called at any point in the script app. The modified document settings are used by **math.eval** for all subsequent calls within the script app (unless modified by a subsequent call to **setEvalSettings**).

Note

All results from the TI-Nspire™ math server are returned as full-precision expressions. If users want to limit the display digits, they must call math.getEvalSettings() and apply the appropriate precision before displaying the value returned by the TI-Nspire™ math server.

Introduced in platform.apiLevel = '2.0'

Chapter 12

Physics Library

This is an interface library to Chipmunk Physics version 5.3.4. For details about this library see http://chipmunk-physics.net/release/Chipmunk-5.x/Chipmunk-5.3.4-Docs/.

To use this library the physics module must be loaded: **"require ('physics')"**.

This library is introduced in platform.apiLevel = 2.0.

12.1 Miscellaneous routines

12.1.1 INFINITY

```
infinity = physics.misc.INFINITY()
```

Parameter	Type	Description
infinity	**out** number	Infinity

Returns a number representing infinity in the physics engine.

Introduced in platform.apiLevel = '2.0'

12.1.2 momentForBox

```
inertia = physics.misc.momentForBox(mass, width, height)
```

Parameter	Type	Description
mass	**in** number	The mass of the box
width	**in** number	The width of the box
height	**in** number	The height of the box
inertia	**out** number	The inertia of the box

This routine computes the moment of inertia for a solid box. This is a useful helper routine for computing the moment of inertia as an input to the physics.Body(...) constructor.

Introduced in platform.apiLevel = '2.0'

12.1.3 momentForCircle

```
inertia = physics.misc.momentForCircle(mass, innerRadius,
                                       outerRadius, offBody)
```

Parameter	Type	Description
mass	**in** number	The mass of the circle
innerRadius	**in** number	The inner radius of the circle
outerRadius	**in** number	The outer radius of the circle
offset	**in** physics.Vect	The offset of the circle from the center of gravity
inertia	**out** number	The inertia of the circle

This routine computes the moment of inertia for a circle. A solid circle has an inner radius of 0. This is a useful helper routine for computing the moment of inertia as an input to the physics.Body(...) constructor.

Introduced in platform.apiLevel = '2.0'

12.1.4 momentForPoly

```
inertia = physics.misc.momentForPoly(mass, vertices, offset)
```

Parameter	Type	Description
mass	**in** number	The mass of the polygon
vertices	**in** {physics.Vect}	A list of vertices defining the shape of the polygon
offset	**in** physics.Vect	The offset of the polygon from the center of gravity
inertia	**out** number	The inertia of the polygon

This routine computes the moment of inertia for a polygon. This is a useful helper routine for computing the moment of inertia as an input to the physics.Body(...) constructor.

Introduced in platform.apiLevel = '2.0'

12.1.5 momentForSegment

```
inertia = physics.misc.momentForSegment(mass, endPointA,
                                        endPointB)
```

Parameter	Type	Description
mass	**in** number	The mass of the segment
endPointA	**in** physics.Vect	The point defining one end of the segment
endPointB	**in** physics.Vect	The point defining the other end of the segment
inertia	**out** number	The inertia of the segment

This routine computes the moment of inertia for a segment. The end points can be in either world or local coordinates. This is a useful helper routine for computing the moment of inertia as an input to the physics.Body(...) constructor.

Introduced in platform.apiLevel = '2.0'

12.2 Vectors

A vector is a 2-dimensional object with x and y components. Its type is TI.cpVect.

12.2.1 Vect

```
vector = physics.Vect(x, y)
vector = physics.Vect(angle)
vector = physics.Vect(vect)
```

Parameter	Type	Description
x	**in** number	The **x** component of the vector
y	**in** number	The **y** component of the vector
angle	**in** number	An angle in radians
vect	**in** physics.Vect	A vector
vector	**out** physics.Vect	A vector

Creates a vector with initial x and y component values. The second form creates a unit vector pointing in direction angle. The third form creates a copy of the input vector.

Introduced in platform.apiLevel = '2.0'

12.2.2 add

```
sum = physics.Vect:add(vec)
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
vec	**in** physics.Vect	A vector to add to **self**
sum	**out** physics.Vect	The vector sum of **self** and **vec**

Returns the vector sum of **self** and **vec**.

The Vect class also implements the addition operator (+). Therefore vectors **v1** and **v2** can be added with the expression **v1** + **v2**.

Introduced in platform.apiLevel = '2.0'

12.2.3 clamp

```
clamped = physics.Vect:clamp(len)
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
len	**in** number	The maximum length of the vector
clamped	**out** physics.Vect	A new vector with a length no longer than **len**

Returns a copy of **self** clamped to length **len**.

Introduced in platform.apiLevel = '2.0'

12.2.4 cross

```
crossprod = physics.Vect:cross(vec)
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
vec	**in** physics.Vect	The vector to cross with **self**
zmag	**out** number	The z magnitude of the cross product of **self** and **vec**

Returns the z magnitude of the cross product of **self** and **vec**.

Introduced in platform.apiLevel = '2.0'

12.2.5 dist

```
dist = physics.Vect:dist(vec)
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
vec	**in** physics.Vect	The vector to which to find the distance from **self**
dist	**out** number	The distance from **self** to **vec**

Returns the distance between **self** and **vec**.

Introduced in platform.apiLevel = '2.0'

12.2.6 distsq

```
distsq = physics.Vect:distsq(vec)
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
vec	**in** physics.Vect	The vector to which to find the distance squared from **self**
distsq	**out** number	The distance squared from **self** to **vec**

Returns the distance squared between **self** and **vec**. For distance comparison this routine is faster than **physics.Vect:dist**.

Introduced in platform.apiLevel = '2.0'

12.2.7 dot

```
dotprod = physics.Vect:dot(vec)
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
vec	**in** physics.Vect	The other vector
dotprod	**out** number	The scalar dot product of **self** and **vec**

Returns the scalar dot product of **self** and **vec**.

Introduced in platform.apiLevel = '2.0'

12.2.8 eql

```
isequ = physics.Vect:eql(vec)
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
vec	**in** physics.Vect	The vector against which to compare with **self**
isequ	**out** boolean	True if the components of **self** equal the components of **vec**

Returns true if the x and y components of **self** equal those of **vec**. Take the usual precautions when comparing floating point numbers for equality.

The Vect class also implements the equal comparison operator (==). Therefore vectors **v1** and **v2** can be compared with the expression **v1** == **v2**.

Introduced in platform.apiLevel = '2.0'

12.2.9 length

```
len = physics.Vect:length()
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
len	**out** number	The length of vector **self**

Returns the magnitude of **self**.

Introduced in platform.apiLevel = '2.0'

12.2.10 lengthsq

```
lensq = physics.Vect:lengthsq()
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
lensq	**out** number	The length squared of vector **self**

Returns the length squared of **self**. This routine is faster than Vect:length() when you only need to compare lengths.

Introduced in platform.apiLevel = '2.0'

12.2.11 lerp

```
v = physics.Vect:lerp(vec, f)
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
vec	**in** physics.Vect	The other vector
f	**in** number	**f** is a fractional number from 0 to 1 representing the proportion of distance between **self** and **vec**
v	**out** physics.Vect	A vector interpolated between **self** and **vec**

Returns the linear interpolation between **self** and **vec** as a vector. **f** is the fraction of distance between **self** and **vec**.

Note

May not behave as expected for f larger than 1.0 or less than 0.

Introduced in platform.apiLevel = '2.0'

12.2.12 lerpconst

```
v = physics.Vect:lerpconst(vec, d)
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
vec	**in** physics.Vect	The other vector
d	**in** number	The distance from **self** to **vec** to interpolate a new vector
v	**out** physics.Vect	

Returns a vector interpolated from **self** towards **vec** with length **d**.

Note

May not behave as expected for d larger than 1.0 or less than 0.

Introduced in platform.apiLevel = '2.0'

12.2.13 mult

```
v = physics.Vect:mult(factor)
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
factor	**in** number	The value to multiply by **self**
v	**out** physics.Vect	The resulting scaled vector

Multiplies a vector by a factor.

Introduced in platform.apiLevel = '2.0'

12.2.14 near

```
isnear = physics.Vect:near(vec, distance)
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
vec	**in** physics.Vect	The value to multiply by **self**
distance	**in** number	The distance from **vec**
isnear	**out** boolean	True if **self** is within **distance** of **vec**

Determines if **self** is near another vector.

Introduced in platform.apiLevel = '2.0'

12.2.15 neg

```
v = physics.Vect:neg()
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
v	**out** physics.Vect	The resulting negated vector

Returns the negative of **self**.

The Vect class also implements the unary minus operator (- **self**).

Introduced in platform.apiLevel = '2.0'

12.2.16 normalize

```
normvec = physics.Vect:normalize()
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
normvec	**out** physics.Vect	The resulting normalized vector

Returns a normalized copy of **self**. The length of a normal vector is 1.

Introduced in platform.apiLevel = '2.0'

12.2.17 normalizeSafe

```
normvec = physics.Vect:normalizeSafe()
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
normvec	**out** physics.Vect	The resulting normalized vector

Returns a normalized copy of **self**. Protects against division by zero.

Introduced in platform.apiLevel = '2.0'

12.2.18 perp

```
perpvec = physics.Vect:perp()
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
perpvec	**out** physics.Vect	The resulting perpendicular vector

Returns a vector perpendicular to **self**. (90 degree rotation)

Introduced in platform.apiLevel = '2.0'

12.2.19 project

```
pvec = physics.Vect:project(vec)
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
vec	**in** physics.Vect	The other vector
pvec	**out** physics.Vect	The vector of **self** projected onto **vec**

Computes the projection of **self** onto another vector.

Introduced in platform.apiLevel = '2.0'

12.2.20 rotate

```
rvec = physics.Vect:rotate(vec)
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
vec	**in** physics.Vect	The other vector
rvec	**out** physics.Vect	The resulting rotated vector

Uses complex multiplication to rotate **self** by **vec**. Scaling will occur if **self** is not a unit vector.

Introduced in platform.apiLevel = '2.0'

12.2.21 rperp

```
perpvec = physics.Vect:rperp()
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
perpvec	**out** physics.Vect	The resulting perpendicular vector

Returns a vector perpendicular to **self**. (90 degree rotation)

Introduced in platform.apiLevel = '2.0'

12.2.22 setx

```
self = physics.Vect:setx(x)
```

Parameter	Type	Description
self	**in** physics.Vect	The vector to modify
x	**in** number	The new value of the **x** component of the vector
self	**out** physics.Vect	The input vector is returned as the output

Changes the value of the **x** component of **self**. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.2.23 sety

```
self = physics.Vect:sety(y)
```

Parameter	Type	Description
self	**in** physics.Vect	The vector to modify
y	**in** number	The new value of the **y** component of the vector
self	**out** physics.Vect	The input vector is returned as the output

Changes the value of the **y** component of **self**. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.2.24 slerp

```
v = physics.Vect:slerp(vec, f)
```

Parameter	Type	Description
self	**in** physics.Vect	A unit vector
vec	**in** physics.Vect	The other unit vector
f	**in** number	**f** is a fractional number from 0 to 1 representing the proportion of distance between **self** and **vec**
v	**out** physics.Vect	A vector interpolated between **self** and **vec**

Computes a spherical linear interpolation between unit vectors **self** and **vec**. **Info**

See [http://en.wikipedia.org/wiki/Slerp] for a discussion of the meaning, value, and usage of spherical linear interpolation.

```
local vect1 = physics.Vect(math.pi/3)      -- unit vector
local vect2 = physics.Vect(math.pi/2)      -- unit vector
local result = vect1:slerp(vect2, 0.55)
```

Note

This routine computes meaningful results only when the two inputs are unit vectors.

May not behave as expected for f larger than 1.0 or less than 0.

Introduced in platform.apiLevel = '2.0'

12.2.25 slerpconst

```
v = physics.Vect:slerpconst(vec, angle)
```

Parameter	Type	Description
self	**in** physics.Vect	A unit vector
vec	**in** physics.Vect	The other unit vector
angle	**in** number	The maximum angle between **self** and **vec** to interpolate a new vector
v	**out** physics.Vect	

Returns the spherical linear interpolation from **self** towards **vec** but by no more than **angle** in radians.

Info

See http://en.wikipedia.org/wiki/Slerp for a discussion of the meaning, value, and usage of spherical linear interpolation.

Note

This routine computes meaningful results only when the two inputs are unit vectors.

Introduced in platform.apiLevel = '2.0'

12.2.26 sub

```
diff = physics.Vect:sub(vec)
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
vec	**in** physics.Vect	A vector to subtract from **self**
diff	**out** physics.Vect	The vector difference between **self** and **vec**

Returns the vector difference of **self** and **vec**.

The Vect class also implements the subtraction operator (-). Therefore vector **v2** can be subtracted from **v1** with the expression **v1** - **v2**.

Introduced in platform.apiLevel = '2.0'

12.2.27 toangle

```
angle = physics.Vect:toangle()
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
angle	**out** number	The angle of **self**

Returns the angle in radians of **self**.

Introduced in platform.apiLevel = '2.0'

12.2.28 unrotate

```
uvec = physics.Vect:unrotate(vec)
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
vec	**in** physics.Vect	The other vector
uvec	**out** physics.Vect	The resulting unrotated vector

Inverse of physics.Vect:rotate(vec).

Introduced in platform.apiLevel = '2.0'

12.2.29 x

```
x = physics.Vect:x()
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
x	**out** number	The value of the **x** component of the vector

Returns the value of the **x** component of the input vector.

Introduced in platform.apiLevel = '2.0'

12.2.30 y

```
y = physics.Vect:y()
```

Parameter	Type	Description
self	**in** physics.Vect	The input vector
y	**out** number	The value of the **y** component of the vector

Returns the value of the **y** component of the input vector.

Introduced in platform.apiLevel = '2.0'

12.3 Bounding Boxes

A bounding box is a structure the contains the left, bottom, right, and top edges of a box. Its type is TI.cpBB.

12.3.1 BB

```
bb = physics.BB(l, b, r, t)
```

Parameter	Type	Description
l	**in** number	left
b	**in** number	bottom
r	**in** number	right
t	**in** number	top
bb	**out** physics.BB	A bounding box with boundaries left, bottom, right, and top

Returns a new bounding box with the given initial edges.

Introduced in platform.apiLevel = '2.0'

12.3.2 b

```
bottom = physics.BB:b()
```

Parameter	Type	Description
self	**in** physics.BB	The input bounding box
bottom	**out** number	The bottom edge of the bounding box

Returns the bottom edge of the bounding box.

Introduced in platform.apiLevel = '2.0'

12.3.3 clampVect

```
cvec = physics.BB:clampVect(vec)
```

Parameter	Type	Description
self	**in** physics.BB	The input bounding box
vec	**in** physics.Vect	A vector
cvec	**out** physics.Vect	A vector clamped to the bounding box

Returns a copy of **vec** clamped to the bounding box.

Introduced in platform.apiLevel = '2.0'

12.3.4 containsBB

```
bool = physics.BB:containsBB(other)
```

Parameter	Type	Description
self	**in** physics.BB	The input bounding box
other	**in** physics.BB	The other bounding box
bool	**out** boolean	True if **self** completely contains the **other** bounding box

Determines if a bouding box contains another bounding box.

Introduced in platform.apiLevel = '2.0'

12.3.5 containsVect

```
bool = physics.BB:containsVect(vec)
```

Parameter	Type	Description
self	**in** physics.BB	The input bounding box
vec	**in** physics.Vect	A vector
bool	**out** boolean	True if **self** contains vector **vec**

Determines if a bounding box contains a vector.

Introduced in platform.apiLevel = '2.0'

12.3.6 expand

```
bb = phyics.BB:expand(vec)
```

Parameter	Type	Description
self	**in** physics.BB	The input bounding box
vec	**in** physics.Vect	A vector
bb	**out** physics.BB	The bounding box **self** expanded to include vector **vec**

Returns the bounding box that contains both **self** and **vec**.

Introduced in platform.apiLevel = '2.0'

12.3.7 intersects

```
bool = physics.BB:intersects(other)
```

Parameter	Type	Description
self	**in** physics.BB	The input bounding box
other	**in** physics.BB	The other bounding box
bool	**out** boolean	True if **self** intersects the **other** bounding box

Determines if two bounding boxes intersect.

Introduced in platform.apiLevel = '2.0'

12.3.8 l

```
left = physics.BB:l()
```

Parameter	Type	Description
self	**in** physics.BB	The input bounding box
left	**out** number	The left edge of the bounding box

Returns the left edge of the bounding box.

Introduced in platform.apiLevel = '2.0'

12.3.9 merge

```
bb = physics.BB:merge(other)
```

Parameter	Type	Description
self	**in** physics.BB	The input bounding box
other	**in** physics.BB	The other bounding box
bb	**out** physics.BB	The bounding box that contains both **self** and the **other** bounding box

Returns the bounding box that contains both **self** and the **other** bounding box.

Introduced in platform.apiLevel = '2.0'

12.3.10 setb

```
self = physics.BB:setb(bottom)
```

Parameter	Type	Description
self	**in** physics.BB	The input bounding box
bottom	**in** number	The new value for the bottom edge of the bounding box
self	**out** physics.BB	The input bounding box is returned as the output

Sets the bottom edge of the bounding box to a new **value**. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.3.11 r

```
right = physics.BB:r()
```

Parameter	Type	Description
self	**in** physics.BB	The input bounding box
right	**out** number	The right edge of the bounding box

Returns the right edge of the bounding box.

Introduced in platform.apiLevel = '2.0'

12.3.12 setl

```
self = physics.BB:setl(left)
```

Parameter	Type	Description
self	**in** physics.BB	The input bounding box
left	**in** number	The new value for the left edge of the bounding box
self	**out** physics.BB	The input bounding box is returned as the output

Sets the left edge of the bounding box to a new **value**. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.3.13 setr

```
self = physics.BB:setr(right)
```

Parameter	Type	Description
self	**in** physics.BB	The input bounding box
right	**in** number	The new value for the right edge of the bounding box
self	**out** physics.BB	The input bounding box is returned as the output

Sets the right edge of the bounding box to a new **value**. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.3.14 sett

```
self = physics.BB:sett(top)
```

Parameter	Type	Description
self	**in** physics.BB	The input bounding box
top	**in** number	The new value for the top edge of the bounding box
self	**out** physics.BB	The input bounding box is returned as the output

Sets the top edge of the bounding box to a new **value**. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.3.15 t

```
top = physics.BB:t()
```

Parameter	Type	Description
self	**in** physics.BB	The input bounding box
top	**out** number	The top edge of the bounding box

Returns the top edge of the bounding box.

Introduced in platform.apiLevel = '2.0'

12.3.16 wrapVect

```
wvec = physics.BB:wrapVect(vec)
```

Parameter	Type	Description
self	**in** physics.BB	The input bounding box
vec	**in** physics.Vect	A vector
wvec	**out** physics.Vect	A vector wrapped to the bounding box

Returns a copy of **vec** wrapped to the bounding box.

Introduced in platform.apiLevel = '2.0'

12.4 Bodies

A body holds the physical properties (mass, position, rotation, velocity, etc.) of an object. It does not have a shape until you attach one (or more) to it. Its type is TI.cpBody.

12.4.1 Body

```
body = physics.Body(mass, inertia)
```

Parameter	Type	Description
mass	**in** number	Mass of the body
inertia	**in** number	The inertia of the body
body	**out** physics.Body	A new Body with the supplied mass and inertia

Returns a new Body with the given mass and moment of inertia.

Use the provided helper functions to compute the moment of inertia.

Introduced in platform.apiLevel = '2.0'

12.4.2 activate

```
self = physics.Body:activate()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
self	**out** physics.Body	The input Body is returned as the output

Activates a sleeping body.

Info

See http://chipmunk-physics.net/release/Chipmunk-5.x/Chipmunk-5.3.4-Docs/ for an explanation of this routine.

Introduced in platform.apiLevel = '2.0'

12.4.3 angle

```
angle = physics.Body:angle()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
angle	**out** number	The angle of the Body in radians

Returns the angle in radians of the orientation of the body.

Introduced in platform.apiLevel = '2.0'

12.4.4 angVel

```
avel = physics.Body:angVel()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
avel	**out** number	The angular velocity of the Body in radians per unit time

Returns the angular velocity of the body in radians per unit time.

Introduced in platform.apiLevel = '2.0'

12.4.5 applyForce

```
self = physics.Body:applyForce(forceVect, rOffset)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
forceVect	**in** physics.Vect	A force vector
rOffset	**in** physics.Vect	Vector offset of the force relative to the Body
self	**out** physics.Body	The input Body is returned as the output

Apply force vector on **self** at a relative offset from the center of gravity.

Introducted in platform.apiLevel = '2.0'

12.4.6 applyImpulse

```
self = physics.Body:applyImpulse(impulseVect, rOffset)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
impulseVect	**in** physics.Vect	Impulse force on the Body
rOffset	**in** physics.Vect	Vector offset of the force relative to the Body
self	**out** physics.Body	The input Body is returned as the output

Apply the impulse vector to **self** at a relative offset from the center of gravity.

Introducted in platform.apiLevel = '2.0'

12.4.7 data

```
obj = physics.Body:data()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
obj	**out** Lua object	An object previously set on the Body by the programmer

Returns the contents of the programmer data field of the Body.

Introducted in platform.apiLevel = '2.0'

12.4.8 force

```
fvec = physics.Body:force()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
fvec	**out** physics.Vect	The force vector on the Body

Returns the force vector on the body.

Introduced in platform.apiLevel = '2.0'

12.4.9 isRogue

```
bool = physics.Body:isRogue()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
bool	**out** boolean	True if the Body is a rogue Body

Returns true if the Body is a rogue Body, never having been added to the simulation Space.

Info

> See http://chipmunk-physics.net/release/Chipmunk-5.x/Chipmunk-5.3.4-Docs/ for an explanation of rogue bodies.

Introduced in platform.apiLevel = '2.0'

12.4.10 isSleeping

```
bool = physics.Body:isSleeping()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
bool	**out** boolean	True if the Body is sleeping

Returns true if the body is sleeping.

Info

See http://chipmunk-physics.net/release/Chipmunk-5.x/Chipmunk-5.3.4-Docs/ for an explanation of sleeping bodies.

Introduced in platform.apiLevel = '2.0'

12.4.11 local2World

```
wvec = physics.Body:local2World(lvec)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
lvec	**in** physics.Vect	A vector relative to the position of the Body
wvec	**out** physics.Vect	A vector in world coordinates

Converts **lvec** from body-relative coordinates to world coordinates. Returns the converted vector.

Introduced in platform.apiLevel = '2.0'

12.4.12 kineticEnergy

```
ke = physics.Body:kineticEnergy()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
ke	**out** number	The total kinetic energy of the Body

Returns the kinetic energy of the body.

Introduced in platform.apiLevel = '2.0'

12.4.13 mass

```
m = physics.Body:mass()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
m	**out** number	The mass of the Body

Returns the mass of the body.

Introduced in platform.apiLevel = '2.0'

12.4.14 moment

```
m = physics.Body:moment()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
m	**out** number	The moment of inertia of the Body

Returns the moment of inertia of the body.

Introduced in platform.apiLevel = '2.0'

12.4.15 pos

```
p = physics.Body:pos()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
p	**out** physics.Vect	The position of the Body

Returns the vector position of the body.

Introduced in platform.apiLevel = '2.0'

12.4.16 resetForces

```
self = physics.Body:resetForces()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
self	**out** physics.Body	The input Body is returned as the output

Zero both the force and torque accumulated on **self**.

Introducted in platform.apiLevel = '2.0'

12.4.17 rot

```
rvec = physics.Body:rot()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
rvec	**out** physics.Vect	The unit vector orientation of the Body

Returns the vector orientation of the body. This is a unit vector cached from the last calculated angle of the Body.

Introduced in platform.apiLevel = '2.0'

12.4.18 setAngle

```
self = physics.Body:setAngle(angle)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
angle	**in** number	The angle of rotation in radians of the Body
self	**out** physics.Body	The input Body is returned as the output

Updates the angle of rotation in radians of the body.

Returns the Body.

Introduced in platform.apiLevel = '2.0'

12.4.19 setAngVel

```
self = physics.Body:setAngVel(vel)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
vel	**in** number	The angular velocity in radians per unit time of the Body
self	**out** physics.Body	The input Body is returned as the output

Updates the angular velocity of the body. The angular velocity is in radians per unit time.

Returns the Body.

Introduced in platform.apiLevel = '2.0'

12.4.20 setData

```
self = physics.Body:setData(value)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
value	**in** object	A programmer-supplied Lua object
self	**out** physics.Body	The input Body is returned as the output

Sets the programmer data field of the Body. The programmer can store any Lua object in this field. This is a handy place to store a reference to a simulation object.

Returns the Body.

Introduced in platform.apiLevel = '2.0'

12.4.21 setForce

```
self = physics.Body:setForce(vector)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
vector	**in** physics.Vect	The vector of force on the Body
self	**out** physics.Body	The input Body is returned as the output

Updates the force vector on the body.

Returns the Body.

Introduced in platform.apiLevel = '2.0'

12.4.22 setMass

```
self = physics.Body:setMass(mass)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
mass	**in** number	The mass of the Body
self	**out** physics.Body	The input Body is returned as the output

Updates the mass of the body.

Returns the Body.

Introduced in platform.apiLevel = '2.0'

12.4.23 setMoment

```
self = physics.Body:setMoment(moment)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
moment	**in** number	The moment of inertia of the Body
self	**out** physics.Body	The input Body is returned as the output

Updates the moment of inertia of the body.

Use the provided helper functions to compute the moment of inertia.

Returns the Body.

Introduced in platform.apiLevel = '2.0'

12.4.24 setPos

```
self = physics.Body:setPos(vector)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
vector	**in** physics.Vect	The position of the Body
self	**out** physics.Body	The input Body is returned as the output

Updates the position of the body.

Returns the Body.

Introduced in platform.apiLevel = '2.0'

12.4.25 setPositionFunc

```
self = physics.Body:setPositionFunc(func)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
func	**in** function(body, dt)	A callback function that updates the position of the Body on each time step
self	**out** physics.Body	The input Body is returned as the output

Sets the position function of the body. The position function must be a function that accepts a Body and a time step value and at some point calls body:updatePosition to update the position of the body.

Returns the Body.

Introduced in platform.apiLevel = '2.0'

12.4.26 setTorque

```
self = physics.Body:setTorque(torque)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
torque	**in** number	The torque of the Body
self	**out** physics.Body	The input Body is returned as the output

Updates the torque on the body. Torque is a numeric magnitude.

Returns the Body.

Introduced in platform.apiLevel = '2.0'

12.4.27 setVel

```
self = physics.Body:setVel(vector)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
vector	**in** physics.Vect	The velocity vector of the Body
self	**out** physics.Body	The input Body is returned as the output

Updates the velocity of the body.

Returns the Body.

Introduced in platform.apiLevel = '2.0'

12.4.28 setVelocityFunc

```
self = physics.Body:setVelocityFunc(func)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
func	**in** function(body, grav, damping, dt)	A callback function that updates the velocity of the Body on each time step
self	**out** physics.Body	The input Body is returned as the output

Sets the velocity function of the body. The velocity function must be a function that accepts a Body, a gravity vector, a numeric damping factor, and a time step value. The function should call body:updateVelocity to adjust the velocity of the body.

Returns the Body.

Example

```
function sampleVelocityFunc(body, gravity, damping, dt)
    local pos    = body:pos()
    local sqdist = pos:lengthsq()
    local g      = pos:mult(-GravityStrength /
                 (sqdist * math.sqrt(sqdist)))
    body:updateVelocity(g, damping, dt)
end

body:setVelocityFunc(sampleVelocityFunc)
```

Introduced in platform.apiLevel = '2.0'

12.4.29 setVLimit

```
self = physics.Body:setVLimit(limit)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
limit	**in** number	The maximum speed of the Body
self	**out** physics.Body	The input Body is returned as the output

Sets the limit for the maximum speed of the body.

Returns the Body.

Introduced in platform.apiLevel = '2.0'

12.4.30 setWLimit

```
self = physics.Body:setWLimit(limit)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
limit	**in** number	The maximum angular velocity of the Body
self	**out** physics.Body	The input Body is returned as the output

Updates the limit of the angular velocity of the body. Angular velocity is in radians per unit time.

Returns the Body.

Introduced in platform.apiLevel = '2.0'

12.4.31 sleep

```
self = physics.Body:sleep()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
self	**out** physics.Body	The input Body is returned as the output

Puts the body to sleep.

Info

See http://chipmunk-physics.net/release/Chipmunk-5.x/Chipmunk-5.3.4-Docs/ for an explanation of sleeping bodies.

Note

The body must be added to a Space before it can be put to sleep.

Calling this function within a query or callback is not allowed.

Introduced in platform.apiLevel = '2.0'

12.4.32 sleepWithGroup

```
self = physics.Body:sleepWithGroup( [group] )
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
group	**in** physics.Body	A sleeping body. If this parameter is not supplied, a new group is created
self	**out** physics.Body	The input Body is returned as the output

Puts the Body to sleep and adds it to a group of other sleeping bodies.

Info

See http://chipmunk-physics.net/release/Chipmunk-5.x/Chipmunk-5.3.4-Docs/ for an explanation of this routine.

Note

The body must be added to a Space before it can be put to sleep.

Calling this function within a query or callback is not allowed.

This routine will raise an exception if **group** is not sleeping.

Introduced in platform.apiLevel = '2.0'

12.4.33 torque

```
t = physics.Body:torque()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
torque	**out** number	The torque on the Body

Returns the torque on the body.

Introduced in platform.apiLevel = '2.0'

12.4.34 updatePosition

```
physics.Body:updatePosition(dt)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
dt	**in** number	The time interval in seconds

Updates the position of the body using Euler integration.

Info

See http://chipmunk-physics.net/release/Chipmunk-5.x/Chipmunk-5.3.4-Docs/ for an explanation of this routine.

Introduced in platform.apiLevel = '2.0'

12.4.35 updateVelocity

```
physics.Body:updateVelocity(grav, damp, dt)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
grav	**in** physics.Vect	The force of gravity
damp	**in** physics.Vect	The damping factor
dt	**in** physics.Vect	The time interval in seconds

Updates the velocity of the body using Euler integration.

Info

See http://chipmunk-physics.net/release/Chipmunk-5.x/Chipmunk-5.3.4-Docs/ for an explanation of this routine.

Introduced in platform.apiLevel = '2.0'

12.4.36 vel

```
vvel = physics.Body:vel()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
vvel	**out** physics.Vect	The velocity of the Body

Returns the vector velocity of the body.

Introduced in platform.apiLevel = '2.0'

12.4.37 vLimit

```
vmax = physics.Body:vLimit()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
vmax	**out** number	The maximum speed of the Body

Returns the speed limit of the body.

Introduced in platform.apiLevel = '2.0'

12.4.38 wLimit

```
wmax = physics.Body:wLimit()
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
wmax	**out** number	The maximum angular velocity of the Body in radians per unit time

Returns the angular velocity limit of the body. The angular velocity is in radians per unit time.

Introduced in platform.apiLevel = '2.0'

12.4.39 world2Local

```
lvec = physics.Body:world2Local(wvec)
```

Parameter	Type	Description
self	**in** physics.Body	The input Body
wvec	**in** physics.Vect	A vector in world coordinates
lvec	**out** physics.Vect	A vector relative to the position of the Body

Converts **wvec** from world coordinates to body-relative coordinates. Returns the converted vector.

Introduced in platform.apiLevel = '2.0'

12.5 Shapes

Shapes contain the surface properties of an object such as how much friction or elasticity it has. All collision shapes implement the following accessor routines.

12.5.1 BB

```
bb = physics.Shape:BB()
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
bb	**in** physics.BB	Bounding box of the Shape

Returns the bounding box of the shape.

Introduced in platform.apiLevel = '2.0'

12.5.2 body

```
body = physics.Shape:body()
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
body	**out** physics.Body	The Body associated with the Shape

Returns the body attached to the shape. If the shape is static, then it will return nil.

Introduced in platform.apiLevel = '2.0'

12.5.3 collisionType

```
coll = physics.Shape:collisionType()
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
coll	**out** number	The programmer-assigned integer collision type

Returns the integer collision type of the Shape.

Introduced in platform.apiLevel = '2.0'

12.5.4 data

```
obj = physics.Shape:data()
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
obj	**out** Lua object	The programmer-assigned data object assigned to this Shape

Returns the contents of the programmer data field of the Shape.

Introducted in platform.apiLevel = '2.0'

12.5.5 friction

```
f = physics.Shape:friction()
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
f	**out** number	The coefficient of friction for this Shape

Returns the friction coefficient of the shape.

Introduced in platform.apiLevel = '2.0'

12.5.6 group

```
g = physics.Shape:group()
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
g	**out** number	The assigned group number

Returns the group number of the shape.

Note

The group number is converted to a positive whole number when stored.

Introduced in platform.apiLevel = '2.0'

12.5.7 layers

```
layers = physics.Shape:layers()
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
layers	**out** number	A bitmap of the layers this shape occupies

Returns the bitmap of layers the shape occupies.

Introduced in platform.apiLevel = '2.0'

12.5.8 rawBB

```
bb = physics.Shape:rawBB()
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
bb	**out** physics.BB	The bounding box of the Shape

Returns the bounding box of the shape. Only valid after a call to physics.Shape:BB() or physics.Space:step().

Introduced in platform.apiLevel = '2.0'

12.5.9 restitution

```
r = physics.Shape:restitution()
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
r	**out** number	The restitution of the Shape

Returns the restitution (or elasticity) of the shape.

Introduced in platform.apiLevel = '2.0'

12.5.10 sensor

```
s = physics.Shape:sensor()
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
s	**out** boolean	True if the Shape is a sensor

Returns true if the shape is a sensor.

Introduced in platform.apiLevel = '2.0'

12.5.11 setCollisionType

```
self = physics.Shape:setCollisionType(collisionType)
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
collisionType	**in** number	Programmer-defined type of collision
self	**out** physics.Shape	The input Shape is returned as the output

Assigns a collision type (an integer value of your choosing) to the shape. It is used to determine which handler to call when a collision occurs. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.5.12 setData

```
self = physics.Shape:setData(obj)
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
obj	**in** Lua object	An object defined by the programmer
self	**out** physics.Shape	The input Shape is returned as the output

Sets the programmer data field of the Shape. The programmer can store any Lua object in this field. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.5.13 setFriction

```
self = physics.Shape:setFriction(f)
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
f	**in** number	Coefficient of friction for the surface of the Shape
self	**out** physics.Shape	The input Shape is returned as the output

Sets the friction coefficient for the shape. Returns **self**.

Note

May not behave as expected for f larger than 1.0 or less than 0.

Introduced in platform.apiLevel = '2.0'

12.5.14 setGroup

```
self = physics.Shape:setGroup(group)
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
group	**in** number	Group number
self	**out** physics.Shape	The input Shape is returned as the output

Sets the group (a number defined by the programmer) of the shape. Shapes in the same group do not generate collisions. Returns **self**.

Note

The group number is converted to a positive whole number when stored.

Introduced in platform.apiLevel = '2.0'

12.5.15 setLayers

```
self = physics.Shape:setLayers(layers)
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
layers	**in** number	A bitmap of integer layer numbers. This implementation permits 32 layers
self	**out** physics.Shape	The input Shape is returned as the output

Sets the layers that the shape inhabits. Shapes only collide if they are in the same layer. **layers** is an integer bitmap of all the layers that the shape occupies. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.5.16 setRestitution

```
self = physics.Shape:setRestitution(r)
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
r	**in** number	The new value for the shape's restitution
self	**out** physics.Shape	The input Shape is returned as the output

Sets the restitution (or elasticity) of the shape. A value of 0.0 gives no bounce and a value of 1.0 gives a perfect bounce. Returns **self**.

Note

May not behave as expected for r larger than 1.0 or less than 0.

12.5.17 setSensor

```
self = physics.Shape:setSensor(bool)
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
bool	**in** boolean	True if the shape is a sensor
self	**out** physics.Shape	The input Shape is returned as the output

Determines if the shape is a sensor (true) or not (false). Sensors call collision handlers but do not generate collisions. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.5.18 setSurfaceV

```
self = physics.Shape:setSurfaceV(vel)
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
vel	**in** physics.Vect	The new vector for the surface velocity
self	**out** physics.Shape	The input Shape is returned as the output

Sets the surface velocity of the shape. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.5.19 surfaceV

```
sv = physics.Shape:surfaceV()
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
sv	**out** physics.Vect	The surface velocity of the Shape

Returns the surface velocity vector of the shape.

Introduced in platform.apiLevel = '2.0'

12.6 Circle Shapes

A CircleShape is a subclass of Shape. Its type is TI.cpCircleShape.

12.6.1 CircleShape

```
cs = physics.CircleShape(body, radius, offset)
```

Parameter	Type	Description
body	**in** physics.Body	A Body or nil
radius	**in** number	The radius of the circle
offset	**in** physics.Vect	The offset of the circle from the Body
cs	**out** physics.CircleShape	A new CircleShape

Returns a new CircleShape with the given body, radius, and offset vector from the body's center of gravity in body-local coordinates. Specify nil for the body to use the space's static body.

Introduced in platform.apiLevel = '2.0'

12.6.2 offset

```
ovec = physics.CircleShape:offset()
```

Parameter	Type	Description
self	**in** physics.CircleShape	The input CircleShape
ovec	**out** physics.Vect	The offset of the shape from the Body

Returns the offset vector of the shape from the body's center of gravity.

Introduced in platform.apiLevel = '2.0'

12.6.3 radius

```
r = physics.CircleShape:radius()
```

Parameter	Type	Description
self	**in** physics.CircleShape	The input CircleShape
r	**out** number	The radius of the shape

Returns the radius of the shape.

Introduced in platform.apiLevel = '2.0'

12.7 Polygon Shapes

Polygon shapes are bounded by a set of line segments. The enclosed area of the polygon must be convex and the vertices must be defined in counterclockwise order. Poygon shapes are of type TI.cpPolyShape.

12.7.1 PolyShape

```
ps = physics.PolyShape(body, vertices, offset)
```

Parameter	Type	Description
body	**in** physics.Body	A Body or nil
vertices	**in** {physics.Vect}	The list of vertices that define the boundaries of the polygon defined in counterclockwise order
offset	**in** physics.Vect	The offset of the PolyShape from the Body
ps	**out** physics.PolyShape	A new PolyShape

Returns a new PolyShape with the given body, table of vertices, and offset from the body's center of gravity. Specify nil for the body to use the space's static body.

Introduced in platform.apiLevel = '2.0'

12.7.2 numVerts

```
nv = physics.PolyShape:numVerts()
```

Parameter	Type	Description
self	**in** physics.PolyShape	The input PolyShape
nv	**out** number	The number of vertices in the PolyShape

Returns the number of vertices in the table of polygon vertices.

Introduced in platform.apiLevel = '2.0'

12.7.3 points

```
points = physics.PolyShape:points()
```

Parameter	Type	Description
self	**in** physics.PolyShape	The input PolyShape
points	**out** {physics.Vect}	A table of vertices that define the boundary of the polygon. The vertices are translated to the polygon's current world coordinates

Returns a copy of the table of vertices defining the bounds of the polygon. The vertices are translated to the polygon's current world coordinates. **Note**

> When a PolyShape has not been added to a Space, it has no world coordinates. In this case, each vertex returned by physics.PolyShape:points() will have x and y equal to 0.

Introduced in platform.apiLevel = '2.0'

12.7.4 vert

```
v = physics.PolyShape:vert(n)
```

Parameter	Type	Description
self	**in** physics.PolyShape	The input PolyShape
v	**out** physics.Vect	The nth vertex of the polygon. The coordinates of the vector are relative to the shape's Body

Returns vertex number **n** of the table of vertices defining the bounds of the polygon. If the shape is static then the vertex values are in world coordinates, otherwise the vertex coordiates are relative to the shape's body. Returns nil if **n** is less than 1 or greater than the number of vertices in the polygon.

Introduced in platform.apiLevel = '2.0'

12.8 Segment Shapes

A segment shape is defined by two end points and a radius. Its type is TI.cpSegmentShape.

12.8.1 SegmentShape

```
ss = physics.SegmentShape(body, a, b, radius)
```

Parameter	Type	Description
body	**in** physics.Body	A Body or nil
a	**in** physics.Vect	The first end point of the segment. The end point is in coordinates relative to the Body
b	**in** physics.Vect	The second end point of the segment relative to the Body
radius	**in** number	The distance of the border of the segment from the line between the end points of the segment
ss	**out** physics.SegmentShape	A new SegmentShape

Returns a new SegmentShape with end point vectors **a** and **b**. **radius** defines the thickness of the segment.

Introduced in platform.apiLevel = '2.0'

12.8.2 a

```
avec = physics.SegmentShape:a()
```

Parameter	Type	Description
self	**in** physics.SegmentShape	The input SegmentShape
avec	**out** physics.Vect	The first end point of the segment

Returns the **a** vector defining one of the end points of the segment.

Introduced in platform.apiLevel = '2.0'

12.8.3 b

```
bvec = physics.SegmentShape:b()
```

Parameter	Type	Description
self	**in** physics.SegmentShape	The input SegmentShape
bvec	**out** physics.Vect	The second end point of the segment

Returns the **b** vector defining one of the end points of the segment.

Introduced in platform.apiLevel = '2.0'

12.8.4 normal

```
nvec = physics.SegmentShape:normal()
```

Parameter	Type	Description
self	**in** physics.SegmentShape	The input SegmentShape
nvec	**out** physics.Vect	The unit normal vector of the segment

Returns the computed unit normal vector to the segment.

Introduced in platform.apiLevel = '2.0'

12.8.5 radius

```
r = physics.SegmentShape:radius()
```

Parameter	Type	Description
self	**in** physics.SegmentShape	The input SegmentShape
r	**out** number	The radius of the segment

Returns the radius of the segment.

Introduced in platform.apiLevel = '2.0'

12.9 Spaces

A physics Space is the basic unit of simulation.

12.9.1 Space

```
s = physics.Space()
```

Parameter	Type	Description
s	**out** physics.Space	A new simulation Space

Returns a new physics simulation Space.

Introduced in platform.apiLevel = '2.0'

12.9.2 addBody

```
self = physics.Space:addBody(body)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
body	**in** physics.Body	Adds the Body to the simulation Space
self	**out** physics.Space	The input Space is returned as the output

Adds a Body to the Space. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.9.3 addConstraint

```
self = physics.Space:addConstraint(constraint)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
constraint	**in** physics.Constraint	Adds a Constraint to the simulation Space
self	**out** physics.Space	The input Space is returned as the output

Adds a Constraint to the Space. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.9.4 addCollisionHandler

```
self = physics.Space:addCollisionHandler(collisionTypeA,
                                         collisionTypeB, callbacksTable)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
collisionTypeA	**in** number	Type of first collision
collisionTypeB	**in** number	Type of second collision
callbacksTable	**in** table of functions	A table of functions to call during collision detection and handling
self	**out** physics.Space	The input Space is returned as the output

Registers a table of callback functions to handle collisions between shapes of collisionTypeA and shapes of collisionTypeB. The **callbacksTable** is a table of the form:

```
{
  begin     = function(arbiter, space, callbacksTable) ... end,
  preSolve  = function(arbiter, space, callbacksTable) ... end,
  postSolve = function(arbiter, space, callbacksTable) ... end,
  separate  = function(arbiter, space, callbacksTable) ... end
}
```

If the **begin** handler or **preSolve** handler return false, further collision calculations are bypassed. If they return true, the collision processing proceeds as normal.

It is not necessary to provide handlers for all callback table entries. Default handling will be provided for unspecified handlers.

Returns **self**.

Info

See http://chipmunk-physics.net/release/Chipmunk-5.x/Chipmunk-5.3.4-Docs/ for an explanation of collision processing and collision handler callbacks.

One important point to note is that these callback handlers must not add or remove Bodies, Shapes, or Constraints from the Space.

See the post-step callback functions for the right way to remove (or add) objects as the result of a collision.

Introduced in platform.apiLevel = '2.0'

12.9.5 addPostStepCallback

```
self = physics.Space:addPostStepCallback(body|shape|constraint,
                                         function(space, object)
                                         ... end )
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
body or shape or constraint	**in** physics.Body or physics.Shape or physics.Constraint	A simulation object that will receive attention after the simulation step
function	**in** function(space, object)	The callback function to run against the simulation object at the end of the simulation step
self	**out** physics.Space	The input Space is returned as the output

Adds a callback function to be called when the current step is finished. One callback may be registered per Body, Shape, or Constraint. Only the first callback for a given object is registered. Any attempt to register another callback for the same object is ignored.

Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.9.6 addShape

```
self = physics.Space:addShape(shape)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
shape	**in** physics.Shape	Adds the Shape to the simulation Space
self	**out** physics.Space	The input Space is returned as the output

Adds a Shape to the Space. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.9.7 addStaticShape

```
self = physics.Space:addStaticShape(staticShape)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
staticShape	**in** physics.Shape	Adds the static Shape to the simulation Space
self	**out** physics.Space	The input Space is returned as the output

Adds a static Shape to the Space. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.9.8 damping

```
d = physics.Space:damping()
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
d	**out** number	The amount of damping of the simulation Space

Introduced in platform.apiLevel = '2.0'

12.9.9 data

```
obj = physics.Space:data()
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
obj	**out** Lua object	The programmer specified object associated with the Space
self	**out** physics.Space	The input Space is returned as the output

Introduced in platform.apiLevel = '2.0'

12.9.10 elasticIterations

```
iters = physics.Space:elasticIterations()
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
iters	**out** number	The number of iterations to use in the impulse solver to solve elastic collisions

Introduced in platform.apiLevel = '2.0'

12.9.11 gravity

```
grav = physics.Space:gravity()
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
grav	**out** physics.Vect	The gravity force vector applied to all Bodies in the simulation Space.

Introduced in platform.apiLevel = '2.0'

12.9.12 idleSpeedThreshold

```
speed = physics.Space:idleSpeedThreshold()
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
speed	**out** number	Threshold speed

Introduced in platform.apiLevel = '2.0'

12.9.13 iterations

```
iters = physics.Space:iterations()
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
iters	**out** number	The number of iterations the solver takes to update one step of the simulation

Introduced in platform.apiLevel = '2.0'

12.9.14 rehashShape

```
self = physics.Space:rehashShape(shape)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
shape	**in** shape	The shape to rehash
self	**out** physics.Space	The input Space is returned as the output

Update an individual static shape that has moved.

Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.9.15 rehashStatic

```
self = physics.Space:rehashStatic()
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
self	**out** physics.Space	The input Space is returned as the output

Rehashes the shapes in the static spatial hash. You must call this if you move any static shapes or Chipmunk won't update their collision detection data.

Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.9.16 removeBody

```
self = physics.Space:removeBody(body)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
body	**in** physics.Body	A Body to remove from the simulation Space
self	**out** physics.Space	The input Space is returned as the output

Removes a Body from the Space. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.9.17 removeConstraint

```
self = physics.Space:removeConstraint(constraint)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
constraint	**in** physics.Constraint	A Constraint to remove from the simulation Space
self	**out** physics.Space	The input Space is returned as the output

Removes a Constraint from the Space. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.9.18 removeShape

```
self = physics.Space:removeShape(shape)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
shape	**in** physics.Shape	A Shape to remove from the simulation Space
self	**out** physics.Space	The input Space is returned as the output

Removes a Shape from the Space. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.9.19 removeStaticShape

```
physics.Space:removeStaticShape(staticShape)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
staticShape	**in** physics.Shape	A static Shape to remove from the simulation Space
self	**out** physics.Space	The input Space is returned as the output

Removes a static Shape from the Space. Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.9.20 resizeActiveHash

```
self = physics.Space:resizeActiveHash(dim, count)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
dim	**in** number	The length of one side of a hash cell. Default is 100
count	**in** number	The number of cells in the hash table. Default is 1000
self	**out** physics.Space	The input Space is returned as the output

The spatial hash of active shapes can be tuned to improve collision detection. **dim** establishes the size of a hash cell (default 100), and **count** sets the number of hash cells (default 1000). **dim** should approximate the side length of a typical Shape. A good rule of thumb is to set **count** to about ten times the number of Shapes in the space.

Introduced in platform.apiLevel = '2.0'

12.9.21 resizeStaticHash

```
self = physics.Space:resizeStaticHash(dim, count)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
dim	**in** number	The length of one side of a hash cell. Default is 100
count	**in** number	The number of cells in the hash table. Default is 1000
self	**out** physics.Space	The input Space is returned as the output

This routine configures the spatial hash of static Shapes. Configure this similarly to resizeActiveHash but for static Shapes.

Introduced in platform.apiLevel = '2.0'

12.9.22 setDamping

Damping drains speed from bodies in the simulation. A value of 0.9 means that each body will lose 10

```
self = physics.Space:setDamping(d)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
d	**in** number	The new amount of damping for the simulation Space
self	**out** physics.Space	The input Space is returned as the output

Amount of viscous damping to apply to the Space.

Note

May not behave as expected for d larger than 1.0 or less than 0.

Introduced in platform.apiLevel = '2.0'

12.9.23 setData

```
self = physics.Space:setData(obj)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
obj	**in** Lua object	A programmer specified object
self	**out** physics.Space	The input Space is returned as the output

The programmer can store any Lua object in this field.

Introduced in platform.apiLevel = '2.0'

12.9.24 setElasticIterations

```
self = physics.Space:setElasticIterations(iters)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
iters	**in** number	The number of iterations to use in the impulse solver to solve elastic collisions. Defaults to 0
self	**out** physics.Space	The input Space is returned as the output

Introduced in platform.apiLevel = '2.0'

12.9.25 setGravity

```
self = physics.Space:setGravity(grav)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
grav	**in** physics.Vect	The gravity force vector applied to all Bodies in the simulation Space. Defaults to physics.Vect(0, 0)
self	**out** physics.Space	The input Space is returned as the output

Global gravity applied to the Space. Can be overridden on a per body basis by writing custom integration functions.

Introduced in platform.apiLevel = '2.0'

12.9.26 setIdleSpeedThreshold

```
self = physics.Space:setIdleSpeedThreshold(speed)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
speed	**in** number	Threshold speed
self	**out** physics.Space	The input Space is returned as the output

The idleSpeedThreshold is the speed below which a body is considered to be idle. This value is used to determine when a body can be put to sleep.

Introduced in platform.apiLevel = '2.0'

12.9.27 setIterations

```
self = physics.Space:setIterations(iters)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
iters	**in** number	Number of iterations to refine the accuracy of the solver. Default is 10
self	**out** physics.Space	The input Space is returned as the output

This value allows the programmer to control the accuracy of the solver. Default is 10.

Introduced in platform.apiLevel = '2.0'

12.9.28 setSleepTimeThreshold

```
self = physics.Space:setSleepTimeThreshold(sleep)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
sleep	**in** number	The amount of time (seconds) below which time if a Shape has not moved, it is put to sleep
self	**out** physics.Space	The input Space is returned as the output

Sleep time threshold is used to calculate when a Body can be put to sleep.

Introduced in platform.apiLevel = '2.0'

12.9.29 sleepTimeThreshold

```
sleep = physics.Space:sleepTimeThreshold()
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
sleep	**out** number	The threshold time used to determine when a Shape can be put to sleep

Introduced in platform.apiLevel = '2.0'

12.9.30 step

```
self = physics.Space:step(dt)
```

Parameter	Type	Description
self	**in** physics.Space	The input simulation Space
dt	**in** number	The length of time (seconds) of one step of the simulation
self	**out** physics.Space	The input Space is returned as the output

Updates the Space for the given time step **dt**. A fixed time step is recommended and increases the efficiency of the contact persistence, requiring an order of magnitude fewer iterations and lower CPU usage.

Returns **self**.

Introduced in platform.apiLevel = '2.0'

12.10 Constraints

All Constraints share common accessors.

Accessors	Type	Description
bodyA	physics.Body	The first Body that the Constraint acts on
bodyB	physics.Body	The second Body that the Constaint acts on
setBiasCoef, biasCoef	number	The fraction of error corrected each step of the simulation. Defaults to 0.1. May not behave as expected for numbers larger than 1.0 or less than 0.
setData, data	Lua object	A programmer-defined object
impulse	number	Calculated impulse applied by the Constraint in the last simulation step. To convert this to the magnitude of the force, divide by the time step passed to physics.Space:step()
setMaxBias, maxBias	number	Maximum speed the Constraint can apply error correction. Defaults to INFINITY
setMaxForce, maxForce	number	Magnitude of maximum force the Constraint can use to act on the two Bodies. Defaults to INFINITY

12.10.1 Damped Rotary Spring

```
spring = physics.DampedRotarySpring(a, b, restAngle,
                                    stiffness, damping)
```

Parameter	Type	Description
a	**in** physics.Body	First Body
b	**in** physics.Body	Second Body
restAngle	**in** number	Relative angle in radians that the Bodies want to maintain
stiffness	**in** number	The spring constant
damping	**in** number	How soft to make the damping of the spring
spring	**out** physics.DampedRotarySpring	New DampedRotarySpring

Like a damped spring, but works in an angular fashion. **restAngle** is the relative angle in radians that the Bodies want to have, **stiffness** and **damping** work basically the same as on a damped spring.

Accessors	Type
setRestAngle, restAngle	number
setStiffness, stiffness	number
setDamping, damping	number

Introduced in platform.apiLevel = '2.0'

12.10.2 Damped Spring

```
spring = physics.DampedSpring(a, b, anchr1, anchr2, restLength,
                              stiffness, damping)
```

Parameter	Type	Description
a	**in** physics.Body	First Body
b	**in** physics.Body	Second Body
anchr1	**in** physics.Vect	Anchor point to first Body
anchr2	**in** physics.Vect	Anchor point to second Body
restLength	**in** number	The distance the spring want to maintain between its Bodies
stiffness	**in** number	The spring constant
damping	**in** number	How soft to make the damping of the spring
spring	**out** physics.DampedSpring	New DampedSpring

Defined much like a SlideJoint. **restLength** is the distance the spring wants to be, **stiffness** is the spring constant, and **damping** is how soft to make the damping of

the spring.

Accessors	Type
setAnchr1, anchr1	physics.Vect
setAnchr2, anchr2	physics.Vect
setRestLength, restLength	number
setStiffness, stiffness	number
setDamping, damping	number

Introduced in platform.apiLevel = '2.0'

12.10.3 Gear Joint

```
joint = physics.GearJoint(a, b, phase, ratio)
```

Parameter	Type	Description
a	**in** physics.Body	First Body
b	**in** physics.Body	Second Body
phase	**in** number	The initial angular offset in radians of the two Bodies
ratio	**in** number	Ratio of velocities between the two Bodies
joint	**out** physics.GearJoint	New GearJoint

Keeps the angular velocity ratio of a pair of Bodies constant. **ratio** is always measured in absolute terms. **phase** is the initial angular offset of the two bodies.

Accessors	Type
setPhase, phase	number
setRatio, ratio	number

Introduced in platform.apiLevel = '2.0'

12.10.4 Groove Joint

```
joint = physics.GrooveJoint(a, b, grooveA, grooveB, anchr2)
```

Parameter	Type	Description
a	**in** physics.Body	First Body
b	**in** physics.Body	Second Body
grooveA	**in** physics.Vect	One end point of the groove
grooveB	**in** physics.Vect	The other end point of the groove
anchr2	**in** physics.Vect	The pivot point of Body b
joint	**out** physics.GlooveJoint	New GlooveJoint

The groove goes from **grooveA** to **grooveB** on Body **a**, and the pivot is attached to **anchr2** on Body **b**. All coordinates are body local.

Accessors	Type
setAnchr2, anchr2	physics.Vect
setGrooveA, grooveA	physics.Vect
setGrooveB, grooveB	physics.Vect
grooveN	physics.Vect

Introduced in platform.apiLevel = '2.0'

12.10.5 Pin Joint

```
joint = physics.PinJoint(a, b, anchr1, anchr2)
```

Parameter	Type	Description
a	**in** physics.Body	First Body
b	**in** physics.Body	Second Body
anchr1	**in** physics.Vect	The anchor point on Body a
anchr2	**in** physics.Vect	The anchor point on Body b
joint	**out** physics.PinJoint	New PinJoint

a and **b** are the two bodies to connect, and **anchr1** and **anchr2** are the anchor points on those bodies. The distance between the two anchor points is measured when the joint is created. If you want to set a specific distance, use the setter function to override it.

Accessors	Type
setAnchr1, anchr1	physics.Vect
setAnchr2, anchr2	physics.Vect
setDist, dist	number

Introduced in platform.apiLevel = '2.0'

12.10.6 Pivot Joint

```
joint = physics.PivotJoint(a, b, pivot)
joint = physics.PivotJoint(a, b, anchr1, anchr2)
```

Parameter	Type	Description
a	**in** physics.Body	First Body
b	**in** physics.Body	Second Body
pivot	**in** physics.Vect	Point of pivot between the two Bodies
anchr1	**in** physics.Vect	The anchor point on Body a
anchr2	**in** physics.Vect	The anchor point on Body b
joint	**out** physics.PivotJoint	New PivotJoint

a and **b** are the two bodies to connect, and **pivot** is the point in world coordinates of the pivot. Because the pivot location is given in world coordinates, you must have the bodies moved into the correct positions already. Alternatively you can specify the joint based on a pair of anchor points, but make sure you have the bodies in the right place as the joint will fix itself as soon as you start simulating the Space.

Accessors	Type
setAnchr1, anchr1	physics.Vect
setAnchr2, anchr2	physics.Vect

Introduced in platform.apiLevel = '2.0'

12.10.7 Ratchet Joint

```
joint = physics.RatchetJoint(a, b, phase, ratchet)
```

Parameter	Type	Description
a	**in** physics.Body	First Body
b	**in** physics.Body	Second Body
phase	**in** number	Initial offset in radians
ratchet	**in** number	The distance in radians between clicks of the ratchet
joint	**out** physics.RatchetJoint	New RatchetJoint

Works like a socket wrench. **ratchet** is the distance between clicks, **phase** is the initial offset to use when deciding where the ratchet angles are.

Accessors	Type
setAngle, angle	number
setPhase, phase	number
setRatchet, ratchet	number

Introduced in platform.apiLevel = '2.0'

12.10.8 Rotary Limit Joint

```
joint = physics.RotaryLimitJoint(a, b, min, max)
```

Parameter	Type	Description
a	**in** physics.Body	First Body
b	**in** physics.Body	Second Body
min	**in** number	The minimum angular distance in radians
max	**in** number	The maximum angular distance in radians
joint	**out** physics.RotaryLimitJoint	New RotaryLimitJoint

Constrains the relative rotations of two bodies. **min** and **max** are the angular limits in radians. It is implemented so that it is possible for the range to be greater than a full revolution.

Accessors	Type
setMin, min	number
setMax, max	number

Introduced in platform.apiLevel = '2.0'

12.10.9 Simple Motor

```
motor = physics.SimpleMotor(a, b, rate)
```

Parameter	Type	Description
a	**in** physics.Body	First Body
b	**in** physics.Body	Second Body
rate	**in** number	The relative angular velocity
motor	**out** physics.SimpleMotor	New SimpleMotor

Keeps the relative angular velocity of a pair of bodies constant. **rate** is the desired relative angular velocity.

Accessors	Type
setRate, rate	number

Introduced in platform.apiLevel = '2.0'

12.10.10 Slide Joints

```
joint = physics.SlideJoint(a, b, anchr1, anchr2, min, max)
```

Parameter	Type	Description
a	**in** physics.Body	First Body
b	**in** physics.Body	Second Body
anchr1	**in** physics.Vect	The anchor point on Body a
anchr2	**in** physics.Vect	The anchor point on Body b
min	**in** number	Minimum distance between Bodies
max	**in** number	Maximum distance between Bodies
joint	**out** physics.SlideJoint	New SlideJoint

a and **b** are the two bodies to connect, **anchr1** and **anchr2** are the anchor points on those bodies, and **min** and **max** define the allowed distances of the anchor points.

Accessors	Type
setAnchr1, anchr1	physics.Vect
setAnchr2, anchr2	physics.Vect
setMin, min	number
setMax, max	number

Introduced in platform.apiLevel = '2.0'

12.11 Arbiters and Collision Pairs

The Arbiter class encapsulates information about each pair of collisions.

12.11.1

```
count = #physics.Arbiter
```

Returns the number of contact points in this Arbiter.

Introduced in platform.apiLevel = '2.0'

12.11.2 a

```
shape = physics.Arbiter:a()
```

Parameter	Type	Description
self	**in** physics.Arbiter	The input Arbiter
shape	**out** physics.Shape	The first Shape in the collision pair

Returns Shape **a** (the first shape) in a collision pair.

Introduced in platform.apiLevel = '2.0'

12.11.3 b

```
shape = physics.Arbiter:b()
```

Parameter	Type	Description
self	**in** physics.Arbiter	The input Arbiter
shape	**out** physics.Shape	The second Shape in the collision pair

Returns Shape **b** (the second shape) in a collision pair.

Introduced in platform.apiLevel = '2.0'

12.11.4 bodies

```
bodyA, bodyB = physics.Arbiter:bodies()
```

Parameter	Type	Description
self	**in** physics.Arbiter	The input Arbiter
bodyA	**out** physics.Body	The first Body in the collision pair
bodyB	**out** physics.Body	The second Body in the collision pair

Returns bodyA and bodyB in the collision pair.

Introduced in platform.apiLevel = '2.0'

12.11.5 depth

```
d = physics.Arbiter:depth(i)
```

Parameter	Type	Description
self	**in** physics.Arbiter	The input Arbiter
i	**in** number	A contact point number
d	**out** number	The penetration depth of the ith contact point

Returns the penetration depth of the ith contact or nil if **i** is out of range of the number of contact points.

Introduced in platform.apiLevel = '2.0'

12.11.6 elasticity

```
e = physics.Arbiter:elasticity()
```

Parameter	Type	Description
self	**in** physics.Arbiter	The input Arbiter
e	**out** number	The calculated elasticity of the collision

Returns the calculated elasticity of this collision pair.

Introduced in platform.apiLevel = '2.0'

12.11.7 friction

```
f = physics.Arbiter:friction()
```

Parameter	Type	Description
self	**in** physics.Arbiter	The input Arbiter
f	**out** number	The calculated friction of the collision

Returns the calculated friction of this collision pair.

Introduced in platform.apiLevel = '2.0'

12.11.8 impulse

```
ivec = physics.Arbiter:impulse([friction])
```

Parameter	Type	Description
self	**in** physics.Arbiter	The input Arbiter
friction	**in** boolean	If true, the calculated friction is included in the calculation
ivec	**out** physics.Vect	The vector impulse applied to resolve the collision

Returns the vector impulse that was applied this step to resolve the collision. If **friction** is true (default false), then the calculated friction is taken into account.

Introduced in platform.apiLevel = '2.0'

12.11.9 isFirstContact

```
bool = physics.Arbiter:isFirstContact()
```

Parameter	Type	Description
self	**in** physics.Arbiter	The input Arbiter
bool	**out** boolean	True if this is the first step that the Shapes touched

Returns true if this is the first step that the Shapes touched. This information only persists until a step when the shapes are no longer touching. Once they are no longer touching this flag is reset.

Introduced in platform.apiLevel = '2.0'

12.11.10 normal

```
nvec = physics.Arbiter:normal(i)
```

Parameter	Type	Description
self	**in** physics.Arbiter	The input Arbiter
i	**in** number	A contact point number
nvec	**out** physics.Vect	Vector normal to the ith contact point

Returns the collision normal vector for the ith contact point. Returns nil if **i** is out of the range of the number of contact points.

Introduced in platform.apiLevel = '2.0'

12.11.11 point

```
pvec = physics.Arbiter:point(i)
```

Parameter	Type	Description
self	**in** physics.Arbiter	The input Arbiter
i	**in** number	A contact point number
pvec	**out** physics.Vect	The position of the ith contact point

Returns the position of the ith contact point. Returns nil if **i** is out of the range of the number of contact points.

Introduced in platform.apiLevel = '2.0'

12.11.12 setElasticity

```
self = physics.Arbiter:setElasticity(e)
```

Parameter	Type	Description
self	**in** physics.Arbiter	The input Arbiter
e	**in** number	Elasticity of the collision
self	**out** physics.Arbier	The input Arbiter is returned as the output

Overrides the calculated elasticity of the collision.

Note

May not behave as expected for e larger than 1.0 or less than 0.

Introduced in platform.apiLevel = '2.0'

12.11.13 setFriction

```
self = physics.Arbiter:setFriction(friction)
```

Parameter	Type	Description
self	**in** physics.Arbiter	The input Arbiter
f	**in** number	Friction in the collision
self	**out** physics.Arbier	The input Arbiter is returned as the output

Overrides the calculated friction of the collision.

Note

May not behave as expected for f larger than 1.0 or less than 0.

Introduced in platform.apiLevel = '2.0'

12.11.14 shapes

```
shapeA, shapeB = physics.Arbiter:shapes()
```

Parameter	Type	Description
self	**in** physics.Arbiter	The input Arbiter
shapeA	**out** physics.Shape	The first Shape in the collision
shapeB	**out** physics.Shape	The second Shape in the collision

Returns shapeA and shapeB in the order they were defined in the collision handler associated with this Arbiter.

Introduced in platform.apiLevel = '2.0'

12.11.15 totalImpulse

```
ivec = physics.Arbiter:totalImpulse()
```

Parameter	Type	Description
self	**in** physics.Arbiter	The input Arbiter
ivec	**out** physics.Vect	The vector impulse applied to resolve the collision

Returns the vector impulse that was applied this step to resolve the collision.

Introduced in platform.apiLevel = '2.0'

12.11.16 totalImpulseWithFriction

```
ivec = physics.Arbiter:totalImpulseWithFriction()
```

Parameter	Type	Description
self	**in** physics.Arbiter	The input Arbiter
ivec	**out** physics.Vect	The vector impulse applied to resolve the collision

Returns the vector impulse that was applied this step to resolve the collision. The calculated friction is taken into account.

Introduced in platform.apiLevel = '2.0'

12.12 Shape Queries

12.12.1 pointQuery

```
bool = physics.Shape:pointQuery(point)
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
point	**in** physics.Vect	A point
bool	**out** boolean	True if **point** lies within the bounds of Shape

Returns true if **point** lies within the Shape.

Introduced in platform.apiLevel = '2.0'

12.12.2 segmentQuery

```
info = physics.Shape:segmentQuery(vecta, vectb)
```

Parameter	Type	Description
self	**in** physics.Shape	The input Shape
vecta	**in** physics.Vect	One end point of the segment
vectb	**in** physics.Vect	The other end point of the segment
info	**out** physics.SegmentQueryInfo	Information about where the segment and Shape intersect. Nil if no intersection

Checks if the line segment from **vecta** to **vectb** intersects the Shape. Returns a SegmentQueryInfo object with the result of the query or nil if no intersection.

Note

If a segment query starts inside of a shape then the result is somewhat undefined. Circles and polygons will not report a collision with that shape, and segments will report an incorrect point and normal if they do detect a collision with that shape. To get around this deficiency, use a separate point query to determine if the segment query starts inside of a shape.

Info

See the SegmentQueryInfo methods below for helper routines to convert the results to world coordinates or absolute distance.

Introduced in platform.apiLevel = '2.0'

12.13 Space Queries

12.13.1 pointQuery

```
physics.Space:pointQuery(point, layers, group,
                         function(shape) ... end)
```

Parameter	Type	Description
self	**in** physics.Space	The input Space
point	**in** physics.Vect	A point
layers	**in** number	A bitmap of the layers. Match if shape.layers intersects **layers**
group	**in** number	The group number to check. Match if Shape is not in **group**
function	function(shape)	A function to call providing each Shape in turn that matches the criteria

Queries the Space for all shapes that contain **point** and match **layers** but not in **group**. The **function** is called with each matching Shape. Sensor Shapes are included.

Introduced in platform.apiLevel = '2.0'

12.13.2 pointQueryFirst

```
shape = physics.Space:pointQueryFirst(point, layers, group)
```

Parameter	Type	Description
self	**in** physics.Space	The input Space
point	**in** physics.Vect	A point
layers	**in** number	A bitmap of the layers. Match if shape.layers intersects **layers**
group	**in** number	The group number to check. Match if Shape is not in **group**

Queries Space at **point** and returns the first Shape that matches the given **layers** and and not in **group**. Returns nil if no Shape was found. Sensor Shapes are ignored.

Introduced in platform.apiLevel = '2.0'

12.13.3 segmentQuery

```
physics.Space:segmentQuery(startvect, stopvect, layers, group,
                           function(shape, t, normal) ... end)
```

Parameter	Type	Description
self	**in** physics.Space	The input Space
startvect	**in** physics.Vect	An end point of the segment
stopvect	**in** physics.Vect	Other end point of the segment
layers	**in** number	A bitmap of the layers. Math if shape.layers intersects **layers**
group	**in** number	The group number to check. Match if object is not in **group**
function	function(shape, t, normal)	A function to call providing each Shape in turn that matches the criteria

Queries the Space for all Shapes that intersect the line segment from **startvect** to **stopvect** and match **layers** and not in **group**. The **function** is called with each matching Shape. Sensor Shapes are included.

The callback function is called with each Shape, proportion of distance along the line segment (a fraction from 0 to 1), and the surface normal vector of the intersection point of the Shape.

Introduced in platform.apiLevel = '2.0'

12.13.4 segmentQueryFirst

```
info = physics.Space:segmentQueryFirst(startvect, stopvect,
                                       layers, group)
```

Parameter	Type	Description
self	**in** physics.Space	The input Space
startvect	**in** physics.Vect	An end point of the segment
stopvect	**in** physics.Vect	Other end point of the segment
layers	**in** number	A bitmap of the layers. Matches if shape.layers intersects **layers**
group	**in** number	The group number to check. Matches if Shape is not in **group**
info	**out** physics.SegmentQueryInfo	Information about where the segment and Shape intersect. Nil if no intersection

Queries Space along the line segment from **startvect** to **stopvect** and returns the first intersecting Shape that matches **layers** and not in **group**. Returns a SegmentQueryInfo object with the first Shape that matches the query or nil if no intersection.

Introduced in platform.apiLevel = '2.0'

12.14 SegmentQueryInfo

A SegmentQueryInfo object is a Lua dictionary table with three fields.

Key	Value
shape	Shape object found in a query.
t	Fractional distance (0 .. 1) from the start of the line segment to the intersection of the Shape.
n	Surface normal vector of the Shape at the intersection point.

This object also has the following helper routines which convert information in a SegmentQueryInfo object to world coordinates or an absolute distance along the line segment.

12.14.1 hitDist

```
d = SegmentQueryInfo:hitDist(startvect, stopvect)
```

Parameter	Type	Description
self	**in** physics.SegmentQueryInfo	The input SegmentQueryInfo
startvect	**in** physics.Vect	An end point of the segment
stopvect	**in** physics.Vect	Other end point of the segment
d	**out** number	Hit distance

Returns the absolute distance where the segment first hit the Shape.

Introduced in platform.apiLevel = '2.0'

12.14.2 hitPoint

```
p = SegmentQueryInfo:hitPoint(startvect, stopvect)
```

Parameter	Type	Description
self	**in** physics.SegmentQueryInfo	The input SegmentQueryInfo
startvect	**in** physics.Vect	An end point of the segment
stopvect	**in** physics.Vect	Other end point of the segment
p	**out** physics.Vect	Hit point

Returns the hit point in world coordinates where the segment between **startvect** and **stopvect** first intersects the Shape.

Introduced in platform.apiLevel = '2.0'

Chapter 13

Platform Library

Platform specific information is available through the platform library.

13.1 apiLevel

```
platform.apiLevel
```

Uniquely identifies the Script API revision offered by a TI-Nspire™ software version.

The following list indicates the currently supported script API revisions:

- '1.0'
- '2.0'

To know the current revision of the script API, read the platform.apiLevel.

```
print(platform.apiLevel)
```

Setting the API level allows scripts written in previous TI-Nspire™ versions to run on the current version. The default value is set to highest revision value supported in the current TI-Nspire™ version.

```
Platform.apiLevel = '1.0'  -- Change the API level to '1.0'
```

Note

- If present, the platform.apiLevel = 'X.X' statement should be in the main part of the script only. It should not be inside any function. It is advisable to place it on the first line of the script.
- Dynamically loaded scripts (**dostring()**) will use the same 'platform.apiLevel = 'X.X" as the main script.

Introduced in platform.apiLevel = "2.0"

13.2 gc

```
platform.gc()
```

Returns a dummy graphics context. It is typically used to measure pixel lengths and heights of strings when a normal graphics context is not available. This may be the case when creating new text elements when the script app is initialized. A graphics context is available only during paint events, and that may be too late to create and size the containers for text fields.

This graphics context should not be used to draw graphics because it is not guaranteed to be associated with a window.

Here is an example of using the dummy graphics context to get the string pixel length and height.

```
local gc = platform.gc()
gc:begin()
local width = gc:getStringWidth(a_string)
local height = gc:getStringHeight(a_string)
gc:finish()
```

It is important to use gc:begin() to set up the graphics context before using it in the getString function and to call gc:finish() to relinquish it when finished with it.

Introduced in platform.apiLevel = "1.0"
Removed in platform.apiLevel = "2.0"

13.3 hw

```
platform.hw()
```

Returns a numeric value that indicates the CPU speed of the host hardware. The higher the number, the faster the hardware.

level	host hardware
3	TI-Nspire™ B&W and CX handheld devices
7	Microsoft® Windows®, Mac® or web player

Introduced in platform.apiLevel = "2.0"

13.4 isColorDisplay

```
platform.isColorDisplay()
```

Returns true if the display of the host platform is color. Returns false if the display is grayscale.

Introduced in platform.apiLevel = "1.0"

13.5 isDeviceModeRendering

```
platform.isDeviceModeRendering()
```

Returns true if the script is running on the handheld device or in the emulator of the desktop software. Returns false if the script is running in the normal view of the desktop software.

Note

platform.isDeviceModeRendering is not available during script initialization or within **on.restore**.

Introduced in platform.apiLevel = "1.0"

13.6 registerErrorHandler

```
platform.registerErrorHandler(function(lineNumber, errorMessage,
                                       callStack, locals) ... end)
```

This function sets the error handler callback function for the script. Setting an error handler callback function provides control over what happens when an error is encountered in the script. Returning a true value prevents reporting the Error to the user. The script will continue executing on the next event.

Note

> The error handler callback function is not called for errors that occur during initialization or within **on.restore**.

Introduced in platform.apiLevel = "2.0"

13.7 window

```
platform.window
```

Returns the window object that the script application currently owns. The window consists of the portion of the page allotted to the script app. Several applications can be visible when the page is arranged in a split layout. Each visible application has its own window.

The window object has several methods of particular interest.

Introduced in platform.apiLevel = "1.0"

13.7.1 height and width

```
platform.window:height()
platform.window:width()
```

Routines **height()** and **width()** return the pixel height and width respectively of the display window.

Introduced in platform.apiLevel = "1.0"

13.7.2 invalidate

```
platform.window:invalidate(x, y, width, height)
```

This function invalidates a region of the window and forces it to repaint. x and y default to (0, 0) and width and height default to the pixel width and height of the window. The

entire window can be forced to repaint with a call to **platform.window:invalidate()**, which allows all parameters to take their default values.

Introduced in platform.apiLevel = "1.0"

13.7.3 setFocus

```
platform.window:setFocus(true or false)
```

This function sets the focus to the main window. Any focus of other objects is removed (D2Editor).

Introduced in platform.apiLevel = "2.0"

13.8 withGC

```
platform.withGC(function, args)
```

Executes function(args) within a dummy graphics context and returns all return values from function(). It is used typically to measure the pixel lengths and heights of strings when a normal graphics context is not available. When creating new text elements, this may be the case when the script app is initialized. A graphics context is available only during paint events, and that may be too late to create and size the containers for text fields.

This graphics context cannot be used to draw.

Here is an example of using withGC() to get the pixel length and height of a string.

```
function setFont(family, style, size, gc)
    f, s, z = gc:setFont(family, style, size)   -- Set the font
end

function getHeightWidth(str, gc)
    width = gc:getStringWidth(str)       -- Ppixel length of str
    height = gc:getStringHeight(str)     -- Pixel height of str
    return height, width
end

platform.withGC(setFont, 'serif', 'b', 12)
height, width = platform.withGC(getHeightWidth, 'Hello World')
```

Note: Although you could combine the two functions above into a single function to avoid calling **withGC()** twice, that is not required because the dummy graphics context remembers its state.

Introduced in platform.apiLevel = "2.0"

Chapter 14

Module Library

```
require '<library name>'
```

Use **require** to load predefined libraries in TI-Nspire™ software. Please see the following table.
The behavior of **require** is the same as in standard Lua but the available libraries are restricted. User-defined libraries are not supported.

Library	Description
color	Table defining colors used in TI-Nspire™ software to color objects via the color picker.
physics	Loads the physics module.

Colors defined in color table:
"black", "darkgray", "gray", "mediumgray", "lightgray", "white", "navy", "blue", "brown", "red", "magenta", "orange", "yellow", "green", "dodgerblue"

Introduced in platform.apiLevel = '2.0'

Chapter 15

String Library Extension

In addition to the standard Lua string functions, a few routines aid handling Unicode strings.

15.1 split

```
string.split(str [,delim])
```

Divides str into substrings based on a delimiter, returning a list of the substrings. The default pattern for the delimiter is white space ("%s+").

Introduced in platform.apiLevel = '1.0'

15.2 uchar

```
string.uchar(chnum, ...)
```

Unicode characters can be included in strings by encoding them in UTF-8. This routine converts one or more Unicode character numbers into a UTF-8 string.

Introduced in platform.apiLevel = '1.0'

15.3 usub

```
string.usub(str, startpos, endpos)
or
str:usub(startpos, endpos)

print(string.usub("abc", 1, 1)) -- prints "a"
print(string.usub("abc", 2, 2)) -- prints "b"
print(string.usub("abc", 2, 3)) -- prints "bc"
```

This routine returns a substring of str. It is the Unicode version of string.sub. It accounts for multi-byte characters encoded in UTF-8.

Caution

> This is an expensive routine. It allocates a temporary memory buffer during its operation.

Introduced in platform.apiLevel = '1.0'

Chapter 16

Timer Library

Each script application has one timer at its disposal. The timer resolution depends on the platform. It is about 0.02 second on the handheld. Please be cautious with short timer periods on the handheld.

The script application should implement the **on.timer()** function to respond to timer expiration.

The timer continues to send ticks to the script application even when its window is not visible on the screen.

The timer is stopped automatically when the document containing the script application is closed or if the script application is deleted from the document.

16.1 getMilliSecCounter

```
timer.getMilliSecCounter()
```

Returns the value of the internal millisecond counter. The counter rolls over to zero when it passes 2^{32} milliseconds.

Introduced in platform.apiLevel = '1.0'

16.2 start

```
timer.start(period)
```

Starts the timer with the given period in seconds. The period must be ≥ 0.01 (10 ms). If the timer is already running when this routine is called, the timer is reset to the new period.

Introduced in platform.apiLevel = '1.0'

Caution

> timer.start() should not be called when processing an on.timer() event unless it is the final statement before the on.timer() event completes.

16.3 stop

```
timer.stop()
```

Stops the timer.

Introduced in platform.apiLevel = '1.0'

Chapter 17

Tool Palette Library

The tool palette provides a menu of commands from which the user can select commands that invoke functionality of the script app.

17.1 register

```
toolpalette.register(menuStructure)
```

The script app uses this routine to register its tool palette with the TI-Nspire™ framework. The menu structure is a table describing the name of each toolbox, the menus that appear in each tool box, and the function to call when the user invokes the menu item.

This example serves to demonstrate the layout of a tool palette's menu structure.

```
menu = {
  {"Mode", -- Tool box "Mode"
    {"Decimal", setDec}, -- Menu item "Decimal" calls setDec()
    {"Hexadecimal", setHex},
    "-", -- Section divider
    {"Signed", setSigned},
    {"Unsigned", setUnsigned},
  },
  {"Boolean",
    {"And", binopAnd},
    {"Or", binopOr},
  },
}  toolpalette.register(menu)
```

toolpalette.register can be called once in the top level flow of the script app. Once registered, the tool palette is managed automatically by the TI-Nspire™ framework. Up to 15 toolboxes can be created with up to 30 menu items each.

When the user chooses an item from a tool box, the associated function is called with two parameters: the name of the toolbox and the name of the menu item.

Info

Beginning with platform.apiLevel = '2.0', the names of the tool palette items can be changed dynamically while the program is running.
Calling toolpalette.register(nil) deactivates the toolpalette.

Introduced in platform.apiLevel = '1.0'

17.2 enable

```
toolpalette.enable(toolname, itemname, enable)
```

This routine enables or disables a menu item in the tool palette. Parameter **toolname** is a string containing the name of the top level tool box. Parameter **itemname** is a string containing the name of the menu item. Parameter **enable** is a Boolean value that enables the menu item if true or disables the menu item if false.

This routine returns true if the menu item was properly enabled or disabled. It returns nil if the toolname / itemname pair cannot be found in the registered menu items.

Note

toolpalette.register() mist be called prior to **toolpalette.enable()**.

Introduced in platform.apiLevel = '1.0'

17.3 enableCut

```
toolpalette.enableCut(enable)
```

This routine enables or disables the Edit > Cut menu command. Parameter **enable** is a Boolean value that enables the command if true or disables the menu item if false.

Introduced in platform.apiLevel = '1.0'

17.4 enableCopy

```
toolpalette.enableCopy(enable)
```

This routine enables or disables the Edit > Copy menu command. Parameter **enable** is a Boolean value that enables the command if true or disables the menu item if false.

Introduced in platform.apiLevel = '1.0'

17.5 enablePaste

```
toolpalette.enablePaste(enable)
```

This routine enables or disables the Edit > Paste menu command. Parameter **enable** is a Boolean value that enables the command if true or disables the menu item if false.

Introduced in platform.apiLevel = '1.0'

Chapter 18

Variable Library

A symbol table is used by the TI-Nspire™ math engine to calculate and store variables. This library gives scripts access to the variables stored in the symbol table.

Not all variables in the symbol table have compatible types in Lua. But many important variable types are supported: real and integer numbers, strings, and lists of numbers and strings, matrices (represented in Lua as lists of lists), and boolean constants true and false.

18.1 list

```
var.list()
```

This function returns a list of names of variables currently defined in the symbol table.

Introduced in platform.apiLevel = '1.0'

18.2 makeNumericList

```
var.makeNumericList(name)
```

Creates a list in the symbol table with the given **name**. The list is optimized to hold numeric values. Routines **storeAt** and **recallAt** operate much more efficiently on lists that are created with this function.

Usage Note

This function cannot be used to create a numeric matrix. Routines var.recallAt and var.storeAt documented below will work with matrices but only if they are created by some other means.

```
var.store("mat", {{1,2}, {3,4}})  -- creates matrix mat
var.storeAt("mat", 13.3, 1, 1)
val = var.recallAt("mat", 1, 1)
```

Introduced in platform.apiLevel = '2.0'

18.3 monitor

```
var.monitor(name)
```

Turns on monitoring of the math variable with given **name**. Whenever another application changes the math variable, this script application's on.varChange handler is called. See the description of on.varChange below. Any other return value from 0 is an error value.

Introduced in platform.apiLevel = '1.0'

18.4 recall

```
var.recall(name)
```

Returns the value of a math variable with the given **name**. If the type of the named variable has no compatible Lua type, then nil and an error message are returned.

Introduced in platform.apiLevel = '1.0'

18.5 recallAt

```
var.recallAt(name, col [,row])
```

Recalls a value from a cell of a list or matrix in the symbol table. **col** is a 1-based column number of the matrix or list. **row** is a 1-based row number. **row** is only required when recalling a value from a matrix.

This function is optimized to work with numeric values and normally returns a number. If the value of the recalled cell is not numeric, this function returns nil and an error message string.

Introduced in platform.apiLevel = '2.0'

18.6 recallStr

```
var.recallStr(name)
```

Returns the value of a math variable with the given **name** as a string. Some math types have no compatible Lua type but all math types can be represented as a string. If the value cannot be recalled even as a string, this function returns nil and an error message.

Introduced in platform.apiLevel = '1.0'

18.7 store

```
var.store(name, value)
```

Stores value as a math variable with the given **name**. If the value cannot be stored, an error message is returned. Otherwise, nil is returned.

Introduced in platform.apiLevel = '1.0'

18.8 storeAt

```
var.storeAt(name, numericValue, col [, row])
```

Stores a numeric value into an element of a math list or matrix with the given **name**. **col** is a 1-based column number of the matrix or list. **row** is a 1-based row number. **row** is only required when storing a value into a matrix.

The value must be numeric. Any other type raises an error.

New values can be appended to a list by storing to one column past the end of the list. This function is useful particularly as an optimization when adding new values to a list during a simulation.

Returns nil on success or "cannot store" if the value cannot be stored at the given index.

Introduced in platform.apiLevel = '2.0'

18.9 unmonitor

```
var.unmonitor(name)
```

Turns off monitoring of the named math variable.

Introduced in platform.apiLevel = '1.0'