Regular API functions

simAddBanner / sim.addBanner

Description	Adds a banner to the scene. Banners created in a simulation script will be automatically removed at simulation end. See also sim.removeBanner and sim.addDrawingObject
C synopsis	simInt simAddBanner(const simChar* label,simFloat size,simInt options,const simFloat* positionAndEulerAngles,simInt parentObjectHandle,const simFloat* labelColors,const simFloat* backgroundColors)
C parameters	**label**: the label to display on the banner **size**: the height in meters of the banner. When the option sim_banner_keepsamesize is used, this argument represents the banner height in pixels instead **options**: a combination of banner options **positionAndEulerAngles**: 6 values representing the banner's position and orientation in space. Those values are absolute if the argument parentObjectHandle is -1, otherwise they are relative to the parent object's position and orientation. This argument can be NULL, in which case the identity transformation is assumed **parentObjectHandle**: the handle of a scene object you wish to attach the banner to, or -1 if the banner should be independent. **labelColors**: 12 values representing the RGB values (0-1) for the 3 color components of the text (ambient_diffuse RGB, 3 reserved values (set to zero), specular RGB and emissive RGB). Can be NULL, in which case a black color will be used **backgroundColors**: 12 values representing the RGB values (0-1) for the 3 color components of the text background (ambient_diffuse RGB, 3 reserved values (set to zero), specular RGB and emissive RGB). Can be NULL, in which case a white color will be used
C return value	handle of the banner if successful, -1 otherwise
Lua synopsis	number bannerID=sim.addBanner(string label,number size,number options,table_6 positionAndEulerAngles=nil,number parentObjectHandle=nil,table_12 labelColors=nil,table_12 backgroundColors=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddDrawingObject / sim.addDrawingObject

Description	Adds a drawing object that will be displayed in the scene. Drawing objects are containers that hold several items of a given type. This can be used for several different applications (simulation of paint, simulation of welding seam, display of 3D objects, etc.). Drawing objects created in a simulation script will be automatically removed at simulation end. See also sim.addDrawingObjectItem, sim.removeDrawingObject and the point cloud functionality.
C synopsis	simInt simAddDrawingObject(simInt objectType,simFloat size,simFloat duplicateTolerance,simInt parentObjectHandle,simInt maxItemCount,const simFloat* ambient_diffuse,const simFloat* setToNULL,const simFloat* specular,const simFloat* emission)
C parameters	**objectType**: a drawing object type combined with attributes **size**: size of the item (width of lines or size of points are in pixels, other sizes are in meters **duplicateTolerance**: if different from 0.0, then a call to simAddDrawingObjectItem will only add the item if there is no other item within duplicateTolerance distance. Useful to avoid adding a too high density of points, is however not appropriate when using a large number of points (slower operation). Applicable only for single vertex items. **parentObjectHandle**: handle of the scene object where the drawing items should keep attached to (if the scene object moves, the drawing items will also move), or -1 if the drawing items are relative to the world (fixed) **maxItemCount**: maximum number of items this object can hold. **ambient_diffuse**: default ambient/diffuse color (pointer to 3 rgb values). Can be NULL **setToNULL**: not used, set to NULL **specular**: default specular color (pointer to 3 rgb values). Can be NULL **emission**: default emissive color (pointer to 3 rgb values). Can be NULL
C return value	handle of the drawing object if successful, -1 otherwise
Lua synopsis	number drawingObjectHandle=sim.addDrawingObject(number objectType,number size,number duplicateTolerance,number parentObjectHandle,number maxItemCount,table_3

	ambient_diffuse=nil,nil,table_3 specular=nil,table_3 emission=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddDrawingObjectItem / sim.addDrawingObjectItem

Description	Adds an item (or clears all items) to a previously inserted drawing object. See also sim.addDrawingObject and sim.removeDrawingObject
C synopsis	simInt simAddDrawingObjectItem(simInt objectHandle,const simFloat* itemData)
C parameters	**objectHandle**: handle of a previously added drawing object **itemData**: data relative to an item. If the item is a point item, 3 values are required (x;y;z). If the item is a line item, 6 values are required, and if the item is a triangle item, 9 values are required. Additional values (auxiliary values) might be required depending on the drawing object attributes. See the drawing object types and attributes for more information. If NULL the drawing object is emptied of all its items
C return value	-1 if operation was not successful. If the point was added, then the return value is >0, if it was not added (e.g. drawing object is saturated or the item was merged with an existing item), then the return value will be 0.
Lua synopsis	number result=sim.addDrawingObjectItem(number drawingObjectHandle,table itemData)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddForce / sim.addForce

Description	Adds a non-central force to a shape object that is dynamically enabled. Added forces are cumulative, and are reset to zero after sim.handleDynamics was called. See also sim.addForceAndTorque.
C synopsis	simInt simAddForce (simInt shapeHandle,const simFloat* position,const simFloat* force)
C parameters	**shapeHandle**: handle of a dynamically enabled shape **position**: pointer to 3 values that represent the relative position where the force should be applied. **force**: pointer to 3 values that represent the force (in relative coordinates) to add.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.addForce(number shapeHandle,table_3 position,table_3 force)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddForceAndTorque / sim.addForceAndTorque

Description	Adds a force and/or torque to a shape object that is dynamically enabled. Forces are applied at the center of mass. Added forces and torques are cumulative, and are reset to zero after sim.handleDynamics was called. See also sim.addForce.
C synopsis	simInt simAddForceAndTorque(simInt shapeHandle,const simFloat* force,const simFloat* torque)
C parameters	**shapeHandle**: handle of a dynamically enabled shape **force**: pointer to 3 values that represent the force (in absolute coordinates) to add. Can be NULL. **torque**: pointer to 3 values that represent the torque (in absolute coordinates) to add. Can be NULL
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.addForceAndTorque(number shapeHandle,table_3 force,table_3 torque)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddGhost / sim.addGhost

Description	Adds a light copy of a shape in its current configuration, as a ghost object. Ghosts have a visual start

	and end time, and are automatically played back during simulation (i.e. visualized), but they do not influence a simulation otherwise. Ghost are a convenient way to visually compare several simulation runs. Ghosts can be modified or cleared with sim.modifyGhost. Ghosts can also be cleared in the environment properties.
C synopsis	simInt simAddGhost(simInt ghostGroup,simInt objectHandle,simInt options,simFloat startTime,simFloat endTime,const simFloat* color)
C parameters	**ghostGroup**: an identifier that allows grouping several ghosts **objectHandle**: the handle of a shape, or the handle of a model base. Only currently visible shapes can be duplicated as ghosts. **options**: bit-coded: **bit0 (1)** set=the provided objectHandle is a model base, and all visible shapes in the model will be duplicated as ghosts **bit1 (2)** set=the provided start- and end-times will be played-back in real-time **bit2 (4)** set=preserve the original colors **bit3 (8)** set=force invisible objects to appear too **bit4 (16)** set=create an invisible ghost **bit5 (32)** set=backface culling for the ghost (only when using custom colors) **startTime**: the time at which the ghost should appear. **endTime**: the time at which the ghost should disappear. **color**: 12 values that represent the color of the ghost (ambient_diffuse RGB, 3 reserved values (set to zero), specular RGB and emissive RGB). Can be NULL for default colors.
C return value	-1 if operation was not successful, otherwise a ghost ID. Several ghosts might share the same ID (e.g. when a ghost was added with bit0 of **options** set)
Lua synopsis	number ghostId=sim.addGhost(number ghostGroup,number objectHandle,number options,number startTime,number endTime,table_12 color=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddModuleMenuEntry

Description	Attaches a menu entry in the simulator's module menu. This is useful if you created an plugin that can display a custom dialog in V-REP for specific settings/operations. If the user selects an item in the simulator's module menu, a sim_message_eventcallback_menuitemselected message is generated. See also the plugin v_repMessage entry point.
C synopsis	simInt simAddModuleMenuEntry(const simChar* entryLabel,simInt itemCount,simInt* itemHandles)
C parameters	**entryLabel**: entry label. The same label can be used in consecutive calls (also from different plugins), in which case a sub-menu will group all items under the same label. If you do not plan adding several items, use "" for entryLabel. **itemCount**: number of items, including separators. If entryLabel is "", then itemCount should be 1 **itemHandles**: pointer to the item handles. Make sure the pointer can hold "itemCount" number of elements. Use simSetModuleMenuItemState to set-up the individual items.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	-
Lua parameters	-
Lua return values	-

simAddObjectCustomData (DEPRECATED)

Description	DEPRECATED. See sim.writeCustomDataBlock instead.

simAddObjectToCollection / sim.addObjectToCollection

Description	Adds an object (or a group of objects) to a collection. See also sim.emptyCollection, sim.removeCollection, sim.getCollectionHandle, sim.getObjectHandle, sim.getCollectionObjects and sim.createCollection.
C synopsis	simInt simAddObjectToCollection(simInt collectionHandle,simInt objectHandle,simInt what,simInt options)

C parameters	**collectionHandle**: the handle of a collection. **objectHandle**: the handle of an object. **what**: the type of object (or group of objects) to add. Following are allowed values: *sim_handle_single* (for a single object), *sim_handle_all* (for all objects in the scene), *sim_handle_tree* (for a tree of objects), or *sim_handle_chain* (for a chain of objects (i.e. an inverted tree)). **options**: bit-coded options: bit 0 set (1): the specified object (or group of objects) is removed from the collection. Otherwise it is added. bit 1 set (2): the specified object is not included in the group of objects, if sim_handle_tree or sim_handle_chain is specified (i.e. the tree base or tip is excluded).
C return value	-1 if operation was not successful.
Lua synopsis	number result=sim.addObjectToCollection(number collectionHandle,number objectHandle,number what,number options)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddObjectToSelection / sim.addObjectToSelection

Description	Adds an object to the selection. See also sim.removeObjectFromSelection and sim.getObjectSelection.
C synopsis	simInt simAddObjectToSelection(simInt what,simInt objectHandle)
C parameters	**what**: indicates what we want to add. Valid values are sim_handle_single (adds just one object), sim_handle_all (adds all objects in the scene), sim_handle_tree (adds the tree with base objectHandle (inclusive)) and sim_handle_chain (adds the chain with tip objectHandle (inclusive)) **objectHandle**: handle of an object. Doesn't have a meaning if "what" is sim_handle_all
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	There are two versions of this function: (1) number result=sim.addObjectToSelection(number what,number objectHandle) (2) number result=sim.addObjectToSelection(table objectHandles)
Lua parameters	(1) Same as C-function. The second argument can be omitted if "what" is sim.handle_all (2) **objectHandles**: table of object handles. Can be nil
Lua return values	Same as C-function

simAddParticleObject / sim.addParticleObject

Description	Adds a particle object that will be simulated and displayed in the scene. Particle objects are containers that hold several items (particles) of a given type. This can be used for several different applications (e.g. simulation of air/water jets) See also sim.addParticleObjectItem and sim.removeParticleObject
C synopsis	simInt simAddParticleObject(simInt objectType,simFloat size,simFloat density,const simVoid* parameters,simFloat lifeTime,simInt maxItemCount,const simFloat* ambient_diffuse,const simFloat* setToNULL,const simFloat* specular,const simFloat* emission)
C parameters	**objectType**: a particle object type combined with attributes **size**: diameter of the particles (spheres) **density**: density of the particles **parameters**: points to an array of values, allowing to specify additional parameters. Can be NULL. The first value (an integer) indicates how many parameters will be set. All following values come in pair (an integer indicating what parameter, and a float indicating the parameter value. Following indicates the parameters: 0: Bullet friction coefficient (default: 0.0) 1: Bullet restitution coefficient (default: 0.0) 2: ODE friction coefficient (default: 0.0) 3: ODE soft ERP value (default: 0.2) 4: ODE soft CFM values (default: 0.0) 5: Bullet, ODE, Newton and Vortex linear drag parameter (default: 0.0). Adds a force opposite to the particle velocity (f=v*parameter) 6: Bullet, ODE, Newton and Vortex quadratic drag parameter (default: 0.0). Adds a force opposite to the particle velocity (f=v*v*parameter) 7: Bullet, ODE, Newton and Vortex linear drag parameter in air (z>0) if sim_particle_water was specified (default: 0.0). Adds a force opposite to the particle velocity (f=v*parameter) 8: Bullet, ODE, Newton and Vortex quadratic drag parameter in air (z>0) if sim_particle_water was specified (default: 0.0). Adds a force opposite to the particle velocity (f=v*v*parameter)

	9: Vortex friction (default: 0.0) 10: Vortex restitution (default: 0.0) 11: Vortex restitution threshold (default: 0.001) 12: Vortex compliance (default: 0.0) 13: Vortex damping (default: 0.0) 14: Vortex adhesive force (default: 0.0) 15: Newton static friction (default: 0.0) 16: Newton kinetic friction (default: 0.0) 17: Newton restitution (default: 0.0) If a parameter is not set, then its default value is used. As an example, following array: [3,0,0.5,2,0.5,9,0.5] will set Bullet's, ODE's and Vortex's friction coefficients to 0.5 **lifeTime**: simulation time after which the particles are destroyed. Set to 0.0 for an unlimited lifetime. **maxItemCount**: the maximum number of particles that this object can hold **ambient_diffuse**: default ambient/diffuse color (pointer to 3 rgb values). Can be NULL **setToNULL**: not used, set to NULL **specular**: default specular color (pointer to 3 rgb values). Can be NULL **emission**: default emissive color (pointer to 3 rgb values). Can be NULL
C return value	handle of the particle object if successful, -1 otherwise
Lua synopsis	number particleObjectHandle=sim.addParticleObject(number objectType,number size,number density,table parameters,number lifeTime,number maxItemCount,table_3 ambient_diffuse=nil,nil,table_3 specular=nil,table_3 emission=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddParticleObjectItem / sim.addParticleObjectItem

Description	Adds an item (or clears all items) to a previously inserted particle object. See also sim.addParticleObject and sim.removeParticleObject
C synopsis	simInt simAddParticleObjectItem(simInt objectHandle,const simFloat* itemData)
C parameters	**objectHandle**: handle of a previously added particle object **itemData**: data relative to an item. All items (particles) require at least 6 values:Â p1x, p1y, p1z, p2x, p2y, p2z with p1 is the particle start position, p2-p1 is the particle initial velocity vector. Auxiliary values might be required depending on the particle object attributes. See the particle object type combined with attributes for more information. If NULL the particle object is emptied of all its items
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.addParticleObjectItem(number particleObjectHandle,table itemData)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddPointCloud / sim.addPointCloud (DEPRECATED)

Description	DEPRECATED. See the point cloud object and point cloud related functions instead.

simAddSceneCustomData (DEPRECATED)

Description	DEPRECATED. See sim.writeCustomDataBlock instead.

simAddScript / sim.addScript

Description	Inserts a new script. Use with care when simulation is running. See also sim.associateScriptWithObject.
C synopsis	simInt simAddScript(simInt scriptType)
C parameters	**scriptType**: type of the script.
C return value	handle of the new script, or -1 in case of an error

Lua synopsis	number scriptHandle=sim.addScript(number scriptType)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddStatusbarMessage / sim.addStatusbarMessage (remote API equivalent: simxAddStatusbarMessage)

Description	Adds a message to the status bar. See also sim.auxiliaryConsoleOpen.
C synopsis	simInt simAddStatusbarMessage(const simChar* message)
C parameters	**message**: message. Will clear the status bar if message is NULL.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.addStatusbarMessage(string message)
Lua parameters	**message**: message. Will clear the status bar if message argument is omitted.
Lua return values	Same as C-function

simAdjustRealTimeTimer

Description	Adjusts the real time timer of a simulation. This allows correcting for effects that might appear if for a reason or another the simAdvanceSimulationByOneStep cannot be called for some time (for instance during a resize action of the simulator window (the main thread is captured in a modal-type message loop)).
C synopsis	simInt simAdjustRealTimeTimer(simInt instanceIndex,simFloat deltaTime)
C parameters	**instanceIndex**: no use anymore. set to 0. **deltaTime**: time correction value in seconds
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	-
Lua parameters	-
Lua return values	-

simAdjustView / sim.adjustView

Description	Adjusts parameters of a view. See also the sim.floatingViewAdd and sim.cameraFitToView functions.
C synopsis	simInt simAdjustView(simInt viewHandleOrIndex,simInt associatedViewableObjectHandle,simInt options,const simChar* viewLabel)
C parameters	**viewHandleOrIndex**: the handle of the view (can also be a floating view), or the index of the view. **associatedViewableObjectHandle**: handle of the object that you wish to associate with the view. Must be a viewable object. Can also be -1, in which case the view is emptied **options**: bit-coded: **bit0-bit3**=the 3D display mode (0=solid rendering, 1=wireframe rendering) **bit4 (16)** set=orthogonal projection (otherwise perspective projection) **bit5 (32)** set=x/y graph display (otherwise time-graph display) **bit6 (64)** set=floating view is removed at simulation end **bit7 (128)** set=floating view is ignored during a scene save operation **bit8 (256)** set=the view is not modified. The return value of the function indicates if the view still exists (2), or does not exist anymore (1). No error is generated. **bit9 (512)** set=the view is not modified. The return value of the function represents the object associated with the view. **bit10 (1024)** set=x/y graph has x view size proportional to y view size. **viewLabel**: a label that will be displayed at the top of a floating view. If NULL is specified, then the name of the associated viewable object is taken as label.
C return value	A value >0 in case of success
Lua synopsis	number result=sim.adjustView(number viewHandleOrIndex,number associatedViewableObjectHandle,number options,string viewLabel=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAdvanceSimulationByOneStep

Description	Advances the simulation time by one time step. Call this function only if the simulation is advancing (see sim.getSimulationState) and after having called simHandleMainScript.
C synopsis	simInt simAdvanceSimulationByOneStep()
C parameters	None
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	-
Lua parameters	-
Lua return values	-

simAnnounceSceneContentChange / sim.announceSceneContentChange

Description	Announces a change in the scene. This is required for the undo/redo function to operate properly when performing changes via the API. Only call this function directly after a change was made through a dialog element (e.g. a checkbox was checked/unchecked) and that change was reported to the scene (e.g. with sim.writeCustomDataBlock). What this call will do is following: the whole scene will be serialized (saved) to memory as a "scene image" and compared to a previously memorized "scene image". If both images are same, then the last image is discarded, otherwise only the changes between the two images are memorized. A call to this function has no effect (and doesn't generate any error) when called during simulation or when in edit mode.
C synopsis	simInt simAnnounceSceneContentChange()
C parameters	None
C return value	-1 if operation was not successful, 0 if nothing was memorized, or 1 if changes were memorized.
Lua synopsis	number result=sim.announceSceneContentChange()
Lua parameters	-
Lua return values	Same as C function

simAppendScriptArrayEntry (DEPRECATED)

Description	DEPRECATED. See sim.setScriptVariable instead.

simApplyMilling / sim.applyMilling

Description	Applies changes made during milling operations to a cuttable object (e.g. a shape). This requires some calculation time. Once changes were applied, they cannot be reset anymore. If the milling operation milled away the whole object, then the object is removed from the scene. The calculation structure linked to the object is removed and an updated calculation structure might be calculated (might take some calculation time). See also sim.resetMilling, sim.handleMill and sim.resetMill.
C synopsis	simInt simApplyMilling(simInt objectHandle)
C parameters	**objectHandle**: handle of the cut object or sim_handle_all to apply changes to all cut objects.
C return value	-1 if operation was not successful, 0 if operation was successful but the object was removed from the scene (because entirely cut away) (only available when sim_handle_all is not specified), or 1 if operation was successful and the object still exists in the scene.
Lua synopsis	number result=sim.applyMilling(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAssociateScriptWithObject / sim.associateScriptWithObject

Description	Sets the associated object of a child script. Use with care when simulation is running. See also sim.getObjectAssociatedWithScript, sim.addScript and sim.setScriptText.
C synopsis	simInt simAssociateScriptWithObject(simInt scriptHandle,simInt objectHandle)
C parameters	**scriptHandle**: handle of the child script **objectHandle**: handle of the associated object, or -1 to remove the association
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.associateScriptWithObject(number scriptHandle,number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAuxiliaryConsoleClose / sim.auxiliaryConsoleClose (remote API equivalent: simxAuxiliaryConsoleClose)

Description	Closes an auxiliary console window. See also sim.auxiliaryConsoleOpen.
C synopsis	simInt simAuxiliaryConsoleClose(simInt consoleHandle)
C parameters	**consoleHandle**: the handle of the console window, previously returned by the simAuxiliaryConsoleOpen command
C return value	-1 if operation was not successful. 0 if the console doesn't exist (anymore), in which case no error is generated. 1 if the console window was closed.
Lua synopsis	number result=sim.auxiliaryConsoleClose(number consoleHandle)
Lua parameters	Same as C-function.
Lua return values	Same as C-function.

simAuxiliaryConsoleOpen / sim.auxiliaryConsoleOpen (remote API equivalent: simxAuxiliaryConsoleOpen)

Description	Opens an auxiliary console window for text display. This console window is different from the application main console window. Console window handles are shared across all simulator scenes. See also sim.auxiliaryConsolePrint, sim.auxiliaryConsoleClose and sim.addStatusbarMessage.
C synopsis	simInt simAuxiliaryConsoleOpen(const simChar* title,simInt maxLines,simInt mode,const simInt* position,const simInt* size,const simFloat* textColor,const simFloat* backgroundColor)
C parameters	**title**: the title of the console window **maxLines**: the number of text lines that can be displayed and buffered **mode**: bit-coded value. Bit0 (1) set indicates that the console window will automatically close at simulation end (when called from a simulation script, the consle window will always automatically close at simulation end), bit1 (2) set indicates that lines will be wrapped, bit2 (4) set indicates that the user can close the console window, bit3 (8) set indicates that the console will automatically be hidden during simulation pause, bit4 (16) set indicates that the console will not automatically hide when the user switches to another scene. **position**: the initial position of the console window (x and y value). Can be NULL **size**: the initial size of the console window (x and y value). Can be NULL **textColor**: the color of the text (rgb values, 0-1). Can be NULL **backgroundColor**: the background color of the console window (rgb values, 0-1). Can be NULL
C return value	-1 if operation was not successful. Otherwise a console window handle
Lua synopsis	number consoleHandle=sim.auxiliaryConsoleOpen(string title,number maxLines,number mode,table_2 position=nil,table_2 size=nil,table_3 textColor=nil,table_3 backgroundColor=nil)
Lua parameters	Same as C-function. Last 4 parameters can be omitted too.
Lua return values	Same as C-function

simAuxiliaryConsolePrint / sim.auxiliaryConsolePrint (remote API equivalent: simxAuxiliaryConsolePrint)

Description	Prints to an auxiliary console window. See also sim.auxiliaryConsoleOpen.
C synopsis	simInt simAuxiliaryConsolePrint(simInt consoleHandle,const simChar* text)
C parameters	**consoleHandle**: the handle of the console window, previously returned by the simAuxiliaryConsoleOpen command

	text: the text to append, or NULL to clear the console window
C return value	-1 if operation was not successful. 0 if the console doesn't exist (anymore), in which case no error is generated. 1 if the operation was successful.
Lua synopsis	number result=sim.auxiliaryConsolePrint(number consoleHandle,string text)
Lua parameters	Same as C-function.
Lua return values	Same as C-function.

simAuxiliaryConsoleShow / sim.auxiliaryConsoleShow (remote API equivalent: simxAuxiliaryConsoleShow)

Description	Shows or hides an auxiliary console window. See also sim.auxiliaryConsoleOpen and sim.auxiliaryConsoleClose.
C synopsis	simInt simAuxiliaryConsoleShow(simInt consoleHandle,simBool showState)
C parameters	**consoleHandle**: the handle of the console window, previously returned by the simAuxiliaryConsoleOpen command **showState**: indicates whether the console should be hidden (0) or shown (!=0)
C return value	-1 if operation was not successful. 0 if the console doesn't exist (anymore), in which case no error is generated. 1 if the console window's show state was changed.
Lua synopsis	number result=sim.auxiliaryConsoleShow(number consoleHandle,Boolean showState)
Lua parameters	Same as C-function.
Lua return values	Same as C-function.

simBoolAnd32 / sim.boolAnd32

Description	Performs a 32-bit Boolean AND operation between two numbers. See also sim.boolOr32 and sim.boolXor32.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result=sim.boolAnd32(number value1,number value2)
Lua parameters	**value1**: first value **value2**: second value
Lua return values	Result of the Boolean operation or nil in case of an error

simBoolOr32 / sim.boolOr32

Description	Performs a 32-bit Boolean OR operation between two numbers. See also sim.boolAnd32 and sim.boolXor32.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result=sim.boolOr32(number value1,number value2)
Lua parameters	**value1**: first value **value2**: second value
Lua return values	Result of the Boolean operation or nil in case of an error

simBoolXor32 / sim.boolXor32

Description	Performs a 32-bit Boolean exclusive-OR operation between two numbers. See also sim.boolAnd32 and sim.boolOr32.
C synopsis	-
C parameters	-
C return value	-

Lua synopsis	number result=sim.boolXor32(number value1,number value2)
Lua parameters	**value1**: first value **value2**: second value
Lua return values	Result of the Boolean operation or nil in case of an error

simBreakForceSensor / sim.breakForceSensor (remote API equivalent: simxBreakForceSensor)

Description	Allows breaking a force sensor during simulation. A broken force sensor will lose its positional and orientational constraints. See also sim.readForceSensor.
C synopsis	simInt simBreakForceSensor(simInt objectHandle)
C parameters	**objectHandle**: handle of the object (must be a force sensor)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.breakForceSensor(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simBroadcastMessage

Description	Allows a plugin to communicate with other plugins by broadcasting messages or data that other plugins can intercept. The message is also sent to the plugin that originally broadcasted the message (that module is free to ignore its own message). See V-REP's main client application source code for more details. See also simSendModuleMessage.
C synopsis	simVoid* simBroadcastMessage(simInt* auxiliaryData,simVoid* customData,simInt* replyData)
C parameters	**auxiliaryData**: pointer to 4 integers. auxiliaryData[0] should be a unique identifier different from 0. Use a large, random identifier, or better, your V-REP's serial number if the message is yours. Otherwise, use the identifier of some other module. auxiliaryData[1] could be the messageID of the message you wish to send to another module. auxiliaryData[2] and auxiliaryData[3] can be any values specific to your application. **customData**: customData of your application (the broadcaster is in charge to release that buffer). Can be NULL. **replyData**: pointer to 4 integers that can be used by a module to reply to a broadcasted message. Can be NULL. If not NULL, all 4 values are automatically initialized to -1. Broadcasted messages can be intercepted in a plugin's "v_repMessage"-function. In the function, broadcasted messages can be recognized when the function's first argument ("message") is sim_message_module_broadcast.
C return value	Pointer to custom reply data that can be used by a module to reply to a broadcasted message. The module that replies is in charge of allocating the data with simCreateBuffer and the original broadcaster is in charge of releasing that data with simReleaseBuffer. A reply to a broadcasted message is triggered by a module that writes a value different from -1 into auxiliaryData[0]-auxiliaryData[3], thus aborting further broadcast of the original message and returning to the broadcaster. If the return value is different from NULL, the broadcast is also interrupted.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simBuildIdentityMatrix / sim.buildIdentityMatrix

Description	Builds an identity transformation matrix. See also the other matrix/transformation functions
C synopsis	simInt simBuildIdentityMatrix(simFloat* matrix)
C parameters	**matrix**: pointer to 12 simFloat values (the last row of the 4x4 matrix (0,0,0,1) is not needed) The x-axis of the orientation component is (matrix[0],matrix[4],matrix[8]) The y-axis of the orientation component is (matrix[1],matrix[5],matrix[9]) The z-axis of the orientation component is (matrix[2],matrix[6],matrix[10]) The position component is (matrix[3],matrix[7],matrix[11])
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be

	available
Lua synopsis	table_12 matrix=sim.buildIdentityMatrix()
Lua parameters	None
Lua return values	**matrix**: table containing the identity matrix (except for the last row), or nil in case of an error. Table values in Lua are indexed from 1, not 0!

simBuildMatrix / sim.buildMatrix

Description	Builds a transformation matrix based on a position vector and Euler angles. See also the other matrix/transformation functions.
C synopsis	simInt simBuildMatrix(const simFloat* position,const simFloat* eulerAngles,simFloat* matrix)
C parameters	**position**: pointer to 3 simFloat values representing the position component **eulerAngles**: pointer to 3 simFloat values representing the angular component **matrix**: pointer to 12 simFloat values representing the transformation matrix The x-axis of the orientation component of the matrix is (matrix[0],matrix[4],matrix[8]) The y-axis of the orientation component of the matrix is (matrix[1],matrix[5],matrix[9]) The z-axis of the orientation component of the matrix is (matrix[2],matrix[6],matrix[10]) The position component of the matrix is (matrix[3],matrix[7],matrix[11])
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_12 matrix=sim.buildMatrix(table_3 position,table_3 eulerAngles)
Lua parameters	**position**: table to 3 numbers representing the position component **eulerAngles**: table to 3 numbers representing the angular component
Lua return values	**matrix**: table containing the transformation matrix (except for the last row), or nil in case of an error. Table values in Lua are indexed from 1, not 0!

simBuildMatrixQ / sim.buildMatrixQ

Description	Builds a transformation matrix based on a position vector and a quaternion. See also the other matrix/transformation functions.
C synopsis	simInt simBuildMatrixQ(const simFloat* position,const simFloat* quaternion,simFloat* matrix)
C parameters	**position**: pointer to 3 simFloat values representing the position component **quaternion**: pointer to 4 simFloat values representing the orientation quaternion (x,y,z,w) **matrix**: pointer to 12 simFloat values representing the transformation matrix The x-axis of the orientation component of the matrix is (matrix[0],matrix[4],matrix[8]) The y-axis of the orientation component of the matrix is (matrix[1],matrix[5],matrix[9]) The z-axis of the orientation component of the matrix is (matrix[2],matrix[6],matrix[10]) The position component of the matrix is (matrix[3],matrix[7],matrix[11])
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_12 matrix=sim.buildMatrixQ(table_3 position,table_4 quaternion)
Lua parameters	**position**: table of 3 numbers representing the position component **quaternion**: table of 4 numbers representing the orientation quaternion (x,y,z,w)
Lua return values	**matrix**: table containing the transformation matrix (except for the last row), or nil in case of an error. Table values in Lua are indexed from 1, not 0!

simCallScriptFunction / sim.callScriptFunction (remote API equivalent: simxCallScriptFunction)

Description	Calls a script function (from a plugin, the main client application, or from another script). This represents a callback inside of a script. Call this only: a) from the main thread, or: b) from a thread that originated from a threaded child script. In that case, you cannot call non-threaded child scripts. When calling simulation scripts, then simulation must be running. See also sim.executeScriptString and sim.setScriptVariable.
C synopsis	From C/C++, call simCallScriptFunctionEx instead.
C parameters	-

C return value	-
Lua synopsis	...=sim.callScriptFunction(string functionNameAtScriptName,number scriptHandleOrType,...)
Lua parameters	**functionNameAtScriptName**: a string representing the function name and script name, e.g. *myFunctionName@theScriptName*. When the script is not associated with an object, then just specify the function name. **scriptHandleOrType**: the handle of the script, otherwise the type of the script: *sim.scripttype_mainscript* (0): the main script will be called. *sim.scripttype_childscript* (1): a child script will be called. *sim.scripttype_customizationscript* (6): a customization script will be called. **...**: any number of arguments that will be handed over to the called function.
Lua return values	**...**: any number of return values from the called function.

simCallScriptFunctionEx (remote API equivalent: simxCallScriptFunction)

Description	Calls a script function (from a plugin, the main client application, or from another script). This represents a callback inside of a script. Call this only: a) from the main thread, or: b) from a thread that originated from a threaded child script. In that case, you cannot call non-threaded child scripts. When calling simulation scripts, then simulation must be running. See also sim.executeScriptString and sim.setScriptVariable. Data exchange between a plugin and a script happens via a stack. Reading and writing arguments from/to the stack gives you a maximum of flexibility, and you wil be able to exchange also complex data structures. But it can also be tedious and error prone. Use instead the helper classes located in *programming/common/stack* and *programming/include/stack*: they will greatly simplify the task. Have a look at the example plugins v_repExtSkeletonPlugin and v_repExtSkeletonPluginNG.
C synopsis	simInt simCallScriptFunctionEx(simInt scriptHandleOrType,const simChar* functionNameAtScriptName,simInt stackId)
C parameters	**scriptHandleOrType**: the handle of the script, otherwise the type of the script: *sim_scripttype_mainscript* (0): the main script will be called. *sim_scripttype_childscript* (1): a child script will be called. In that case, *functionNameAtScriptName* should also contain the name of the object associated with the script. *sim_scripttype_customizationscript* (6): a customization script will be called. In that case, *functionNameAtScriptName* should also contain the name of the object associated with the script. **functionNameAtScriptName**: the name of the Lua function to call in the specified script. If *scriptHandleOrType* is *sim_scripttype_childscript*, or *sim_scripttype_customizationscript*, then *functionNameAtScriptName* should also contain the name of the object associated with the script: "functionName@objectName". **stackId**: a stack handle. The stack represents the function's in/out values. See also the available stack functions.
C return value	-1 in case of an error
Lua synopsis	From Lua, call sim.callScriptFunction instead.
Lua parameters	-
Lua return values	-

simCameraFitToView / sim.cameraFitToView

Description	Shifts and adjusts a camera associated with a view to fill the view entirely with the specified objects or models. See also the sim.adjustView and sim.floatingViewAdd functions.
C synopsis	simInt simCameraFitToView(simInt viewHandleOrIndex,simInt objectCount,const simInt* objectHandles,simInt options,simFloat scaling)
C parameters	**viewHandleOrIndex**: the handle of the view (can also be a floating view), or the index of the view. If the camera is not associated with any view, then you can specify the handle of the camera, together with the *sim_handleflag_camera* flag. **objectCount**: number of intems in the objectHandles pointer. Can be 0, in which case the whole visible scene will be filling the view. **objectHandles**: pointer to objectHandles. Only visible objects will be taken into account. Can be NULL, in which case the whole visible scene will be filling the view. **options**: bit-coded:

	bit0 (1): if set, then individual objects will be filling the view. If not set, then models associated with model base objects will also be included bit1 (2): if set, then the view proportions will be 1 by 1, independently on what the view size is **scaling**: scaling factor. Use '1' for normal behaviour.
C return value	-1 if operation was not successful. 0 for a silent error (e.g. when the indicated view doesn't exist anymore), 1 for success
Lua synopsis	number result=sim.cameraFitToView(number viewHandleOrIndex,table objectHandles=nil,simInt options=0,simFloat scaling=1)
Lua parameters	Similar as C-function
Lua return values	Same as C-function

simCheckCollision / sim.checkCollision

Description	Checks whether two entities are colliding. Detection is silent (no visual feedback) compared to sim.handleCollision. Also, the collidable flags of the entities are overridden if the entities are objects. See also sim.readCollision and sim.checkCollisionEx.
C synopsis	simInt simCheckCollision(simInt entity1Handle,simInt entity2Handle)
C parameters	**entity1Handle**: handle of entity 1 (can be an object handle or a collection handle) **entity2Handle**: handle of entity 2 (can be an object handle or a collection handle), or sim_handle_all to check entity1 against all other collidable objects
C return value	-1 in case of an error, 0 or 1 to indicate a collision state
Lua synopsis	number result=sim.checkCollision(number entity1Handle,number entity2Handle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCheckCollisionEx / sim.checkCollisionEx

Description	Checks whether two entities are colliding. This is the extended functionality version of sim.checkCollision, and will return all intersections between the two entities. Detection is silent (no visual feedback) compared to sim.handleCollision. Also, the collidable flags of the entities are overridden if the entities are objects. See also sim.readCollision.
C synopsis	simInt simCheckCollisionEx(simInt entity1Handle,simInt entity2Handle,simFloat** intersectionSegments)
C parameters	**entity1Handle**: handle of entity 1 (can be an object handle or a collection handle) **entity2Handle**: handle of entity 2 (can be an object handle or a collection handle), or sim_handle_all to check entity1 against all other collidable objects **intersectionSegments**: pointer to an array of simFloat values that represent the intersections (segments) between the two entities (pt1(x,y,z), pt2(x,y,z), pt1(x,y,z), etc). This can be NULL. The user should use simReleaseBuffer to delete the returned data. That data is only valid if return value is >0
C return value	-1 in case of an error, otherwise the number of segments returned
Lua synopsis	number result,table intersections=sim.checkCollisionEx(number entity1Handle,number entity2Handle)
Lua parameters	**entity1Handle**: handle of entity 1 (can be an object handle or a collection handle) **entity2Handle**: handle of entity 2 (can be an object handle or a collection handle), or sim.handle_all to check entity1 against all other collidable objects
Lua return values	**result**: -1 for error, otherwise the number of segments returned **intersections**: a table that contains the intersection segments between the two entities.

simCheckDistance / sim.checkDistance

Description	Checks the minimum distance between two entities. Detection is silent (no visual feedback) compared to sim.handleDistance. Also, the measurable flags of the entities are overridden if the entities are objects. See also sim.readDistance.
C synopsis	simInt simCheckDistance(simInt entity1Handle,simInt entity2Handle,simFloat threshold,simFloat* distanceData)
C parameters	**entity1Handle**: handle of entity 1 (can be an object handle or a collection handle) **entity2Handle**: handle of entity 2 (can be an object handle or a collection handle), or sim_handle_all

	to check entity1 against all other measurable objects **threshold**: if distance is bigger than the threshold, the distance is not calculated and return value is 0. If threshold is 0 or negative, then no threshold is used. **distanceData**: distanceData[0]-distanceData[5] represents the distance segment, distanceData[6] is the distance between the entities. This data is valid only if return value is 1
C return value	0 or 1 if operation was successful (1 if distance is smaller than threshold), -1 otherwise
Lua synopsis	number result,table_7 distanceData=sim.checkDistance(number entity1Handle,number entity2Handle,number threshold)
Lua parameters	**entity1Handle**: handle of entity 1 (can be an object handle or a collection handle) **entity2Handle**: handle of entity 2 (can be an object handle or a collection handle), or sim.handle_all to check entity1 against all other measurable objects **threshold**: if distance is bigger than the threshold, the distance is not calculated and result is 0. If threshold is 0 or negative, then no threshold is used.
Lua return values	**result**: 0 or 1 if operation was successful (1 if distance is smaller than threshold), -1 otherwise **distanceData**: distanceData[1]-distanceData[6] represents the distance segment, distanceData[7] is the distance between the entities. distanceData is nil if result is different from 1

simCheckIkGroup / sim.checkIkGroup

Description	Solves a registered IK group, but unlike the sim.handleIkGroup function, sim.checkIkGroup will not apply the calculated joint values, but instead return them in an array. See also sim.handleIkGroup, sim.computeJacobian and sim.generateIkPath.
C synopsis	simInt simCheckIkGroup(simInt ikGroupHandle,simInt jointCnt,const simInt* jointHandles,simFloat* jointValues,const simInt* jointOptions)
C parameters	**ikGroupHandle**: handle of the IK group. Only IK groups that are flagged as explicitely handled will be considered as valid handles for this function. See also simGetIkGroupHandle **jointCnt**: the number of joint handles provided in the jointHandles array. **jointHandles** (input): an array with jointCnt entries, that specifies the joint handles for the joints we wish to retrieve the values calculated by the IK. **jointValues** (output): an array with jointCnt entries, that will receive the IK calculated joint values, as specified by the jointHandles array. **jointOptions**: a bit-coded value corresponding to each specified joint handle. Can also be NULL. Bit 0 (i.e. (1) indicates the corresponding joint is dependent of another joint.
C return value	-1 in case of an error, otherwise an IK calculation result.
Lua synopsis	number ikCalculationResult,table jointValues=sim.checkIkGroup(number ikGroupHandle,table jointHandles,table jointOptions=nil)
Lua parameters	Similar as C-function
Lua return values	Similar as C-function

simCheckOctreePointOccupancy / sim.checkOctreePointOccupancy

Description	Checks whether the provided points collide with the octree voxels. See also sim.insertVoxelsIntoOctree and the other octree related functions.
C synopsis	simInt checkOctreePointOccupancy(simInt octreeHandle,simInt options,const simFloat* pts,simInt ptCnt,simUInt* tag,simUInt64* location,simVoid* reserved)
C parameters	**octreeHandle**: the handle of the octree. See also simGetObjectHandle **options**: bit-coded: bit0 set (1): specified points are relative to the octree reference frame, otherwise they are relative to the world reference frame **pts**: a pointer to the points specified as X/Y/Z coordinates **ptCnt**: the number of points contained in pts **tag**: a pointer to a tag value, receiving the tag value of the voxel that collides with a single point. If several points are tested, then this pointer is ignored. Can be NULL. **location**: a pointer to a uint64 value, which specifies the location of the voxel that collides with a single point. If several points are tested, then this pointer is ignored. Can be NULL. The location value is coded in following way: bit0 - bit5: the depth level of the voxel in the octree structure (1-63). bit6 - bit63: a triple bit-value for each depth level. triple bit-values represent the node location relative to the parent node: 0: (-1,-1,-1) 1: (+1,-1,-1) 2: (-1,+1,-1)

	3: (+1,+1,-1) 4: (-1,-1,+1) 5: (+1,-1,+1) 6: (-1,+1,+1) 7: (+1,+1,+1) as an example, 0xE02 would indicate a depth level 2, with coordinates: (+1,+1,+1)*voxelSize*2+(-1,-1,-1)*voxelSize. **reserved**: reserved for future extensions. Set to NULL
C return value	-1 if operation was not successful, 0 if the points do not collide with the voxels, 1 if the points collide with the voxels
Lua synopsis	number result,number tag,number locationLow,number locationHigh=sim.checkOctreePointOccupancy(number octreeHandle,number options,table points)
Lua parameters	Same as C-function
Lua return values	if a single point is specified and collides with a voxel, 4 values are returned, otherwise only **result** will be returned. The **location** value is divided into a low- and high-value (i.e. first 32 bits and last 32 bits of **location**).

simCheckProximitySensor / sim.checkProximitySensor

Description	Checks whether the proximity sensor detects the indicated entity. Detection is silent (no visual feedback) compared to sim.handleProximitySensor. Also, the detectable flags of the entity are overridden if the entity is an object. See also sim.readProximitySensor and sim.checkProximitySensorEx.
C synopsis	simInt simCheckProximitySensor(simInt sensorHandle,simInt entityHandle,simFloat* detectedPoint)
C parameters	**sensorHandle**: handle of the proximity sensor object **entityHandle**: handle of entity to detect (object or collection), or sim_handle_all to detect all detectable objects **detectedPoint**: coordinates of detected point relative to the sensor origin (detectedPoint[0]-detectedPoint[2]), and distance of detected point to the sensor origin (detectedPoint[3]). Can be NULL
C return value	-1 if operation was not successful, otherwise 0 (no detection) or 1 (detection)
Lua synopsis	number result,number distance,table_3 detectedPoint=sim.checkProximitySensor(number sensorHandle,number entityHandle)
Lua parameters	**sensorHandle**: handle of the proximity sensor object **entityHandle**: handle of entity to detect (object or collection), or sim.handle_all to detect all detectable objects
Lua return values	**result**: -1 (error), 0 (not detected) or 1 (detected) **distance**: distance from the sensor origin to the detected point. Is nil if result is different from 1 **detectedPoint**: position of the detected point relative to the sensor origin. Is nil if result is different from 1

simCheckProximitySensorEx / sim.checkProximitySensorEx

Description	Checks whether the proximity sensor detects the indicated entity. This is the extended functionality version of sim.checkProximitySensor. Detection is silent (no visual feedback) compared to sim.handleProximitySensor. Also, the detectable flags of the entity are overridden if the entity is an object. see also sim.readProximitySensor and sim.checkProximitySensorEx2.
C synopsis	simInt simCheckProximitySensorEx(simInt sensorHandle,simInt entityHandle,simInt detectionMode,simFloat detectionThreshold,simFloat maxAngle,simFloat* detectedPoint,simInt* detectedObjectHandle,simFloat* surfaceNormalVector)
C parameters	**sensorHandle**: handle of the proximity sensor object **entityHandle**: handle of entity to detect (object or collection), or sim_handle_all to detect all detectable objects **detectionMode**: bit coded: bit0 (1) for front face detection, bit1 (2) for back face detection (bit0|bit1 needs to be true), bit2 (4) for fast detection (doesn't search for the closest point, just any point in the detection volume), bit3 (8) for limited angle detection (if set, maxAngle is taken into account), bit4 (16) for occlusion check. **detectionThreshold**: doesn't detect objects farther than detectionThreshold distance from sensor origin **maxAngle**: maximum detection angle (angle between detection ray and normal vector of the surface). Can be (0;pi/2). Only if bit3 of detectionMode is set will this parameter have an effect. Use this to realistically simulate ultrasonic sensors. **detectedPoint**: coordinates of detected point relative to the sensor origin (detectedPoint[0]-detectedPoint[2]), and distance of detected point to the sensor origin (detectedPoint[3]). Can be NULL

	detectedObjectHandle: handle of detected object (useful when entity to be detected is a collection or sim_handle_all). Can be NULL **surfaceNormalVector**: normal vector of the surface where the point was detected. Normalized. Relative to the sensor reference frame. Can be NULL
C return value	-1 if operation was not successful, otherwise 0 (no detection) or 1 (detection)
Lua synopsis	number result,number distance,table_3 detectedPoint,number detectedObjectHandle, table_3 surfaceNormalVector=sim.checkProximitySensorEx(number sensorHandle,number entityHandle,number detectionMode,number detectionthreshold,number maxAngle)
Lua parameters	**sensorHandle**: handle of the proximity sensor object **entityHandle**: handle of entity to detect (object or collection), or sim.handle_all to detect all detectable objects **detectionMode**: bit coded: bit0 (1) for front face detection, bit1 (2) for back face detection (bit0|bit1 needs to be true), bit2 (4) for fast detection (doesn't search for the closest point, just any point in the detection volume), bit3 (8) for limited angle detection (if set, maxAngle is taken into account). **detectionThreshold**: doesn't detect objects farther than detectionThreshold distance from sensor origin **maxAngle**: maximum detection angle (angle between detection ray and normal vector of the surface). Can be (0;pi/2). Only if bit3 of detectionMode is set will this parameter have an effect. Use this to realistically simulate ultrasonic sensors.
Lua return values	**result**: -1 (error), 0 (not detected) or 1 (detected) **distance**: distance from the sensor origin to the detected point. Is nil if result is different from 1 **detectedPoint**: position of the detected point relative to the sensor origin. Is nil if result is different from 1 **detectedObjectHandle**: handle of detected object. Is nil if result is different from 1 **surfaceNormalVector**: normal vector of the surface where the point was detected. Normalized. Relative to the sensor reference frame. Is nil if result is different from 1

simCheckProximitySensorEx2 / sim.checkProximitySensorEx2

Description	Checks whether the proximity sensor detects the indicated points, segments or triangles. Detection is silent (no visual feedback). See also sim.readProximitySensor and sim.checkProximitySensorEx.
C synopsis	simInt simCheckProximitySensorEx2(simInt sensorHandle,simFloat* vertexPointer,simInt itemType,simInt itemCount,simInt detectionMode,simFloat detectionThreshold,simFloat maxAngle,simFloat* detectedPoint,simFloat* normalVector)
C parameters	**sensorHandle**: handle of the proximity sensor object **vertexPointer**: a pointer to vertices **itemType**: 0 for points, 1 for segments and 2 for triangles **itemCount**: the number of items that vertexPointer points at For the other parameters, see the description in simCheckProximitySensorEx. (simCheckProximitySensorEx2 doesn't support occlusion checking)
C return value	-1 if operation was not successful, otherwise 0 (no detection) or 1 (detection)
Lua synopsis	number result,number distance,table_3 detectedPoint,table_3 normalVector=sim.checkProximitySensorEx2(number sensorHandle,table vertices,number itemType,number itemCount,number mode,number threshold,number maxAngle)
Lua parameters	**sensorHandle**: handle of the proximity sensor object **vertices**: a table containing vertices **itemType**: 0 for points, 1 for segments and 2 for triangles **itemCount**: the number of items that the 'vertices' table contains For the other parameters, see the description in sim.checkProximitySensorEx. (sim.checkProximitySensorEx2 doesn't support occlusion checking)
Lua return values	**result**: -1 (error), 0 (not detected) or 1 (detected) For the other return values, see the description in sim.checkProximitySensorEx.

simCheckVisionSensor / sim.checkVisionSensor

Description	Checks whether the vision sensor detects the indicated entity. Detection is silent (no visual feedback) compared to sim.handleVisionSensor. Also, the renderable flag of the entity is overridden if the entity is an object. See also sim.readVisionSensor and sim.checkVisionSensorEx.
C synopsis	simInt simCheckVisionSensor(simInt sensorHandle,simInt entityHandle,simFloat** auxValues,simInt**

	auxValuesCount)

C parameters	**sensorHandle**: handle of the vision sensor object
	entityHandle: handle of entity to detect (object or collection), or sim_handle_all to detect all detectable objects
	auxValues: auxiliary values returned from the applied filters (refer to the filter's documentation for details). By default V-REP returns one packet of 15 auxiliary values:the minimum of intensity, red, green, blue, depth value, the maximum of intensity, red, green, blue, depth value, and the average of intensity, red, green, blue, depth value. If additional filter components return values, then they will be appended as packets to the first packet. AuxValues can be NULL. The user is in charge of releasing the auxValues buffer with simReleaseBuffer(*auxValues).
	auxValuesCount: contains information about the number of auxiliary value packets and packet sizes returned in auxValues. The first value is the number of packets, the second is the size of packet1, the third is the size of packet2, etc. Can be NULL if auxValues is also NULL. The user is in charge of releasing the auxValuesCount buffer with simReleaseBuffer(*auxValuesCount).
	
	Usage example:
	
	`float* auxValues=NULL;`
	`int* auxValuesCount=NULL;`
	`float averageColor[3]={0.0f,0.0f,0.0f};`
	`if (simCheckVisionSensor(sensorHandle,entityHandle,&auxValues,&auxValuesCount)>=0)`
	`{`
	`  if ((auxValuesCount[0]>0)||(auxValuesCount[1]>=15))`
	`  {`
	`  averageColor[0]=auxValues[11];`
	`  averageColor[1]=auxValues[12];`
	`  averageColor[2]=auxValues[13];`
	`  }`
	`  simReleaseBuffer((char*)auxValues);`
	`  simReleaseBuffer((char*)auxValuesCount);`
	`}`

C return value	-1 if operation was not successful, otherwise 0 (no detection) or 1 (detection)
Lua synopsis	number result,table auxiliaryValuePacket1,table auxiliaryValuePacket2, etc.=sim.checkVisionSensor(number sensorHandle,number entityHandle)
Lua parameters	**sensorHandle**: handle of the vision sensor object **entityHandle**: handle of entity to detect (object or collection), or sim.handle_all to detect all detectable objects
Lua return values	**result**: -1 if operation was not successful, otherwise 0 (no detection) or 1 (detection) **auxiliaryValuePacket1**: default auxiliary value packet (same as for the C-function) (table values in Lua are indexed from 1, not 0!) **auxiliaryValuePacket2**: additional auxiliary value packet (e.g. from a filter component) **auxiliaryValuePacket3**: etc. (the function returns as many tables as there are auxiliary value packets)

simCheckVisionSensorEx / sim.checkVisionSensorEx

Description	Checks whether the vision sensor detects the indicated entity. This is the extended functionality version of sim.checkVisionSensor. Detection is silent (no visual feedback) compared to sim.handleVisionSensor. Also, the renderable flag of the entity is overridden if the entity is an object. See also sim.readVisionSensor.
C synopsis	simFloat* simCheckVisionSensorEx(simInt sensorHandle,simInt entityHandle,simBool returnImage)
C parameters	**sensorHandle**: handle of the vision sensor object **entityHandle**: handle of entity to detect (object or collection), or sim_handle_all to detect all detectable objects **returnImage**: specifies what should be returned. If true, the sensor's image buffer is returned, otherwise its depth buffer is returned
C return value	image or depth buffer (use simGetVisionSensorResolution for correct size), or NULL in case of an error. The user is in charge of releasing the returned buffer with simReleaseBuffer
Lua synopsis	table buffer=sim.checkVisionSensorEx(number sensorHandle,number entityHandle,boolean returnImage)
Lua parameters	Same as C-function
Lua return values	Similar to C-function: a table containing the image or depth buffer is returned (or nil in case of an error)

simClearFloatSignal / sim.clearFloatSignal (remote API equivalent: simxClearFloatSignal)

Description	Clears a float signal (removes it). See also the other signal functions.
C synopsis	simInt simClearFloatSignal(const simChar* signalName)
C parameters	**signalName**: name of the signal or NULL to clear all float signals
C return value	-1 if operation was not successful, otherwise the number of signals cleared
Lua synopsis	number result=sim.clearFloatSignal(string signalName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simClearIntegerSignal / sim.clearIntegerSignal (remote API equivalent: simxClearIntegerSignal)

Description	Clears an integer signal (removes it). See also the other signal functions.
C synopsis	simInt simClearIntegerSignal(const simChar* signalName)
C parameters	**signalName**: name of the signal or NULL to clear all integer signals
C return value	-1 if operation was not successful, otherwise the number of signals cleared
Lua synopsis	number result=sim.clearIntegerSignal(string signalName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simClearScriptVariable (DEPRECATED)

Description	DEPRECATED. See sim.setScriptVariable instead.

simClearStringSignal / sim.clearStringSignal (remote API equivalent: simxClearStringSignal)

Description	Clears a string signal (removes it). See also the other signal functions.
C synopsis	simInt simClearStringSignal(const simChar* signalName)
C parameters	**signalName**: name of the signal or NULL to clear all string signals
C return value	-1 if operation was not successful, otherwise the number of signals cleared
Lua synopsis	number result=sim.clearStringSignal(string signalName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCloseModule / sim.closeModule

Description	Releases resources reserved with the sim.openModule command. This command can only be called from the main script. Call it from the main script in the last simulation pass (usually with sim.handle_all argument). sim.closeModule is not available in the C-API. Look at the default main script to get an idea about how to use sim.openModule, sim.handleModule and simCloseModule.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result=sim.closeModule(number sim.handle_all) number result=sim.closeModule(string moduleName)
Lua parameters	**sim.handle_all**: indicates that all plugins should be closed **moduleName**: the name of a specific plugin that should be closed
Lua return values	**result**: -1 in case of an error, otherwise result is the number of plugins that executed the command.

simCloseScene (remote API equivalent: simxCloseScene)

Description	Closes current scene, and switches to another open scene. If there is no other open scene, a new scene is then created. See also sim.loadScene and sim.saveScene.
C synopsis	simInt simCloseScene()
C parameters	none
C return value	-1 if operation was not successful, otherwise the current scene index.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simCombineRgbImages / sim.combineRgbImages

Description	Combines two RGB images. See also sim.transformImage.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	string outImg=sim.combineRgbImages(string img1,table_2 img1Res,string img2,table_2 img2Res,number operation)
Lua parameters	**img1**: input image 1, as succession of rgb values. **img1Res**: x/y resolution of image 1. **img2**: input image 2, as succession of rgb values. **img2Res**: x/y resolution of image 2. **operation**: operation to be performed with the two input images.
Lua return values	output image as succession of rgb values, or nil in case of an error.

simComputeJacobian / sim.computeJacobian

Description	Computes the Jacobian of a registered IK group. The result can then be read via sim.getIkGroupMatrix. See also sim.handleIkGroup and sim.checkIkGroup.
C synopsis	simInt simComputeJacobian(simInt ikGroupHandle,simInt options,simVoid* reserved)
C parameters	**ikGroupHandle**: handle of the IK group. See also simGetIkGroupHandle. **options**: bit-coded: bit0 set (1): the joint IK weights are taken into account. **reserved**: reserved for future extensions. Set to NULL.
C return value	-1 if operation failed
Lua synopsis	number result=sim.computeJacobian(number ikGroupHandle,number options)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simComputeMassAndInertia / sim.computeMassAndInertia

Description	Computes and applies the mass and inertia properties for a convex shape (or convex compound shape), based on a density value. If call this function while the simulation is running, you will have to call sim.resetDynamicObject upon the object, for the changes to take effect. See also sim.getShapeMassAndInertia and sim.convexDecompose.
C synopsis	simInt simComputeMassAndInertia(simInt shapeHandle,simFloat density))
C parameters	**shapeHandle**: handle of shape. The shape must be convex (or a convex compound). **density**: the density expressed in kg/m^3
C return value	-1 in case of an error, 0 if the shape is not convex, otherwise 1.
Lua synopsis	number result=sim.computeMassAndInertia(number shapeHandle,number density)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simConvexDecompose / sim.convexDecompose

Description	Calculates the convex decomposition of a shape using the HACD or V-HACD algorithms. See also sim.getQHull, sim.getDecimatedMesh, sim.ungroupShape and sim.computeMassAndInertia.
C synopsis	simInt simConvexDecompose(simInt shapeHandle,simInt options,const simInt* intParams,const simFloat* floatParams)
C parameters	**shapeHandle**: handle of the shape to operate on **options**: bit-coded: bit0 set (1): the specified shape will be morphed into its convex decomposition. Otherwise, the convex decomposition will smply be added to the scene bit1 set (2): specified convex decomposition parameters will be displayed in a dialog, allowing the user to modify them. bit2 set (4): same convex decomposition parameters will be used as a previous call to this function. Only when this bit is set can the convex decomposition parameters be omitted. bit3 set (8): HACD: extra points will be added when computing the concavity bit4 set (16): HACD: faces points will be added when computing the concavity bit5 set (32): each individual mesh of a compound shape will be handled on its own during decomposition, otherwise the compound shape is considered as a single mesh bit6 (64): reserved. Do not set. bit7 set (128): the V-HACD algorithm â—‹will be used. If not set, the HACD algorithm will be used. bit8 set (256): V-HACD: pca is enabled (default is disabled). bit9 set (512): V-HACD: tetrahedron-based approximate convex decomposition. If not set, then the voxel-based approximate convex decomposition will be used (default). **intParams**: 10 int values: intParams[0]: HACD: the minimum number of clusters to be generated (e.g. 1) intParams[1]: HACD: the targeted number of triangles of the decimated mesh (e.g. 500) intParams[2]: HACD: the maximum number of vertices for each generated convex hull (e.g. 100) intParams[3]: HACD: the maximum number of iterations. Use 0 for the default value (i.e. 4). intParams[4]: reserved. Set to 0. intParams[5]: V-HACD: resolution (10000-64000000, 100000 is default). intParams[6]: V-HACD: depth (1-32, 20 is default). intParams[7]: V-HACD: plane downsampling (1-16, 4 is default). intParams[8]: V-HACD: convex hull downsampling (1-16, 4 is default). intParams[9]: V-HACD: max. number of vertices per convex hull (4-1024, 64 is default). **floatParams**: 10 float values: floatParams[0]: HACD: the maximum allowed concavity (e.g. 100.0) floatParams[1]: HACD: the maximum allowed distance to get convex clusters connected (e.g. 30) floatParams[2]: HACD: the threshold to detect small clusters. The threshold is expressed as a percentage of the total mesh surface (e.g. 0.25) floatParams[3]: reserved. Set to 0.0 floatParams[4]: reserved. Set to 0.0 floatParams[5]: V-HACD: concavity (0.0-1.0, 0.0025 is default). floatParams[6]: V-HACD: alpha (0.0-1.0, 0.05 is default). floatParams[7]: V-HACD: beta (0.0-1.0, 0.05 is default). floatParams[8]: V-HACD: gamma (0.0-1.0, 0.00125 is default). floatParams[9]: V-HACD: min. volume per convex hull (0.0-0.01, 0.0001 is default).
C return value	-1 if operation was not successful. Otherwise the handle of the new shape, or the handle of the original shape when morphing.
Lua synopsis	number shapeHandle=sim.convexDecompose(number shapeHandle,number options,table_4 intParams,table_3 floatParams)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCopyMatrix / sim.copyMatrix

Description	Copies a transformation matrix. See also the other matrix/transformation functions.
C synopsis	simInt simCopyMatrix(const simFloat* matrixIn,simFloat* matrixOut)
C parameters	**matrixIn**: matrix to be copied **matrixOut**: copy of matrixIn (after the call) matrixIn and matrixOut are pointers to 12 simFloat values (the last row of the 4x4 matrix (0,0,0,1) is

	not needed) The x-axis of the orientation component is (matrix[0],matrix[4],matrix[8]) The y-axis of the orientation component is (matrix[1],matrix[5],matrix[9]) The z-axis of the orientation component is (matrix[2],matrix[6],matrix[10]) The position component is (matrix[3],matrix[7],matrix[11])
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_12 matrixOut=sim.copyMatrix(table_12 matrixIn)
Lua parameters	**matrixIn**: matrix to be copied
Lua return values	**matrixOut**: copied matrix, or nil in case of an error

simCopyPasteObjects / sim.copyPasteObjects (remote API equivalent: simxCopyPasteObjects)

Description	Copies and pastes objects, together with all their associated calculation objects and associated scripts. See also sim.removeObject and sim.removeModel.
C synopsis	simInt simCopyPasteObjects(simInt* objectHandles,simInt objectCount,simInt options)
C parameters	**objectHandles**: array containing the handles of the objects to copy and paste. The same array will receive the copied object handles. **objectCount**: the number of handles contained in the objectHandles array. **options**: bit-coded. If bit0 is set (i.e. 1), then whole models will be copied. In that case, all specified objects should be flagged as *model base*..
C return value	-1 if operation was not successful, otherwise the number of handles returned in the objectHandles array.
Lua synopsis	table copiedObjectHandles=sim.copyPasteObjects(table objectHandles,number options)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCopyPasteSelectedObjects (DEPRECATED)

Description	DEPRECATED. See sim.copyPasteObjects instead.

simCopyStack

Description	Duplicates a stack object. See also the other stack functions.
C synopsis	simInt simCopyStack(simInt stackHandle)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack.
C return value	-1 in case of an error, otherwise the handle of the duplicated stack.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simCreateBuffer (remote API equivalent: simxCreateBuffer)

Description	Creates a buffer. The buffer needs to be released with simReleaseBuffer except otherwise explicitly specified.
C synopsis	simChar* simCreateBuffer(simInt size)
C parameters	**size**: size of the buffer
C return value	buffer if operation was successful, NULL otherwise
Lua synopsis	-
Lua parameters	-
Lua return values	-

simCreateCollection / sim.createCollection

Description	Creates a new collection. See also sim.removeCollection and sim.addObjectToCollection.
C synopsis	simInt simCreateCollection(const simChar* collectionName,simInt options)
C parameters	**collectionName**: the name of the collection. Use an empty string for a default name. **options**: bit-coded options: bit 0 set (1): collection overrides collidable, measurable, renderable, cuttable and detectable properties.
C return value	-1 if operation was not successful, otherwise the handle of the new collection.
Lua synopsis	number collectionHandle=sim.createCollection(string collectionName,number options)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateDummy / sim.createDummy

Description	Creates a dummy.
C synopsis	simInt simCreateDummy(simFloat size,const simFloat* color)
C parameters	**size**: the dummy size **color**: pointer to 4x3 values representing the dummy color (ambient/diffuse rgb, 3 reserved values (set to zero), specular rgb and emission rgb). Can be NULL for default values
C return value	-1 if operation was not successful, otherwise the handle of the dummy
Lua synopsis	number dummyHandle=sim.createDummy(number size,table_12 color=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateForceSensor / sim.createForceSensor

Description	Creates a force sensor.
C synopsis	simInt simCreateForceSensor(simInt options,const simInt* intParams,const simFloat* floatParams,const float* color)
C parameters	**options**: bit-coded options: bit 0 set (1): force threshold enabled bit 1 set (2): torque threshold enabled **intParams (input)**: 5 integer parameters: intParams[0]: filter type (0=average, 1=median) intParams[1]: value count the filter operates on intParams[2]: number of consecutive threshold violation for the sensor to break intParams[3]: reserved. Set to 0 intParams[4]: reserved. Set to 0 **floatParams (input)**: 5 floating point parameters: floatParams[0]: sensor size floatParams[1]: force threshold value floatParams[2]: torque threshold value floatParams[3]: reserved. Set to 0.0 floatParams[4]: reserved. Set to 0.0 **color**: pointer to 2x4x3 values representing the various colors of the sensor ((part1, part2) x (ambient_diffuse rgb, 3 reserved values (set to zero), specular rgb and emission rgb)). Can be NULL for default values
C return value	-1 if operation was not successful, otherwise the handle of the force sensor
Lua synopsis	number sensorHandle=sim.createForceSensor(number options,table_5 intParams,table_5 floatParams,table_24 color=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateHeightfieldShape / sim.createHeightfieldShape

Description	Creates a heightfield shape. See also sim.createPureShape, sim.createMeshShape and sim.addParticleObject.
C synopsis	simInt simCreateHeightfieldShape(simInt options,simFloat shadingAngle,simInt xPointCount,simInt yPointCount,simFloat xSize,const simFloat* heights)
C parameters	**options**: bit-coded options: bit 0 set (1): back faces are culled bit 1 set (2): overlay mesh is visible bit 2 set (4): a simple shape is generated instead of a heightfield bit 3 set (8): the heightfield is not respondable **shadingAngle**: the shading angle **xPointCount/yPointCount**: the number of rows and lines of the heightfield. **xSize**: the length of the x side of the heightfield **heights**: a pointer to xPointCount*yPointCount height values.
C return value	-1 if operation was not successful, otherwise the handle of the newly created shape
Lua synopsis	number objectHandle=sim.createHeightfieldShape(number options,number shadingAngle,number xPointCount,number yPointCount,number xSize,table heights)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateIkElement / sim.createIkElement

Description	Creates an IK element. See also sim.createIkGroup.
C synopsis	simInt simCreateIkElement(simInt ikGroupHandle,simInt options,const simInt* intParams,const simFloat* floatParams,const simVoid* reserved)
C parameters	**ikGroupHandle**: the handle to an IK group which will contain this IK element. **options**: bit-coded options: bit 0 set (1): the element is inactive **intParams**: an array of 4 integer parameters: intParams[0]: the handle of the tip dummy. intParams[1]: the handle of the base object, or -1 for none (i.e. world). intParams[2]: the handle of an object that will represent an alternative base for constraint evaluation, or -1 if the the constraints should be evaluated relative the the base object. intParams[3]: the IK constraints. **floatParams**: an optional array of 4 float parameters (i.e. array can be NULL): floatParams[0]: the linear precision. floatParams[1]: the angular precision. floatParams[2]: the position weight. floatParams[3]: the orientation weight. **reserved**: reserved. Set to NULL.
C return value	-1 if operation was not successful.
Lua synopsis	number result=sim.createIkElement(number ikGroupHandle,number options,table intParams,table floatParams=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateIkGroup / sim.createIkGroup

Description	Creates an IK group. See also sim.removeIkGroup and sim.createIkElement.
C synopsis	simInt simCreateIkGroup(simInt options,const simInt* intParams,const simFloat* floatParams,const simVoid* reserved)
C parameters	**options**: bit-coded options: bit 0 set (1): the group is inactive. bit 1 set (2): joint limits are taken into account during calculation (i.e. only for redundant kinematics). bit 2 set (4): restore if position not reached. bit 3 set (8): restore if orientation not reached. bit 4 set (16): do not ignore the joint's max. step sizes. bit 5 set (32): the group is explicitely handled. **intParams**: an optional array of 2 integer parameters (i.e. array can be NULL):

	intParams[0]: the IK calculation method. intParams[1]: the maximum number of iterations. **floatParams**: an optional array of 4 float parameters (i.e. array can be NULL): floatParams[0]: the DLS factor. floatParams[1]: the joint limit weight. floatParams[2]: the prismatic joint limit threshold. floatParams[3]: the revolute joint limit threshold. **reserved**: reserved. Set to NULL.
C return value	-1 if operation was not successful, otherwise the IK group handle.
Lua synopsis	number ikGroupHandle=sim.createIkGroup(number options,table intParams=nil,table floatParams=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateJoint / sim.createJoint

Description	Creates a joint. See also sim.setJointInterval.
C synopsis	simInt simCreateJoint(simInt jointType,simInt jointMode,simInt options,const simFloat* sizes,const simFloat* colorA,const simFloat* colorB)
C parameters	**jointType**: sim_joint_revolute_subtype, sim_joint_prismatic_subtype or sim_joint_spherical_subtype **jointMode**: a joint mode value **options**: bit-coded. For now only bit 0 is used (if set (1), the joint operates in hybrid mode) **sizes**: pointer to 2 values indicating the joint length and diameter. Can be NULL for default values **colorA**: pointer to 4x3 values for joint color A (ambient_diffuse rgb, 3 reserved values (set to zero), specular rgb and emission rgb). Can be NULL for default values **colorB**: pointer to 4x3 values for joint color B (ambient_diffuse rgb, 3 reserved values (set to zero), specular rgb and emission rgb). Can be NULL for default values
C return value	-1 if operation was not successful, otherwise the handle of the joint
Lua synopsis	number jointHandle=sim.createJoint(number jointType,number jointMode,number options,table_2 sizes=nil,table_12 colorA=nil,table_12 colorB=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateMeshShape / sim.createMeshShape

Description	Creates a mesh shape. See also sim.createPureShape, sim.createHeightfieldShape and sim.getShapeMesh, and see sim.importMesh for a usage example.
C synopsis	simInt simCreateMeshShape(simInt options,simFloat shadingAngle,const simFloat* vertices,simInt verticesSize,const simInt* indices,simInt indicesSize,simFloat* reserved)
C parameters	**options**: Bit-coded: if bit0 is set (1), backfaces are culled. If bit1 is set (2), edges are visible **shadingAngle**: the shading angle **vertices**: an array of vertices **verticesSize**: the size of the vertice array **indices**: an array of indices **indicesSize**: the size of the indice array **reserved**: reserved for future extensions. Keep at NULL.
C return value	-1 if operation was not successful, otherwise the handle of the newly created shape
Lua synopsis	number objectHandle=sim.createMeshShape(number options,number shadingAngle,table vertices,table indices)
Lua parameters	**options**: Bit-coded: if bit0 is set (1), backfaces are culled. If bit1 is set (2), edges are visible **shadingAngle**: the shading angle **vertices**: a table of vertices **indices**: a table of indices
Lua return values	Same as C-function

simCreateMotionPlanning (DEPRECATED)

Description	DEPRECATED. See the OMPL library based path/motion planning functionality instead.

	Creates a motion planning task. See also simRemoveMotionPlanning.
C synopsis	simInt simCreateMotionPlanning(simInt jointCnt,const simInt* jointHandles,const simInt* jointRangeSubdivisions,const simFloat* jointMetricWeights,simInt options,const simInt* intParams,const simFloat* floatParams,const simVoid* reserved)
C parameters	**jointCnt**: the number of joint handles that are submitted in *jointHandles*. **jointHandles**: an array containing *jointCnt* joint handles. **jointRangeSubdivisions**: an array containing *jointCnt* joint range subdivisions. Can be NULL for default values. **jointMetricWeights**: an array containing *jointCnt* joint metric weights. Can be NULL for default values. **options**: bit-coded options. Not used for now (set to zero). **intParams**: an optional array of 5 integer parameters (i.e. array can be NULL): intParams[0]: the handle of the associated IK group. intParams[1]: the self-collision entity 1. intParams[2]: the self-collision entity 2. intParams[3]: the robot-environment collision entity 1. intParams[4]: the robot-environment collision entity 2. **floatParams**: an optional array of 6 float parameters (i.e. array can be NULL): floatParams[0]: the self-collision distance threshold (set to 0.0 for collision detection). floatParams[1]: the robot-environment collision distance threshold (set to 0.0 for collision detection). floatParams[2]-floatParams[5]: the Cartesian space metric for X, Y, Z and (alpha-beta-gamma). **reserved**: reserved. Set to NULL.
C return value	-1 if operation was not successful, otherwise the motion planning task handle.
Lua synopsis	number motionPlanningHandle=simCreateMotionPlanning(table jointHandles,table jointRangeSubdivisions=nil,table jointMetricWeights=nil,number options,table intParams,table floatParams=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateOctree / sim.createOctree

Description	Creates an empty octree. See also sim.removeObject and the other octree related functions.
C synopsis	simInt simCreateOctree(simFloat voxelSize,simInt options,simFloat pointSize,simVoid* reserved)
C parameters	**voxelSize**: the size of the voxels **options**: bit-coded: bit0 set (1): voxels have random colors bit1 set (2): show octree structure bit2 set (4): show points instead of voxels bit3 set (8): reserved. keep unset bit4 set (16): color is emissive **pointSize**: the size of the points in pixels, when voxels are rendered with points **reserved**: reserved for future extensions. Set to NULL
C return value	-1 if operation was not successful, otherwise the handle of the octree
Lua synopsis	number handle=sim.createOctree(number voxelSize,number options,number pointSize)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreatePath / sim.createPath

Description	Creates a path object. See also sim.insertPathCtrlPoints and sim.cutPathCtrlPoints.
C synopsis	simInt simCreatePath(simInt attributes,const simInt* intParams,const simFloat* floatParams,const simFloat* color)
C parameters	**attributes**: a combination of path properties, or -1 for default attributes **intParams (input)**: NULL for default values, or 3 integer values: intParams[0]: line size of the path intParams[1]: the path length calculation method intParams[2]: reserved. Set to 0 **floatParams (input)**: NULL for default values, or 3 float values:

	floatParams[0]: control point size floatParams[1]: the angular to linear conversion coefficient floatParams[2]: the virtual distance scaling factor **color (input)**: pointer to 4x3 values representing the colors of the path (ambient_diffuse rgb, 3 reserved values (set to zero), specular rgb and emission rgb). Can be NULL for default values
C return value	-1 if operation was not successful, otherwise the handle of the newly created path
Lua synopsis	number pathHandle=sim.createPath(number attributes,table_3 intParams=nil,table_3 floatParams=nil,table_12 color=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreatePointCloud / sim.createPointCloud

Description	Creates an empty point cloud. See also sim.removeObject, sim.setPointCloudOptions and the other point cloud related functions.
C synopsis	simInt simCreatePointCloud(simFloat maxVoxelSize,simInt maxPtCntPerVoxel,simInt options,simFloat pointSize,simVoid* reserved)
C parameters	**maxVoxelSize**: the maximum size of the octree voxels containing points **maxPtCntPerVoxel**: the maximum number of points allowed in a same octree voxel **options**: bit-coded: bit0 set (1): points have random colors bit1 set (2): show octree structure bit2 set (4): reserved. keep unset bit3 set (8): do not use an octree structure. When enabled, point cloud operations are limited, and point clouds will not be collidable, measurable or detectable anymore, but adding points will be much faster bit4 set (16): color is emissive **pointSize**: the size of the points, in pixels **reserved**: reserved for future extensions. Set to NULL
C return value	-1 if operation was not successful, otherwise the handle of the point cloud
Lua synopsis	number handle=sim.createPointCloud(number maxVoxelSize,number maxPtCntPerVoxel,number options,number pointSize)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateProximitySensor / sim.createProximitySensor

Description	Creates a proximity sensor.
C synopsis	simInt simCreateProximitySensor(simInt sensorType,simInt subType,simInt options,const simInt* intParams,const simFloat* floatParams,const simFloat* color)
C parameters	**sensorType**: the desired proximity sensor type (e.g. sim_proximitysensor_cone_subtype) **subType**: the desired proximity sensor sub-type (e.g. sim_objectspecialproperty_detectable_ultrasonic) **options**: bit-coded options: bit 0 set (1): the sensor will be explicitely handled bit 1 set (2): the detection volumes are not shown when detecting something bit 2 set (4): the detection volumes are not shown when not detecting anything bit 3 set (8): front faces are not detected bit 4 set (16): back faces are not detected bit 5 set (32): fast detection (i.e. not exact detection) bit 6 set (64): the normal of the detected surface with the detection ray will have to lie below a specified threshold angle bit 7 set (128): occlusion check is active bit 8 set (256): smallest distance threshold will be active bit 9 set (512): randomized detection (only with ray-type proximity sensors) **intParams (input)**: 8 integer parameters: intParams[0]: face count (volume description) intParams[1]: face count far (volume description) intParams[2]: subdivisions (volume description) intParams[3]: subdivisions far (volume description) intParams[4]: randomized detection, sample count per reading intParams[5]: randomized detection, individual ray detection count for triggering intParams[6]: reserved. Set to 0

	intParams[7]: reserved. Set to 0 **floatParams (input)**: 15 floating point parameters: floatParams[0]: offset (volume description) floatParams[1]: range (volume description) floatParams[2]: x size (volume description) floatParams[3]: y size (volume description) floatParams[4]: x size far (volume description) floatParams[5]: y size far (volume description) floatParams[6]: inside gap (volume description) floatParams[7]: radius (volume description) floatParams[8]: radius far (volume description) floatParams[9]: angle (volume description) floatParams[10]: threshold angle for limited angle detection (see bit 6 above) floatParams[11]: smallest detection distance (see bit 8 above) floatParams[12]: sensing point size floatParams[13]: reserved. Set to 0.0 floatParams[14]: reserved. Set to 0.0 **color**: pointer to 4x4x3 values representing the various colors of the sensor ((passive, active, ray, closest distance) x (ambient_diffuse rgb, 3 reserved values (set to zer0), specular rgb and emission rgb)). Can be NULL for default values
C return value	-1 if operation was not successful, otherwise the handle of the force sensor
Lua synopsis	number sensorHandle=sim.createProximitySensor(number sensorType,number subType,number options,table_8 intParams,table_15 floatParams,table_48 color=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreatePureShape / sim.createPureShape

Description	Creates a pure primitive shape. See also sim.createMeshShape, sim.createHeightfieldShape and sim.addParticleObject.
C synopsis	simInt simCreatePureShape(simInt primitiveType,simInt options,const simFloat* sizes,simFloat mass,const simInt* precision)
C parameters	**primitiveType**: 0 for a cuboid, 1 for a sphere, 2 for a cylinder and 3 for a cone **options**: Bit-coded: if bit0 is set (1), backfaces are culled. If bit1 is set (2), edges are visible. If bit2 is set (4), the shape appears smooth. If bit3 is set (8), the shape is respondable. If bit4 is set (16), the shape is static. If bit5 is set (32), the cylinder has open ends **sizes**: 3 values indicating the size of the shape **mass**: the mass of the shape **precision**: 2 values that allow specifying the number of sides and faces of a cylinder or sphere. Can be NULL for default values
C return value	-1 if operation was not successful, otherwise the handle of the newly created shape
Lua synopsis	number objectHandle=sim.createPureShape(number primitiveType,number options,table_3 sizes,number mass,table_2 precision=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateStack

Description	Creates a stack object that is mainly used to exchange complex data in an easy way with scripts. See also simReleaseStack and the other stack functions.
C synopsis	simInt simCreateStack()
C parameters	-
C return value	-1 in case of an error, otherwise a stack handle.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simCreateTexture / sim.createTexture

Description	Creates a planar shape, that will be textured with a new, or imported texture. See also sim.getTextureId, sim.readTexture, sim.writeTexture and sim.setShapeTexture.
C synopsis	simInt simCreateTexture(const simChar* fileName,simInt options,const simFloat* planeSizes,const simFloat* scalingUV,const simFloat* xy_g,simInt fixedResolution,simInt* textureId,simInt* resolution,const simVoid* reserved)
C parameters	**fileName**: the filename of the texure to import, or an empty string if you wish to create a new texture. **options**: bit-coded: **bit0** set (1) =do not interpolate adjacent color patches. **bit1** set (2) =apply the texture in decal-mode. **bit2** set (4) =repeat the texture along the U direction. **bit3** set (8) =repeat the texture along the V direction. **planeSizes**: a pointer to 2 values: the dimensions of the planar shape that will be generated. Can be NULL for default dimensions. **scalingUV**: a pointer to 2 values: the desired scaling of the texture, along the U and V directions. Can be NULL for default scalings. **xy_g**: a pointer to 3 values: the texture x/y shift, and the texture gamma-rotation. Can be NULL for default shift/rotation values. **fixedResolution**: 0 to import the shape with its original resolution. Otherwise, specify a power of 2 value which will be the maximum texture resolution (the texture will also be applied a power of 2 resolution). **resolution**: a pointer to 2 values representing the desired texture resolution when creating a new texture. The same pointer is used to return the effective resolution of the created/imported texture. **textureId**: a pointer to an integer that will be used to return the new texture ID. If a same texture is already present, the old texture ID will be returned. Can be NULL. **reserved**: reserved. Set to NULL.
C return value	-1 in case of an error, otherwise the object handle of the created planar shape.
Lua synopsis	number shapeHandle,number textureId,table_2 resolution=sim.createTexture(string fileName,number options,table_2 planeSizes=nil,table_2 scalingUV=nil,table_2 xy_g=nil,number fixedResolution=0,table_2 resolution=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateUI (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simCreateUIButton (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simCreateUIButtonArray (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simCreateVisionSensor / sim.createVisionSensor

Description	Creates a vision sensor.
C synopsis	simInt simCreateVisionSensor(simInt options,const simInt* intParams,const simFloat* floatParams,const simFloat* color)
C parameters	**options**: bit-coded options: bit 0 set (1): the sensor will be explicitely handled bit 1 set (2): the sensor will be in perspective operation mode bit 2 set (4): the sensor volume will not be shown when not detecting anything

	bit 3 set (8): the sensor volume will not be shown when detecting something bit 4 set (16): the sensor will be passive (use an external image) bit 5 set (32): the sensor will use local lights bit 6 set (64): the sensor will not render any fog bit 7 set (128): the sensor will use a specific color for default background (i.e. "null" pixels) **intParams (input)**: 4 integer parameters: intParams[0]: sensor resolution x intParams[1]: sensor resolution y intParams[2]: reserved. Set to 0 intParams[3]: reserved. Set to 0 **floatParams (input)**: 11 floating point parameters: floatParams[0]: near clipping plane floatParams[1]: far clipping plane floatParams[2]: view angle / ortho view size floatParams[3]: sensor size x floatParams[4]: sensor size y floatParams[5]: sensor size z floatParams[6]: "null" pixel red-value floatParams[7]: "null" pixel green-value floatParams[8]: "null" pixel blue-value floatParams[9]: reserved. Set to 0.0 floatParams[10]: reserved. Set to 0.0 **color**: pointer to 4x4x3 values representing the various colors of the sensor ((sensor passive, sensor active) x (ambient_diffuse rgb, 3 reserved values (set to zero), specular rgb and emission rgb)). Set the last 24 values to zero. Can be NULL for default values
C return value	-1 if operation was not successful, otherwise the handle of the force sensor
Lua synopsis	number sensorHandle=sim.createVisionSensor(number options,table_4 intParams,table_11 floatParams,table_48 color=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCutPathCtrlPoints / sim.cutPathCtrlPoints

Description	Removes one or several control points from a path object. See also sim.insertPathCtrlPoints and sim.createPath.
C synopsis	simInt simCutPathCtrlPoints(simInt pathHandle,simInt startIndex,simInt ptCnt)
C parameters	**pathHandle**: the handle of the path. Refer also to simGetObjectHandle. **startIndex**: the zero-based index of the first control point to remove, or -1 to remove all the control points. **ptCnt**: the number of control points to remove.
C return value	-1 if operation was not successful.
Lua synopsis	number result=sim.cutPathCtrlPoints(number pathHandle,number startIndex,number ptCnt)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simDebugStack

Description	Prints the content of the specified stack to the console. See also the other stack functions.
C synopsis	simInt simDebugStack(simInt stackHandle,simInt cIndex)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **cIndex**: the zero-based index of the stack location to print, or -1 to print the full stack.
C return value	-1 in case of an error.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simDelegateChildScriptExecution (DEPRECATED)

Description	DEPRECATED. Has no effect.

simDeleteSelectedObjects (DEPRECATED)

Description	DEPRECATED. See sim.removeObject instead.

simDeleteUIButtonArray (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simDisplayDialog / sim.displayDialog (remote API equivalent: simxDisplayDialog)

Description	Displays a generic dialog box. Use in conjunction with sim.getDialogResult ,sim.getDialogInput and sim.endDialog. From C, the function will only create non-modal dialogs (non-blocking), from Lua, modal dialogs can be created if called from a child script that runs in a thread. Use custom user interfaces instead if a higher customization level is required. Dialogs displayed from a main script or a child script will automatically close at simulation end. See also sim.msgBox and sim.fileDialog.
C synopsis	simInt simDisplayDialog(const simChar* titleText,const simChar* mainText,simInt dialogType,const simChar* initialText,const simFloat* titleColors,const simFloat* dialogColors,simInt* uiHandle)
C parameters	**titleText**: Title bar text **mainText**: Information text **dialogType**: generic dialog style **initialText**: Initial text in the edit box if the dialog is of type sim_dlgstyle_input. Can be NULL **titleColors**: Title bar color (6 simFloat values for RGB for background and foreground), can be NULL for default colors **dialogColors**: Dialog color (6 simFloat values for RGB for background and foreground), can be NULL for default colors **uiHandle**: corresponding OpenGl-based custom UI handle. Can be NULL
C return value	handle of generic dialog (different from OpenGl-based custom UI handle!!) if operation was successful, -1 otherwise. The handle should be used with following functions: simGetDialogResult ,simGetDialogInput and simEndDialog.
Lua synopsis	number dialogHandle,number uiHandle=sim.displayDialog(string titleText,string mainText,number dialogType,boolean modalDialog,string initialText,table_6 titleColors,table_6 dialogColors,number uiHandle)
Lua parameters	**titleText**: Title bar text **mainText**: information text **dialogType**: generic dialog style **modalDialog**: specifies whether the dialog is modal. Modal dialogs are only allowed when not called from the main thread. **initialText**: Initial text in the edit box if the dialog is of type sim_dlgstyle_input. Can be nil or omitted **titleColors**: Title bar color (6 values for RGB for background and foreground), can be nil for default colors, or omitted **dialogColors**: Dialog color (6 values for RGB for background and foreground), can be nil for default colors, or omitted
Lua return values	**dialogHandle**: handle of generic dialog (different from OpenGl-based custom UI handle!!), or nil if operation failed **uiHandle**: handle of corresponding OpenGl-based custom UI, or nil if operation failed

simDoesFileExist

Description	Indicates whether a file exists.
C synopsis	simInt simDoesFileExist(const simChar* filename)
C parameters	**filename**: The filename extension is required
C return value	1 if the filename exists, 0 if it does not exist, or -1 in case of an error

Lua synopsis	-
Lua parameters	-
Lua return values	-

simEmptyCollection / sim.emptyCollection

Description	Clears a collection from all the objects it contains. An empty collection will not survive, unless you add some objects to it again with sim.addObjectToCollection right after. See also sim.removeCollection.
C synopsis	simInt simEmptyCollection(simInt collectionHandle)
C parameters	**collectionHandle**: handle of the collection to clear. sim_handle_all clears all collections from their obects.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.emptyCollection(number collectionHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simEnableEventCallback

Description	Enables or disables a specific event callback, on a plugin base. Some event callbacks might be called very frequently, and are not enabled by default, in order not to slow down V-REP. A plugin may enable one or several of such event callbacks, in which case only that plugin will receive the event callback notifications. If a given plugin registers twice the same event callback, it will be disabled again.
C synopsis	simInt simEnableEventCallback(simInt eventCallbackType,const simChar* plugin,simInt reserved)
C parameters	**eventCallbackType**: one of following values: sim_message_eventcallback_renderingpass, sim_message_eventcallback_opengl, sim_message_eventcallback_openglframe or sim_message_eventcallback_openglcameraview. **plugin**: the case-sensitive name of the plugin that wishes to receive the notifications. For the plugin "v_repExtMyPlugin", specify "MyPlugin". **reserved**: reserved. Set to -1.
C return value	-1 in case of an error. Otherwise 1 if the callback is enabled, or 0 if it is disabled.
Lua synopsis	
Lua parameters	-
Lua return values	-

simEnableWorkThreads (DEPRECATED)

Description	DEPRECATED. Has no effect.

simEndDialog / sim.endDialog (remote API equivalent: simxEndDialog

Description	Closes and releases resource from a previous call to sim.displayDialog. Even if the dialog is not visible anymore, you should release resources by using this function (however at the end of a simulation, all dialog resources allocated from a main script or a child script are automatically released).
C synopsis	simInt simEndDialog(simInt genericDialogHandle)
C parameters	**genericDialogHandle**: handle of generic dialog (return value of simDisplayDialog)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.endDialog(number genericDialogHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simExecuteScriptString / sim.executeScriptString

Description	Executes some Lua code in a specific script (from a plugin, the main client application, or from another script). Call this only: a) from the main thread, or: b) from a thread that originated from a threaded child script. In that case, you cannot call non-threaded child scripts. When calling simulation scripts, then simulation must be running. See also sim.callScriptFunction. Data exchange between a plugin and a script happens via a stack. Reading and writing arguments from/to the stack gives you a maximum of flexibility, and you wil be able to exchange also complex data structures. But it can also be tedious and error prone. Use instead the helper classes located in *programming/common/stack* and *programming/include/stack*: they will greatly simplify the task. Have a look at the example plugins v_repExtSkeletonPlugin and v_repExtSkeletonPluginNG.
C synopsis	simInt simExecuteScriptString(simInt scriptHandleOrType,const simChar* stringAtScriptName,simInt stackId)
C parameters	**scriptHandleOrType**: the handle of the script, otherwise the type of the script: *sim_scripttype_mainscript* (0): the main script. *sim_scripttype_childscript* (1): a child script. In that case, *stringAtScriptName* should also contain the name of the object associated with the script. *sim_scripttype_customizationscript* (6): a customization script. In that case, *stringAtScriptName* should also contain the name of the object associated with the script. *sim_scripttype_sandboxscript* (8): the sandbox script. **stringAtScriptName**: some Lua code to execute in the specified script. If *scriptHandleOrType* is *sim_scripttype_childscript*, or *sim_scripttype_customizationscript*, then *stringAtScriptName* should also contain the name of the object associated with the script: "string@objectName". **stackId**: 0 (for no stack) or a stack handle. The stack holds possible out values. See also the available stack functions.
C return value	-1 in case of an error
Lua synopsis	number result,executionResult=sim.executeScriptString(string stringAtScriptName,number scriptHandleOrType)
Lua parameters	Similar to the C-function
Lua return values	**result**: -1 in case of an error **executionResult**: return value of the executed code

simExportIk / sim.exportIk

Description	Exports the IK content of a scene. The generated file can be used with the external IK, to do IK computations in an external application.
C synopsis	simInt simExportIk(const simChar* pathAndFilename,simInt reserved1,simVoid* reserved2)
C parameters	**pathAndFilename**: the location and the name of the file to create. **reserved1**: reserved argument, keep at 0. **reserved2**: reserved argument, keep at NULL.
C return value	-1 or 0 if operation was not successful.
Lua synopsis	number result=sim.exportIk(string pathAndFilename)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simExportMesh / sim.exportMesh

Description	Exports a mesh to a file. See also sim.importMesh and sim.getShapeMesh
C synopsis	simInt simExportMesh(simInt fileformat,const simChar* pathAndFilename,simInt options,simFloat scalingFactor,simInt elementCount,simFloat** vertices,const simInt* verticesSizes,simInt** indices,const simInt* indicesSizes,simFloat** reserved,simChar** names)
C parameters	**fileformat**: the fileformat to export to. 0 for OBJ format, 1 for DXF format and 4 for BINARY STL format **pathAndFilename**: the location of the file to create. **options**: keep at 0

scalingFactor: the scaling factor to apply to the vertices to export
vertices: an array to vertice arrays. See the example below
verticesSizes: an array indicating the individual vertice array sizes. See the example below
indices: an array to indice arrays. See the example below
indicesSizes: an array indicating the individual indice array sizes. See the example below
reserved: reserved for future extensions. Keep at NULL.
names: an array to mesh names. See the example below

USAGE EXAMPLE:

```
// Exports all shapes in the scene
simInt shapeCount=0;
while (simGetObjects(shapeCount++,sim_object_shape_type)!=-1);
shapeCount--;
simFloat** vertices=new simFloat*[shapeCount];
simInt* verticesSizes=new simInt[shapeCount];
simInt** indices=new simInt*[shapeCount];
simInt* indicesSizes=new simInt[shapeCount];
simChar** names=new simChar*[shapeCount];
simInt index=0;
while (true)
{
    simInt shapeHandle=simGetObjects(index++,sim_object_shape_type);
    if (shapeHandle<0)
        break;
    simFloat* vert;
    simInt vertS;
    simInt* ind;
    simInt indS;
    simGetShapeMesh(shapeHandle,&vert,&vertS,&ind,&indS,NULL);
    vertices[index-1]=vert;
    verticesSizes[index-1]=vertS;
    indices[index-1]=ind;
    indicesSizes[index-1]=indS;
    names[index-1]=simGetObjectName(shapeHandle);
    simFloat m[12];
    simGetObjectMatrix(shapeHandle,-1,m);
    for (simInt i=0;i<vertS/3;i++)
    {
        simFloat v[3]={vert[3*i+0],vert[3*i+1],vert[3*i+2]};
        simTransformVector(m,v);
        vert[3*i+0]=v[0];
        vert[3*i+1]=v[1];
        vert[3*i+2]=v[2];
    }
}
simExportMesh(1,"d:\\example.dxf",0,1,shapeCount,vertices,
                verticesSizes,indices,indicesSizes,NULL,names);
for (simInt i=0;i<shapeCount;i++)
{
    simReleaseBuffer((simChar*)vertices[i]);
    simReleaseBuffer((simChar*)indices[i]);
    simReleaseBuffer(names[i]);
}
delete[] vertices;
delete[] verticesSizes;
delete[] indices;
delete[] indicesSizes;
delete[] names;
```

C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.exportMesh(number fileformat,string pathAndFilename,number options,number scalingFactor,table_of_table vertices,table_of_table indices,nil,table names)
Lua parameters	**fileformat**: the fileformat to export to. 0 for OBJ format, 1 for DXF format and 4 for BINARY STL format **pathAndFilename**: the location of the file to create. **options**: keep at 0 **scalingFactor**: the scaling factor to apply to the vertices to export **vertices**: a table of vertice tables. See the example below **indices**: a table of indice tables. See the example below **nil**: reserved for future extensions. **names**: a table of mesh names. See the example below USAGE EXAMPLE (e.g. in a customization script):

```
-- Exports all shapes in the scene
if (exportButtonPressed) then
    allVertices={}
```

```
    allIndices={}
    allNames={}
    shapeIndex=0
    while (true) do
        h=sim.getObjects(shapeIndex,sim.object_shape_type)
        if (h<0) then
            break
        end
        shapeIndex=shapeIndex+1
        vertices,indices=sim.getShapeMesh(h)
        m=sim.getObjectMatrix(h,-1)
        for i=1,#vertices/3,1 do
            v={vertices[3*(i-1)+1],vertices[3*(i-1)+2],vertices[3*(i-1)+3]}
            v=sim.multiplyVector(m,v)
            vertices[3*(i-1)+1]=v[1]
            vertices[3*(i-1)+2]=v[2]
            vertices[3*(i-1)+3]=v[3]
        end
        table.insert(allVertices,vertices)
        table.insert(allIndices,indices)
        table.insert(allNames,sim.getObjectName(h))
    end
    if (#allVertices>0) then
        sim.exportMesh(1,"d:\\example.dxf",0,1,allVertices,allIndices,nil,allNames)
    end
end
```

Lua return values	Same as C-function

simFileDialog / sim.fileDialog

Description	Opens a dialog that allows selecting a file for save or load operations, or a folder. See also sim.msgBox.
C synopsis	simChar* simFileDialog(simInt dlgType,const simChar* title,const simChar* startPath,const simChar* initName,const simChar* extName,const simChar* ext)
C parameters	**dlgType**: the file dialog type. **title**: title of the dialog **startPath**: the initial path. Indicate an empty string for the path to V-REP's application **initName**: the initial name. Can be an empty string **extName**: the extension name, e.g. "text file". Should be an empty string with dlgType=*sim_filedlg_type_folder*. **ext**: the extension, e.g. "txt". Should be an empty string with dlgType=*sim_filedlg_type_folder*.
C return value	pointer to a string representing the selected file name and path, or folder name and path. In case several files were selected for a load operation, then they will be separated by a semicolon. The user is in charge of releasing the buffer with simReleaseBuffer.
Lua synopsis	string pathAndName=sim.fileDialog(number mode,string title,string startPath,string initName,string extName,string ext)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simFindIkPath (DEPRECATED)

Description	DEPRECATED. See sim.generateIkPath instead.

simFindMpPath (DEPRECATED)

Description	DEPRECATED. See the OMPL library based path/motion planning functionality instead. Searches for a collision-free path leading a serial manipulator from a start configuration to a goal configuration. The function uses V-REP's motion planning functionality. The first time this function is called on the given motion planning object, a preprocessing stage will prepare a calculation structure. This might take a few seconds depending mainly on the number of phase1 nodes. By default, a found path will not be simplified, and you should call simSimplifyMpPath on it afterwards (or use a bit in the options parameter).See also sim.getConfigForTipPose, simSimplifyMpPath, sim.generateIkPath,

	simGetMpConfigTransition and sim.checkIkGroup.
C synopsis	simFloat* simFindMpPath(simInt motionPlanningObjectHandle,const simFloat* startConfig,const simFloat* goalConfig,simInt options,simFloat stepSize,simInt* outputConfigsCnt,simInt maxTimeInMs,simFloat* reserved,const simInt* auxIntParams,const simFloat* auxFloatParams)
C parameters	**motionPlanningObjectHandle**: the handle of a motion planning object. Refer to simGetMotionPlanningHandle. **startConfig**: the start or initial configuration of the robot (i.e. its joint positions). Should contain x values where x is the number of DoFs of the specified motion planning task. **goalConfig**: the goal configuration of the robot (i.e. its joint positions). Should contain x values where x is the number of DoFs of the specified motion planning task. You can use simGetConfigForTipPose if the goal configuration is not known. **options**: bit-coded: bit0: if set (1), then the searched collision-free nodes (organized in a search tree) will be visualized in red. Only the search tree having root at the start configuration will be visualized. bit1: if set (2), then the searched collision-free nodes (organized in a search tree) will be visualized in blue. Only the search tree having root at the goal configuration will be visualized. bit2: if set (4), then the found path will be visualized in yellow. bit3: if set (8), then some information will be output to the console. bit4: if set (16), then robot self-interferences will be ignored and calculations can drastically be sped-up. bit5: if set (32), then robot-environment interferences will be ignored and calculations can drastically be sped-up. bit6: if set (64), then the found path will be directly simplified. This can take quite some time and is not part of the maximum calculation time specified. See also simSimplifyMpPath. bit7: not used, keep unset. bit8: if set (256), then the returned Cartesian space distances will ignore the orientational distance component. **stepSize**: the maximum configuration space distance between individual collision-free phase2 nodes. A distance calculation will use the weight specified for each joint in the motion planning properties. **outputConfigsCnt**: a pointer to an integer receiving the number of returned configurations. If a single configuration is returned, this means that the specified start and goal configurations are coincident. **maxTimeInMs**: the maximum time in milliseconds after which the search operation is aborted. Specify 0 for an infinite time. **reserved**: reserved. Keep NULL. **auxIntParams**: reserved. Keep NULL. **auxFloatParams**: reserved. Keep NULL.
C return value	NULL in case of an error, or when the search failed. Otherwise a buffer of float values that the user is in charge of releasing with simReleaseBuffer. The returned buffer contains: **the found path** (x*n values): n configurations with each x values (x is the number of DoFs of the specified motion planning task). The configurations will include the start and the goal configuration, except when start and goal are coincident, in which case a single configuration is returned. **the configuration space distances** (n values): for each returned configuration, a distance to the start configuration (following the path). The last of the n values represents the length of the found path in the configuration space. The distance is calculated using the weight specified for each joint in the motion planning properties. **the end-effector positions** (3*n values): for each returned configuration, the position of the corresponding end-effector (x, y, z). **the end-effector quaternions** (4*n values): for each returned configuration, the quaternion of the corresponding end-effector (x, y, z, w). **the Cartesian space distances** (n values): for each returned configuration, a distance to the start pose (following the path). The last of the n values represents the length of the found path in the Cartesian space. The distance is calculated using the Cartesian space metric specified in the motion planning properties.
Lua synopsis	table path,table confSpaceDist,table tipPositions,table tipQuaternions,table cartesianSpaceDist=simFindMpPath(number motionPlanningObjectHandle,table startConfig,table goalConfig,number options,number stepSize,number maxTimeInMs=0,table auxIntParams=nil,table auxFloatParams=nil)
Lua parameters	Similar as C-function
Lua return values	Similar as C-function

simFloatingViewAdd / sim.floatingViewAdd

Description	Adds a floating view to current page. See also the sim.floatingViewRemove, sim.adjustView and

	sim.cameraFitToView functions.
C synopsis	simInt simFloatingViewAdd(simFloat posX,simFloat posY,simFloat sizeX,simFloat sizeY,simInt options)
C parameters	**posX & posY**: relative position of the center of the floating view. Accepted values are between 0 and 1. **sizeX & sizeY**: relative size of the floating view. Accepted values are between 0 and 1. **options**: bit-coded: bit0 set (1)=double click allows swapping the floating view with the main view bit1 set (2)=the floating view doesn't have a close button bit2 set (4)=the floating view cannot be shifted bit3 set (8)=the floating view cannot be resized
C return value	Handle of the floating view, or -1 in case of an error.
Lua synopsis	number floatingViewHandle=sim.floatingViewAdd(number posX,number posY,number sizeX,number sizeY,number options)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simFloatingViewRemove / sim.floatingViewRemove

Description	Removes a floating view previously added with sim.floatingViewAdd.
C synopsis	simInt simFloatingViewRemove(simInt floatingViewHandle)
C parameters	**floatingViewHandle**: handle of the floating view to be removed
C return value	-1 in case of an error, 0 if the floating view could not be found (e.g. because closed by the user), or 1 if the floating view was closed.
Lua synopsis	number result=sim.floatingViewRemove(number floatingViewHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simFollowPath / sim.followPath

Description	Moves an object along a path object. This function can only be called from child scripts running in a thread (since this is a blocking operation) and is not available from the C-API. See also sim.rmlPos, sim.rmlVel and sim.moveToObject.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number deltaTimeLeft =sim.followPath(number objectHandle,number pathHandle,number positionAndOrOrientation,number relativeDistanceOnPath,number velocity,number accel)
Lua parameters	**objectHandle**: handle of the object to be moved **pathHandle**: handle of the path object **positionAndOrOrientation**: a value between 1 and 3 (1: only position is modified, 2: only orientation is modified, 3: position and orientation is modified). Can be nil in which case 3 is applied. **relativeDistanceOnPath**: a value between 0 and 1, where 0 is the beginning of the path, and 1 the end of the path. Make sure you selected the appropriate path length calculation method (refer to the path position calculation method section). **velocity**: movement nominal velocity. **accel**: the acceleration/deceleration.
Lua return values	**deltaTimeLeft**: if the time needed to follow the path is not a multiple of the simulation time step, then deltatimeLeft is the execution time left at current simulation time. deltaTimeLeft is also memorized internally on a thread-basis and used as compensation or correction factor in subsequent blocking commands. deltaTimeLeft is nil in case of an error.

simGenerateIkPath / sim.generateIkPath

Description	Generates a path that drives a robot from its current configuration to its target dummy in a straight line (i.e. shortest path in Cartesian space).
C synopsis	simFloat* simGenerateIkPath(simInt ikGroupHandle,simInt jointCnt,const simInt* jointHandles,simInt ptCnt,simInt collisionPairCnt,const simInt* collisionPairs,const simInt* jointOptions,simVoid* reserved)

C parameters	**ikGroupHandle**: the handle of an IK group that is in charge of bringing the manipulator's tip onto a target. The IK group can also be marked as *explicit handling* if needed. See also simGetIkGroupHandle. **jointCnt**: the number of joint handles provided in the *jointHandles* array. **jointHandles** (input): an array with *jointCnt* entries, that specifies the joint handles for the joints we wish to retrieve the values calculated by the IK. **ptCnt**: the desired number of path points. Each path point contains a robot configuration. A minimum of two path points is required. If the tip-target dummy distance is large, a larger number for *ptCnt* leads to better results for this function. **collisionPairCnt**: the number of collision pairs. Can be 0 if collision checking is not required. **collisionPairs**: an array containing 2 entity handles for each collision pair. A collision pair is represented by a collider and a collidee, that will be tested against each other. The first pair could be used for robot self-collision testing, and a second pair could be used for robot-environment collision testing. The collider can be an object or a collection handle. The collidee can be an object or collection handle, or *sim_handle_all*, in which case the collider will be checked agains all other collidable objects in the scene. Can be NULL if collision checking is not required. **jointOptions**: a bit-coded value corresponding to each specified joint handle. Bit 0 (i.e. 1) indicates the corresponding joint is dependent of another joint. Can be NULL. **reserved**: reserved for future extension. Set to NULL.
C return value	a pointer to the computed path, or NULL if no path could be computed. The pointer points to *ptCnt* jointCnt* values, representing *ptCnt* robot configurations. The user is in charge of releasing the returned array with simReleaseBuffer.
Lua synopsis	table path=sim.generateIkPath(number ikGroupHandle,table jointHandles,number ptCnt,table collisionPairs=nil,table jointOptions=nil)
Lua parameters	Similar as C-function
Lua return values	Similar as C-function

simGetApiFunc / sim.getApiFunc

Description	Retrieves all API functions and variables that match a specific word. Useful for script code auto-completion functionality. See also sim.getApiInfo.
C synopsis	simChar* simGetApiFunc(simInt scriptHandleOrType,const simChar* apiWord)
C parameters	**scriptHandleOrType**: the handle of the script, otherwise the type of the script. **apiWord**: the word that API functions and variables should match, e.g. "sim.getObj"
C return value	NULL in case of an error, or if there is no match. Otherwise all matching API functions and variables, space-separated. The user is in charge of releasing the returned buffer with simReleaseBuffer.
Lua synopsis	table funcsAndVars=sim.getApiFunc(number scriptHandleOrType,string apiWord)
Lua parameters	Similar to the C-function
Lua return values	**funcsAndVars**: nil in case of an error, otherwise a table containing all matching API functions and variables.

simGetApiInfo / sim.getApiInfo

Description	Returns the call tip (or info text) for an API function or variable. See also sim.getApiFunc.
C synopsis	simChar* simGetApiInfo(simInt scriptHandleOrType,const simChar* apiWord)
C parameters	**scriptHandleOrType**: the handle of the script, otherwise the type of the script. **apiWord**: the API functions or variable to retrieve the info for, e.g. "sim.getObjectHandle"
C return value	NULL in case of an error, or if there is no information. Otherwise the information related to the API function or variable. The user is in charge of releasing the returned buffer with simReleaseBuffer.
Lua synopsis	string info=sim.getApiInfo(number scriptHandleOrType,string apiWord)
Lua parameters	Similar to the C-function
Lua return values	Similar to the C-function

simGetArrayParameter / sim.getArrayParameter (remote API equivalent: simxGetArrayParameter)

Description	Retrieves 3 values from an array. See the array parameter identifiers. See also sim.setArrayParameter, sim.getBoolParameter, sim.getInt32Parameter, sim.getFloatParameter and sim.getStringParameter.
C synopsis	simInt simGetArrayParameter(simInt parameter,simVoid* parameterValues)

C parameters	**parameter**: array parameter identifier **parameterValues**: a simFloat pointer (simVoid is kept for backward compatibility). The 3 values will be copied to that location
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table parameterValues=sim.getArrayParameter(number parameter)
Lua parameters	Same as C-function
Lua return values	**parameterValues**: a table that holds the returned values, or nil in case of an error

simGetBoolParameter / sim.getBoolParameter (remote API equivalent: simxGetBooleanParameter)

Description	Retrieves a boolean value. See the Boolean parameter identifiers. See also sim.setBoolParameter, sim.getInt32Parameter, sim.getFloatParameter, sim.getArrayParameter and sim.getStringParameter.
C synopsis	simInt simGetBoolParameter(simInt parameter)
C parameters	**parameter**: boolean parameter identifier
C return value	value of the parameter (0 or 1) or -1 in case of an error
Lua synopsis	boolean boolState=sim.getBoolParameter(number parameter)
Lua parameters	Same as C-function
Lua return values	**boolState**: value of the Boolean parameter, or nil in case of an error

simGetBooleanParameter / sim.getBoolParameter (DEPRECATED)

Description	DEPRECATED. See sim.getBoolParameter instead.

simGetClosestPositionOnPath / sim.getClosestPositionOnPath

Description	Retrieves the intrinsic relative position on a path that is closest to the specified point. The returned value is dependent on the selected path length calculation method for the given path object. See also sim.getPathPosition, sim.getPathLength, sim.getPositionOnPath and sim.getOrientationOnPath.
C synopsis	simInt simGetClosestPositionOnPath(simInt pathHandle,simFloat* relativePosition,simFloat* pathPosition)
C parameters	**pathHandle**: handle of the path object **relativePosition**: a point in coordinates (x, y and z) relative to the path object position **pathPosition**: (return value). The intrinsic relative position on the path, a value between 0 and 1, where 0 is the beginning of the path, and 1 the end of the path (that value is dependent on the selected path length calculation method).
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number positionOnPath=sim.getClosestPositionOnPath(number pathHandle,table_3 relativePosition)
Lua parameters	**pathHandle**: handle of the path object **relativePosition**: a table containing a point in relative coordinates (x, y and z)
Lua return values	**positionOnPath**: the intrinsic relative position on the path, a value between 0 and 1, where 0 is the beginning of the path, and 1 the end of the path (value is dependent on the selected path length calculation method), or nil in case of an error.

simGetCollectionHandle / sim.getCollectionHandle (remote API equivalent: simxGetCollectionHandle)

Description	Retrieves a collection handle based on its name. The operation of this function depends on the current name suffix settings (see sim.getNameSuffix, sim.setNameSuffix, and the section on accessing general-type objects). See also sim.createCollection, sim.addObjectToCollection, sim.getCollectionObjects and sim.isHandleValid.
C synopsis	simInt simGetCollectionHandle(const simChar* collectionName)
C parameters	**collectionName**: name of the collection

C return value	Handle of the collection, or -1 in case of an error
Lua synopsis	number collectionHandle=sim.getCollectionHandle(string collectionName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetCollectionName / sim.getCollectionName

Description	Retrieves the name of a collection based on its handle. See also sim.setCollectionName.
C synopsis	simChar* simGetCollectionName(simInt collectionHandle)
C parameters	**collectionHandle**: handle of the collection
C return value	pointer to the name of the collection or NULL if an error occurred. The user is in charge of destroying the returned buffer with simReleaseBuffer
Lua synopsis	string collectionName=sim.getCollectionName(number collectionHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function (but nil instead of NULL, and simReleaseBuffer does not need to be called)

simGetCollectionObjects / sim.getCollectionObjects

Description	Retrieves the object handles that compose a given collection.
C synopsis	simInt* simGetCollectionObjects(simInt collectionHandle,simInt* objectCount)
C parameters	**collectionHandle**: handle of the collection **objectCount**: pointer to a value receiving the number of returned object handles
C return value	pointer to n object handles, or NULL if an error occurred. The user is in charge of destroying the returned buffer with simReleaseBuffer
Lua synopsis	table objectHandles=sim.getCollectionObjects(number collectionHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function (but nil instead of NULL, and simReleaseBuffer does not need to be called)

simGetCollisionHandle / sim.getCollisionHandle (remote API equivalent: simxGetCollisionHandle)

Description	Retrieves the handle of a collision object. The operation of this function depends on the current name suffix settings (see sim.getNameSuffix, sim.setNameSuffix, and the section on accessing general-type objects). See also sim.isHandleValid.
C synopsis	simInt simGetCollisionHandle(const simChar* collisionObjectName)
C parameters	**collisionObjectName**: name of the collision object
C return value	handle of collision object or -1 if operation was not successful
Lua synopsis	number collisionObjectHandle=simGetCollisionObject(string collisionObjectName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetConfigForTipPose / sim.getConfigForTipPose

Description	Searches for a manipulator configuration that matches a given end-effector position/orientation in space. Search is randomized.
C synopsis	simInt simGetConfigForTipPose(simInt ikGroupHandle,simInt jointCnt,const simInt* jointHandles,simFloat thresholdDist,simInt maxTimeInMs,simFloat* retConfig,const simFloat* metric,simInt collisionPairCnt,const simInt* collisionPairs,const simInt* jointOptions,const simFloat* lowLimits,const simFloat* ranges,simVoid* reserved)
C parameters	**ikGroupHandle**: the handle of an IK group that is in charge of bringing the manipulator's tip onto a target. The IK group can also be marked as *explicit handling* if needed. See also simGetIkGroupHandle. **jointCnt**: the number of joint handles provided in the *jointHandles* array. **jointHandles** (input): an array with *jointCnt* entries, that specifies the joint handles for the joints we

	wish to retrieve the values calculated by the IK. **thresholdDist**: a distance indicating when IK should be computed in order to try to bring the tip onto the target: since the search algorithm proceeds by generating random configurations, many of them produce a tip pose that is too far from the target pose to run IK successfully. Choosing a large value will result in slow calculations, choosing a small value might produce a smaller subset of solutions. Distance between two poses is calculated using a *metric*. **maxTimeInMs**: a maximum time in ms after which the search is aborted. **retConfig** (output): an array with *jointCnt* entries, that will receive the IK calculated joint values, as specified by the *jointHandles* array. **metric** (input): an array to 4 values indicating a metric used to compute pose-pose distances: distance=sqrt((dx*metric[0])^2+(dy*metric[1])^2+(dz*metric[2])^2+(angle*metric[3])^2). Can be NULL for a default metric of {1.0,1.0,1.0,0.1}. **collisionPairCnt**: the number of collision pairs. Can be 0 if collision checking is not required. **collisionPairs**: an array containing 2 entity handles for each collision pair. A collision pair is represented by a collider and a collidee, that will be tested against each other. The first pair could be used for robot self-collision testing, and a second pair could be used for robot-environment collision testing. The collider can be an object or a collection handle. The collidee can be an object or collection handle, or *sim_handle_all*, in which case the collider will be checked agains all other collidable objects in the scene. Can be NULL if collision checking is not required. **jointOptions**: a bit-coded value corresponding to each specified joint handle. Bit 0 (i.e. 1) indicates the corresponding joint is dependent of another joint. Can be NULL. **lowLimits**: an optional array pointing to different low limit values for each specified joint. This can be useful when you wish to explore a sub-set of the joint's intervals. Can be NULL for the default joint's low limit values. If not NULL, then *ranges* should also not be NULL. **ranges**: an optional array pointing to different range values for each specified joint. This can be useful when you wish to explore a sub-set of the joint's intervals. Can be NULL for the default joint's range values. If not NULL, then *lowLimits* should also not be NULL. **reserved**: reserved for future extension. Set to NULL.
C return value	-1 in case of an error, 0 if no result was found, otherwise 1.
Lua synopsis	table jointPositions=sim.getConfigForTipPose(number ikGroupHandle,table jointHandles,number distanceThreshold,number maxTimeInMs,table_4 metric=nil,table collisionPairs=nil,table jointOptions=nil,table lowLimits=nil,table ranges=nil)
Lua parameters	Similar as C-function
Lua return values	Similar as C-function

simGetConfigurationTree / sim.getConfigurationTree

Description	Retrieves configuration information for a hierarchy tree (object relative positions/orientations, joint/path values). Calling sim.setConfigurationTree at a later time, will restore the object configuration (use this function to temporarily save object positions/orientations/joint/path values)
C synopsis	simChar* simGetConfigurationTree(simInt objectHandle)
C parameters	**objectHandle**: handle of the object that is at the base of the tree (all objects built on top of this one (including this one)) will have their configuration retrieved. sim_handle_all will retrieve the configuration for the whole scene
C return value	Configuration data if operation was successful, NULL otherwise. The returned data should be deleted with simReleaseBuffer when not used anymore
Lua synopsis	number rawBufferHandle=sim.getConfigurationTree(number objectHandle)
Lua parameters	Same as C-function. In addition, child scripts can use the argument sim.handle_self to retrieved the configuration tree of the object that the child script is attached to
Lua return values	**rawBufferHandle**: handle to a block of raw memory, or -1 in case of an error. Use that value to restore the configuration tree with sim.setConfigurationTree. The raw buffer is attached to the script until the simulation ends, at which time it is automatically released. Alternatively, you can release that buffer with the simReleaseScriptRawBuffer-function

simGetContactInfo / sim.getContactInfo

Description	Retrieves contact point information of a dynamic simulation pass.
C synopsis	simInt simGetContactInfo(simInt dynamicPass,simInt objectHandle,simInt index,simInt* objectHandles,simFloat* contactInfo)
C parameters	**dynamicPass**: a specific dynamic sub-step index or sim_handle_all. By default a call to simHandleDynamics executes the dynamics engine x times, with x times smaller time steps (where x is a parameter that can be adjusted). At each of those sub-steps, contacts are created and destroyed.

	With the dynamicPass argument you can select which sub-step you wish to retrieve contacts from (zero-based index), or sim_handle_all to retrieve the contacts of all sub-steps. See also simGetInt32Parameter(sim_intparam_dynamic_step_divider). **objectHandle**: handle of a specific object you wish to retrieve contacts from, or sim_handle_all to retrieve all contacts in the scene. **index**: zero-based index of the contact to retrieve. Optionally, you may add sim_handleflag_extended to the index, if you also wish to retrieve the normal vector (see further down) **objectHandles**: handles of the two objects contacting. The handles might also refer to particle objects that are not treated as regular scene objects. **contactInfo**: pointer to 6 values (or 9 values if sim_handleflag_extended was added to index), where the 3 first values represent the contact position, the 3 next values represent the force generated by the contact, and the (optional) 3 last values represent the normal vector at the contact.
C return value	-1 in case of an error, 0 if no contact was found at the given index or 1 if a contact was returned.
Lua synopsis	table_2 collidingObjects,table_3 collisionPoint,table_3 reactionForce,table_3 normalVector=sim.getContactInfo(number dynamicPass,number objectHandle,number index)
Lua parameters	Same as C-function
Lua return values	**collidingObjects**: handles of the two objects contacting. The handles might also refer to particle objects that are not treated as regular scene objects **collisionPoint**: coordinates of the contact **reactionForce**: vector that represents the force generated by the contact **normalVector**: the normal vector at the contact point

simGetCustomizationScriptAssociatedWithObject / sim.getCustomizationScriptAssociatedWithObject

Description	Retrieves a customization script's handle based on its associated object. See also sim.getObjectAssociatedWithScript and sim.getScriptAssociatedWithObject.
C synopsis	simInt simGetCustomizationScriptAssociatedWithObject(simInt objectHandle)
C parameters	**objectHandle**: handle of the object that might have a customization script associated
C return value	handle of the customization script associated with the object, or -1 if the operation was not successful or the object doesn't have an associated script
Lua synopsis	number scriptHandle=sim.getCustomizationScriptAssociatedWithObject(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetDataOnPath / sim.getDataOnPath

Description	Retrieves interpolated user data along a path object. See also sim.getPositionOnPath, sim.getOrientationOnPath, sim.getPathPosition and sim.getClosestPositionOnPath.
C synopsis	simInt simGetDataOnPath(simInt pathHandle,simFloat relativeDistance,simInt dataType,simInt* intData,simFloat* floatData)
C parameters	**pathHandle**: handle of the path object **relativeDistance**: a value between 0 and 1, where 0 is the beginning of the path, and 1 the end of the path. Make sure you selected the appropriate path length calculation method. See also simGetPathLength. In order to retrieve data that lies exactly on a specific path control point, specify following for *relativeDistance*: -ctrlPtIndex-1 (the value will be rounded appropriately). **dataType**: the type of the desired data. Keep 0. **intData**: pointer to a one-dimensional array that will receive the auxiliary flags value at the given path relative distance. **floatData**: pointer to a 4-dimensional array that will receive the auxiliary channel values at the given path relative distance.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number auxFlags,table_4 auxChannels=sim.getDataOnPath(number pathHandle,number relativeDistance)
Lua parameters	Same as C-function
Lua return values	Similar as C-function

simGetDecimatedMesh / sim.getDecimatedMesh

Description	Retrieves a decimated mesh (i.e. a simplified mesh). See also sim.convexDecompose and sim.getQHull.
C synopsis	simInt simGetDecimatedMesh(const simFloat* inVertices,simInt inVerticesL,const simInt* inIndices,simInt inIndicesL,simFloat** verticesOut,simInt* verticesOutL,simInt** indicesOut,simInt* indicesOutL,simFloat decimationPercent,simInt reserved1,const simFloat* reserved2)
C parameters	**inVertices**: a pointer to the input vertices (succession of x/y/z values). **inVerticesL**: the number of input vertices times 3. **inIndices**: a pointer to the input indices (3 values for each triangle). **inIndicesL**: the number of input triangles times 3. **verticesOut**: a pointer to a pointer to the output vertices. The output vertices are allocated by V-REP and the user is in charge of releasing the buffer via simReleaseBuffer. **verticesOutL**: a pointer to the number of output vertices times 3. **indicesOut**: a pointer to a pointer to the output indices. The output indices are allocated by V-REP and the user is in charge of releasing the buffer via simReleaseBuffer. **indicesOutL**: a pointer to the number of output indices (i.e. the number of triangles times 3). **decimationPercent**: the percentage of the desired decimation (0.1-0.9). **reserved1**: reserved, set to 0. **reserved2**: reserved, set to NULL.
C return value	-1 or 0 if operation was not successful.
Lua synopsis	table verticesOut,table indicesOut=sim.getDecimatedMesh(table verticesIn,table indicesIn,number decimationPercentage)
Lua parameters	**verticesIn**: a table containing the input vertices (succession of x/y/z values). **indicesIn**: a table containing the input indices (3 values for each triangle). **decimationPercentage**: the percentage of the desired decimation (0.1-0.9).
Lua return values	**verticesOut**: a table containing the output vertices (succession of x/y/z values). **indicesOut**: a table containing the output indices (3 values for each triangle).

simGetDialogInput / sim.getDialogInput (remote API equivalent: simxGetDialogInput)

Description	Queries the text of the edit box of a generic dialog box of style sim_dlgstyle_input. To be used after sim.displayDialog was called and after sim.getDialogResult returned sim.dlgret_ok
C synopsis	simChar* simGetDialogInput(simInt genericDialogHandle)
C parameters	**genericDialogHandle**: handle of the generic dialog
C return value	Pointer to a text buffer or NULL in case of an error. The user is in charge of releasing the returned string with simReleaseBuffer.
Lua synopsis	string input=sim.getDialogInput(number genericDialogHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function (but nil instead of NULL, and simReleaseBuffer does not need to be called)

simGetDialogResult / sim.getDialogResult (remote API equivalent: simxGetDialogResult)

Description	Queries the result of a dialog box. To be used after sim.displayDialog was called
C synopsis	simInt simGetDialogResult(simInt genericDialogHandle)
C parameters	**genericDialogHandle**: handle of the generic dialog
C return value	result of the dialog or -1 in case of an error. Note. If the return value is sim_dlgret_still_open, the dialog was not closed and no button was pressed. Otherwise, you should free resources with simEndDialog (the dialog might not be visible anymore, but is still present)
Lua synopsis	number result=sim.getDialogResult(number genericDialogHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetDistanceHandle / sim.getDistanceHandle (remote API equivalent: simxGetDistanceHandle)

Description	Retrieves the handle of a distance object. The operation of this function depends on the current name suffix settings (see sim.getNameSuffix, sim.setNameSuffix, and the section on accessing general-type objects). See also sim.isHandleValid.
C synopsis	simInt simGetDistanceHandle(const simChar* distanceObjectName)
C parameters	**distanceObjectName**: name of distance object
C return value	handle of the distance object or -1 if operation was not successful
Lua synopsis	number distanceObjectHandle=sim.getDistanceHandle(string distanceObjectName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetEngineBoolParameter / sim.getEngineBoolParameter

Description	Retrieves a Boolean value from the physics engine properties. See also the other engine properties setter and getter API functions.
C synopsis	simBool simGetEngineBoolParameter(simInt paramId,simInt objectHandle,const simVoid* object,simBool* ok)
C parameters	**paramId**: the engine parameter identifier. **objectHandle**: the handle of the shape or joint, or -1 to retrieve a global engine parameter. If -1, then the *object* argument will be evaluated. **object**: a pointer to a shape or joint objects, or NULL to retrieve a global engine parameter. If NULL, then the *objectHandle* argument will be evaluated. **ok**: an optional pointer to a value that can be used to determine the success of the API call. Can be NULL.
C return value	value of the requested parameter. This function call doesn't generate any error message.
Lua synopsis	boolean boolParam=sim.getEngineBoolParameter(number paramId,number objectHandle)
Lua parameters	Similar as C-function
Lua return values	Same as C-function

simGetEngineFloatParameter / sim.getEngineFloatParameter

Description	Retrieves a float value from the physics engine properties. See also the other engine properties setter and getter API functions.
C synopsis	simFloat simGetEngineFloatParameter(simInt paramId,simInt objectHandle,const simVoid* object,simBool* ok)
C parameters	**paramId**: the engine parameter identifier. **objectHandle**: the handle of the shape or joint, or -1 to retrieve a global engine parameter. If -1, then the *object* argument will be evaluated. **object**: a pointer to a shape or joint objects, or NULL to retrieve a global engine parameter. If NULL, then the *objectHandle* argument will be evaluated. **ok**: an optional pointer to a value that can be used to determine the success of the API call. Can be NULL.
C return value	value of the requested parameter. This function call doesn't generate any error message.
Lua synopsis	number floatParam=sim.getEngineFloatParameter(number paramId,number objectHandle)
Lua parameters	Similar as C-function
Lua return values	Same as C-function

simGetEngineInt32Parameter / sim.getEngineInt32Parameter

Description	Retrieves an int32 value from the physics engine properties. See also the other engine properties setter and getter API functions.
C synopsis	simInt simGetEngineInt32Parameter(simInt paramId,simInt objectHandle,const simVoid* object,simBool* ok)
C parameters	**paramId**: the engine parameter identifier. **objectHandle**: the handle of the shape or joint, or -1 to retrieve a global engine parameter. If -1, then the *object* argument will be evaluated. **object**: a pointer to a shape or joint objects, or NULL to retrieve a global engine parameter. If NULL, then the *objectHandle* argument will be evaluated.

	ok: an optional pointer to a value that can be used to determine the success of the API call. Can be NULL.
C return value	value of the requested parameter. This function call doesn't generate any error message.
Lua synopsis	number int32Param=sim.getEngineInt32Parameter(number paramId,number objectHandle)
Lua parameters	Similar as C-function
Lua return values	Same as C-function

simGetEulerAnglesFromMatrix / sim.getEulerAnglesFromMatrix

Description	Retrieves the Euler angles from a transformation matrix. See also the other matrix/transformation functions.
C synopsis	simInt simGetEulerAnglesFromMatrix(const simFloat* matrix,simFloat* eulerAngles)
C parameters	**matrix**: pointer to 12 simFloat values (the last row of the 4x4 matrix (0,0,0,1) is not needed) The x-axis of the orientation component is (matrix[0],matrix[4],matrix[8]) The y-axis of the orientation component is (matrix[1],matrix[5],matrix[9]) The z-axis of the orientation component is (matrix[2],matrix[6],matrix[10]) The position component is (matrix[3],matrix[7],matrix[11]) **eulerAngles**: pointer to 3 simFloat values representing the Euler angles of the matrix
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_3 eulerAngles=sim.getEulerAnglesFromMatrix(table_12 matrix)
Lua parameters	**matrix**: table to 12 numbers (the last row of the 4x4 matrix (0,0,0,1) is not needed). Table values in Lua are indexed from 1, not 0!
Lua return values	**eulerAngles**: table to 3 numbers representing the Euler angles, or nil in case of an error

simGetExplicitHandling / sim.getExplicitHandling

Description	Retrieves the explicit handling flags for a general object. See also sim.setExplicitHandling.
C synopsis	simInt simGetExplicitHandling(simInt generalObjectHandle)
C parameters	**generalObjectHandle**: handle of a general object. Can be a scene object, collision object, distance object, etc.
C return value	-1 if command was not successful, otherwise the explicit handling flags for the specified general object (for now only bit 0 is used).
Lua synopsis	number explicitHandlingFlags=sim.getExplicitHandling(number generalObjectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function. See also the boolean operators in Lua.

simGetExtensionString / sim.getExtensionString

Description	Retrieves a string that describes additional environment or object properties, mainly used by extension plugins.
C synopsis	simChar* simGetExtensionString(simInt objectHandle,simInt index,const simChar* key)
C parameters	**objectHandle**: the handle of an object, or -1 if you wish to retrieve the extension string of the environment. **index**: keep to -1, unless the object is a shape, and you wish to retrieve the extension string of a shape component (since a shape can be a compound of several other shapes). **key**: an optional key indicating what value to retrieve. If none is specified, then the whole extension string will be returned. Keys should have the form of *key@parentKey@...@parentKey*. To retrieve the *shadow enabled* value of extension string "povray{ shadow {true} fadeXDist {0.00}}", specify following key: *shadow@povray*. The key is case sensitive.
C return value	a string if the operation was successful. The user is in charge of releasing the buffer with simReleaseBuffer.
Lua synopsis	string theString=sim.getExtensionString(number objectHandle,number index,string key=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetFloatParameter / sim.getFloatParameter (remote API equivalent: simxGetFloatingParameter)

Description	Retrieves a floating point value. See the floating-point parameter identifiers. See also sim.setFloatParameter, sim.getBoolParameter, sim.getInt32Parameter, sim.getArrayParameter and sim.getStringParameter.
C synopsis	simInt simGetFloatParameter(simInt parameter,simFloat* floatState)
C parameters	**parameter**: floating parameter identifier **floatState**: value of the parameter
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number parameterValue=sim.getFloatParameter(number parameter)
Lua parameters	**parameter**: parameter identifier (sim_floatparam_...)
Lua return values	**parameterValue**: value of the parameter or nil in case of an error

simGetFloatSignal / sim.getFloatSignal (remote API equivalent: simxGetFloatSignal)

Description	Gets the value of a float signal. Signals are cleared at simulation start. See also sim.setFloatSignal, the other signal functions, and sim.persistentDataRead.
C synopsis	simInt simGetFloatSignal(const simChar* signalName,simFloat* signalValue)
C parameters	**signalName**: name of the signal **signalValue**: value of the signal
C return value	-1 if operation was not successful, 0 if signal does not exist, 1 if signalValue was retrieved
Lua synopsis	number signalValue=sim.getFloatSignal(string signalName)
Lua parameters	**signalName**: name of the signal
Lua return values	**signalValue**: value of the signal. nil if operation was not successful or if signal does not exist

simGetFloatingParameter / sim.getFloatParameter (DEPRECATED)

Description	DEPRECATED. See sim.getFloatParameter instead.

simGetIkGroupHandle / sim.getIkGroupHandle

Description	Retrieves the handle of an IK group. The operation of this function depends on the current name suffix settings (see sim.getNameSuffix, sim.setNameSuffix, and the section on accessing general-type objects). See also sim.isHandleValid.
C synopsis	simInt simGetIkGroupHandle(const simChar* ikGroupName)
C parameters	**ikGroupName**: name of an IK group
C return value	Handle of the IK group or -1 if operation was not successful
Lua synopsis	number ikGroupHandle=sim.getIkGroupHandle(string ikGroupName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetIkGroupMatrix / sim.getIkGroupMatrix

Description	Retrieves various information from an IK group, such as the Jacobian. For the result to be valid, the Jacobian must previously have been computed via the sim.computeJacobian, sim.handleIkGroup or sim.checkIkGroup functions. Following situations have to be differentiated: **a)** If you call sim.computeJacobian just before calling sim.getIkGroupMatrix, then the returned Jacobian will be the one at the current state/configuration of the mechanism. This is the recommended way to retrieve the Jacobian.

b) If you call sim.handleIkGroup or sim.checkIkGroup just before calling sim.getIkGroupMatrix, then the returned Jacobian will be the one **computed last, while trying to reach the target** (since it usually takes at least 2-3 iterations to reach a target because of the linearization), which is not the current state/configuration of the mechanism, unless the target overlaps the tip.

See also sim.getIkGroupHandle.

C synopsis	simFloat* simGetIkGroupMatrix(simInt ikGroupHandle,simInt options,simInt* matrixSize)

C parameters	**ikGroupHandle**: handle of an IK group
	options: a value indicating what kind of information to retrieve:
	0: to retrieve the last calculated Jacobian. matrixSize[0] is the number of rows (the number of DoFs), matrixSize[1] is the number of columns. See further down how to extract the Jacobian.
	1: to retrieve the manipulability value, i.e. sqrt(det(J*JT)), where J is the Jacobian, and JT the Jacobian transpose. The returned pointer points onto a single value.
	matrixSize: the dimensions of the returned matrix (x=rows, y=columns)
	
	USAGE EXAMPLE (to correctly interpret the Jacobian):
	
	`float jacobianSize[2];`
	`float* jacobian=simGetIkGroupMatrix(ikGroupHandle,0,jacobianSize);`
	
	`// jacobianSize[0] represents the row count of the Jacobian`
	`// (i.e. the number of equations or the number of joints involved`
	`// in the IK resolution of the given kinematic chain)`
	`// Joints appear from tip to base.`
	
	`// jacobianSize[1] represents the column count of the Jacobian`
	`// (i.e. the number of constraints, e.g. x,y,z,alpha,beta,gamma,jointDependency)`
	
	`// The Jacobian data is ordered row-wise, i.e.:`
	`// [row0,col0],[row1,col0],..,[rowN,col0],[row0,col1],[row1,col1],etc.`
	
	`for (int i=0;i<jacobianSize[0];i++)`
	`{`
	`  std::string str;`
	`  for (int j=0;j<jacobianSize[1];j++)`
	`  {`
	`  if (str.size()==0)`
	`  str+=", ";`
	`  str+=boost::str(boost::format("%.1e") % jacobian[j*jacobianSize[0]+i]);`
	`  }`
	`  printf(str.c_str());`
	`}`

C return value	A pointer to x*y float values. The values should be copied since the pointer might not remain valid.
Lua synopsis	table matrix,table_2 matrixSize=sim.getIkGroupMatrix(number ikGroupHandle,number options)

Lua parameters	Same as C-function
	
	USAGE EXAMPLE (to correctly interpret the Jacobian):
	
	`jacobian,jacobianSize=sim.getIkGroupMatrix(ikGroupHandle,0)`
	
	`-- jacobianSize[1] represents the row count of the Jacobian`
	`-- (i.e. the number of equations or the number of joints involved`
	`-- in the IK resolution of the given kinematic chain)`
	`-- Joints appear from tip to base.`
	
	`-- jacobianSize[2] represents the column count of the Jacobian`
	`-- (i.e. the number of constraints, e.g. x,y,z,alpha,beta,gamma,jointDependency,etc.)`
	
	`-- The Jacobian data is ordered row-wise, i.e.:`
	`-- [row1,col1],[row2,col1],..,[rowN,col1],[row1,col2],[row2,col2],etc.`
	
	`for i=1,jacobianSize[1],1 do`
	`  str=''`
	`  for j=1,jacobianSize[2],1 do`
	`  if #str~=0 then`
	`  str=str..', '`
	`  end`
	`  str=str..string.format("%.1e",jacobian[(j-1)*jacobianSize[1]+i])`
	`  end`
	`  print(str)`
	`end`

Lua return values	Same as C-function

simGetInt32Parameter / sim.getInt32Parameter (remote API equivalent: simxGetIntegerParameter)

Description	Retrieves an integer value. See the integer parameter identifiers. See also sim.setInt32Parameter, sim.getBoolParameter, sim.getFloatParameter, sim.getArrayParameter and sim.getStringParameter.
C synopsis	simInt simGetInt32Parameter(simInt parameter,simInt* intState)
C parameters	**parameter**: integer parameter identifier **intState**: value of the parameter
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number parameterValue=sim.getInt32Parameter(number parameter)
Lua parameters	**parameter**: parameter identifier (sim_intparam_...)
Lua return values	**parameterValue**: value of the parameter or nil in case of an error

simGetIntegerParameter / sim.getInt32Parameter (DEPRECATED)

Description	DEPRECATED. See sim.getInt32Parameter instead.

simGetIntegerSignal / sim.getIntegerSignal (remote API equivalent: simxGetIntegerSignal)

Description	Gets the value of an integer signal. Signals are cleared at simulation start. See also sim.setIntegerSignal, the other signal functions, and sim.persistentDataRead.
C synopsis	simInt simGetIntegerSignal(const simChar* signalName,simInt* signalValue)
C parameters	**signalName**: name of the signal **signalValue**: value of the signal
C return value	-1 if operation was not successful, 0 if signal does not exist, 1 if signalValue was retrieved
Lua synopsis	number signalValue=sim.getIntegerSignal(string signalName)
Lua parameters	**signalName**: name of the signal
Lua return values	**signalValue**: value of the signal. nil if operation was not successful or if signal does not exist

simGetInvertedMatrix (DEPRECATED)

Description	DEPRECATED. See sim.invertMatrix instead.

simGetJointForce / sim.getJointForce (remote API equivalent: simxGetJointForce)

Description	Retrieves the force or torque applied to a joint along/about its active axis. This function retrieves meaningful information only if the joint is prismatic or revolute, and is dynamically enabled. With the Bullet engine, this function returns the force or torque applied to the joint motor (torques from joint limits are not taken into account). With the ODE and Vortex engine, this function returns the total force or torque applied to a joint along/about its z-axis. See also sim.setJointForce and sim.readForceSensor.
C synopsis	simInt simGetJointForce(simInt jointHandle,simFloat* forceOrTorque)
C parameters	**jointHandle**: handle of the joint. Can be combined with sim_handleflag_rawvalue (simply add sim_handleflag_rawvalue to jointHandle), if you wish to access the raw values generated by each individual dynamic simulation step (by default, there are 10 dynamic simulation steps for each simulation step). Raw values can only be accessed from inside a joint callback function. **forceOrTorque**: the force or the torque applied to the joint along/about its z-axis.
C return value	-1 if operation was not successful. 0 if no value is available yet (e.g. when simHandleDynamics hasn't yet handled that joint), otherwise a value >0.
Lua synopsis	number forceOrTorque=sim.getJointForce(number jointHandle)
Lua parameters	Same as C-function
Lua return values	**forceOrTorque**: the force or the torque applied to the joint along/about its z-axis, or nil in case of an

	error, or if no value is available yet.

simGetJointInterval / sim.getJointInterval

Description	Retrieves the interval parameters of a joint. See also sim.setJointInterval.
C synopsis	simInt simGetJointInterval(simInt objectHandle,simBool* cyclic,simFloat* interval)
C parameters	**objectHandle**: handle of the joint **cyclic**: indicates whether the joint is cyclic (the joint varies between -pi and +pi in a cyclic manner) **interval**: interval of the joint. interval[0] is the joint minimum allowed value, interval[1] is the joint range (the maximum allowed value is interval[0]+interval[1]). When the joint is "cyclic", then the interval parameters don't have any meaning.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	boolean cyclic,table_2 interval=sim.getJointInterval(number objectHandle)
Lua parameters	**objectHandle**: handle of the joint
Lua return values	**cyclic**: indicates whether the joint is cyclic (the joint varies between -pi and +pi in a cyclic manner). Is nil in case of an error. **interval**: interval of the joint. interval[0] is the joint minimum allowed value, interval[1] is the joint range (the maximum allowed value is interval[0]+interval[1]). When the joint is "cyclic", then the interval parameters don't have any meaning. Is nil in case of an error.

simGetJointMatrix / sim.getJointMatrix (remote API equivalent: simxGetJointMatrix)

Description	Retrieves the intrinsic transformation matrix of a joint (the transformation caused by the joint movement). See also sim.setSphericalJointMatrix.
C synopsis	simInt simGetJointMatrix(simInt objectHandle,simFloat* matrix)
C parameters	**objectHandle**: handle of the joint **matrix**: pointer to 12 simFloat values (the last row of the 4x4 matrix (0,0,0,1) is not needed) The x-axis of the orientation component is (matrix[0],matrix[4],matrix[8]) The y-axis of the orientation component is (matrix[1],matrix[5],matrix[9]) The z-axis of the orientation component is (matrix[2],matrix[6],matrix[10]) The translation component is (matrix[3],matrix[7],matrix[11])
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_12 matrix=sim.getJointMatrix(number objectHandle)
Lua parameters	**objectHandle**: handle of the joint
Lua return values	**matrix**: table of 12 numbers (the last row of the 4x4 matrix (0,0,0,1) is not returned), or nil in case of an error. Table values in Lua are indexed from 1, not 0!

simGetJointMode / sim.getJointMode

Description	Retrieves the operation mode of a joint. See also sim.setJointMode.
C synopsis	simInt simGetJointMode(simInt jointHandle,simInt* options)
C parameters	**jointHandle**: handle of the joint object **options** (output): bit-coded: if bit0 is set (1), the joint operates in hybrid mode.
C return value	-1 if operation was not successful, otherwise the joint mode.
Lua synopsis	number jointMode,number options=sim.getJointMode(number jointHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function. See also the boolean operators in Lua.

simGetJointPosition / sim.getJointPosition (remote API equivalent: simxGetJointPosition)

Description	Retrieves the intrinsic position of a joint. This function cannot be used with spherical joints (use

	sim.getJointMatrix instead). See also sim.setJointPosition.
C synopsis	simInt simGetJointPosition(simInt objectHandle,simFloat* position)
C parameters	**objectHandle**: handle of the joint **position**: intrinsic position of the joint. This is a one-dimensional value: if the joint is revolute, the rotation angle is returned, if the joint is prismatic, the translation amount is returned, etc.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number position=sim.getJointPosition(number objectHandle)
Lua parameters	**objectHandle**: handle of the joint
Lua return values	**position**: intrinsic position of the joint or nil in case of an error

simGetJointTargetPosition / sim.getJointTargetPosition

Description	Retrieves the target position of a joint. See also sim.setJointTargetPosition.
C synopsis	simInt simGetJointTargetPosition(simInt objectHandle,simFloat* targetPosition)
C parameters	**objectHandle**: handle of the joint object **targetPosition** (output): target position of the joint (angular or linear value depending on the joint type)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number targetPosition=sim.getJointTargetPosition(number objectHandle)
Lua parameters	Same as C-function
Lua return values	**targetPosition**: target position of the joint, or nil in case of an error.

simGetJointTargetVelocity / sim.getJointTargetVelocity

Description	Retrieves the intrinsic target velocity of a non-spherical joint. See also sim.setJointTargetVelocity.
C synopsis	simInt simGetJointTargetVelocity(simInt objectHandle,simFloat* targetVelocity)
C parameters	**objectHandle**: handle of the joint object **targetVelocity** (output): target velocity of the joint (linear or angular velocity depending on the joint-type).
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number targetVelocity=sim.getJointTargetVelocity(number objectHandle)
Lua parameters	Same as C-function
Lua return values	**targetVelocity**: target velocity of the joint, or -1 in case of an error.

simGetJointType / sim.getJointType

Description	Retrieves the type of a joint
C synopsis	simInt simGetJointType(simInt objectHandle)
C parameters	**objectHandle**: handle of the joint
C return value	Type of the joint (sim_joint_revolute_subtype, sim_joint_prismatic_subtype or sim_joint_spherical_subtype), or -1 if operation was not successful
Lua synopsis	number jointType=sim.getJointType(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetLastError / sim.getLastError (remote API equivalent: simxGetLastErrors)

Description	Retrieves the last generated error message for the calling thread and/or script. By calling this function, the last error message is reset and a subsequent call to this function returns NULL. Errors are

	memorized on a thread- and/or script basis (e.g. each script have each an individual error handler, so does the C API functions (when the simulation thread and the GUI thread are differentiated)). See also simSetLastError, the sim.intparam_error_report_mode and the error report modes.
C synopsis	simChar* simGetLastError()
C parameters	None
C return value	Error message buffer or NULL if no error message is present. The user has to delete the returned buffer with a call to simReleaseBuffer
Lua synopsis	string lastError=sim.getLastError()
Lua parameters	Same as C-function
Lua return values	Same as C-function (but nil instead of NULL, and simReleaseBuffer does not need to be called)

simGetLightParameters / sim.getLightParameters

Description	Retrieves various parameters of a light object. See also sim.setLightParameters.
C synopsis	simInt simGetLightParameters(simInt objectHandle,simFloat* setToNULL,simFloat* diffusePart,simFloat* specularPart)
C parameters	**objectHandle**: handle of the light **setToNULL**: not used, set to NULL **diffusePart**: red, green and blue component of the light's diffuse part. Can be NULL **specularPart**: red, green and blue component of the light's specular part. Can be NULL
C return value	-1 in case of an error, otherwise bit-coded: for now, only bit 0 is used: 1=light on
Lua synopsis	number state,table_3 zero,table_3 diffusePart,table_3 specularPart=sim.getLightParameters(number objectHandle)
Lua parameters	**objectHandle**: handle of the light
Lua return values	**state**: -1 in case of an error, otherwise bit-coded: for now, only bit 0 is used: 1=light on. See also the boolean operators in Lua. **zero**: ignore this value **diffusePart**: red, green and blue component of the light's diffuse part **specularPart**: red, green and blue component of the light's specular part

simGetLinkDummy / sim.getLinkDummy

Description	Retrieves the object handle of the dummy linked to this one. See also sim.setLinkDummy.
C synopsis	simInt simGetLinkDummy(simInt dummyHandle)
C parameters	**dummyHandle**: handle of the dummy whose linked dummy has to be retrieved.
C return value	Handle of the dummy linked to the specified dummy object, or -1 if the dummy is not linked or in case of an error
Lua synopsis	number linkedDummyHandle=sim.getLinkDummy(number dummyHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetMainWindow

Description	Retrieves the handle or pointer of the main window.
C synopsis	simVoid* simGetMainWindow(simInt type)
C parameters	**type**: type of the desired return value. 0 for a native window handle, 1 for a pointer to a QWidget object.
C return value	a native window handle or a pointer to a QWidget object.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetMaterialId (DEPRECATED)

Description	DEPRECATED. See sim.setShapeMaterial instead.

simGetMechanismHandle / sim.getMechanismHandle

Description	Retrieves the handle of a mechanism to be solved by the geometric constraint solver. The operation of this function depends on the current name suffix settings (see sim.getNameSuffix, sim.setNameSuffix, and the section on accessing general-type objects). See also sim.isHandleValid.
C synopsis	simInt simGetMechanismHandle(const simChar* mechanismName)
C parameters	**mechanismName**: name of the mechanism
C return value	Handle of the mechanism if operation was successful, -1 otherwise
Lua synopsis	number mechanismHandle=sim.getMechanismHandle(string mechanismName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetModelProperty / sim.getModelProperty (remote API equivalent: simxGetModelProperty)

Description	Retrieves the properties of a model. See also sim.setModelProperty, sim.getObjectProperty and sim.getObjectSpecialProperty.
C synopsis	simInt simGetModelProperty(simInt objectHandle)
C parameters	**objectHandle**: handle of the object that serves as the model base
C return value	model property values, or -1 if operation was not successful
Lua synopsis	number property=sim.getModelProperty(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function. See also the boolean operators in Lua.

simGetModuleInfo / sim.getModuleInfo

Description	Returns auxiliary information about a loaded plugin. See also simSetModuleInfo and sim.getModuleName.
C synopsis	simInt simGetModuleInfo(const simChar* moduleName,simInt infoType,simChar** stringInfo,simInt* intInfo)
C parameters	**moduleName**: the name of the plugin. See sim.getModuleName. **infoType**: the type of information to retrieve: 0: *extended version string* 1: *build date string* 2: *extended version integer* **stringInfo**: a pointer to a string information, in case the information type is for a string. The user is in charge of releasing the string buffer with sim.releaseBuffer **intInfo**: a pointer to an integer information, in case the information type is for an integer.
C return value	-1 in case of an error
Lua synopsis	string/number info=sim.getModuleInfo(string moduleName,number infoType)
Lua parameters	Similar to the C-function
Lua return values	Similar to the C-function

simGetModuleName / sim.getModuleName

Description	Retrieves a plugin name that was previously registered with sim.loadModule. The simulator normally automatically loads and registers plugins present in the application directory. Users can use the sim.getModuleName to verify if a specific module is present
C synopsis	simChar* simGetModuleName(simInt index,sumUChar* moduleVersion)

C parameters	**index**: index to a module. To list-up all module names, start with index=0 and increment index until return value is NULL **moduleVersion**: version of the plugin. Can be NULL.
C return value	Name of the module or NULL if no module is available at index position, or in case of an error. The user is in charge of destroying the returned name with simReleaseBuffer
Lua synopsis	string moduleName,number moduleVersion=sim.getModuleName(number index)
Lua parameters	**index**: index to a module. To list-up all module names, start with index=0 and increment index until return value is nil
Lua return values	**moduleName**: name of the module or nil if no module is available at that index position, or in case of an error. **moduleVersion**: version of the plugin, or nil if moduleName is also nil

simGetMotionPlanningHandle (DEPRECATED)

Description	DEPRECATED. See the OMPL library based path/motion planning functionality instead. Retrieves the handle of a motion planning object. The operation of this function depends on the current name suffix settings (see sim.getNameSuffix, sim.setNameSuffix, and the section on accessing general-type objects). See also sim.isHandleValid.
C synopsis	simInt simGetMotionPlanningHandle(const simChar* motionPlanningObjectName)
C parameters	**motionPlanningObjectName**: name of the motion planning object
C return value	handle of the motion planning object or -1 if operation was not successful
Lua synopsis	number motionPlanningObjectHandle=simGetMotionPlanningObject(string motionPlanningObjectName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetMpConfigForTipPose (DEPRECATED)

Description	DEPRECATED. See sim.getConfigForTipPose instead.

simGetMpConfigTransition (DEPRECATED)

Description	DEPRECATED. See the OMPL library based path/motion planning functionality instead. Searches for a collision-free path leading a serial manipulator from a start configuration to a goal configuration, by trying to follow a predefined path in the Cartesian space. This is useful for redundant manipulators, but also for safely driving non-redundant manipulators via IK through a singular configuration. The function uses V-REP's motion planning functionality. The first time this function is called on the given motion planning object, a preprocessing stage will prepare a calculation structure. This might take a few seconds depending mainly on the number of phase1 nodes. See also sim.getConfigForTipPose, simFindMpPath, sim.generateIkPath and sim.checkIkGroup.
C synopsis	simFloat* simGetMpConfigTransition(simInt motionPlanningObjectHandle,const simFloat* startConfig,const simFloat* goalConfig,simInt options,const simInt* select,simFloat calcStepSize,simFloat maxOutStepSize,simInt wayPointCnt,const simFloat* wayPoints,simInt* outputConfigsCnt,const simInt* auxIntParams,const simFloat* auxFloatParams)
C parameters	**motionPlanningObjectHandle**: the handle of a motion planning object. Refer to simGetMotionPlanningHandle. **startConfig**: the start or initial configuration of the robot (i.e. its joint positions). Should contain x values where x is the number of DoFs of the specified motion planning task. **goalConfig**: the goal configuration of the robot (i.e. its joint positions). Should contain x values where x is the number of DoFs of the specified motion planning task. You can use simGetConfigForTipPose if the goal configuration is not known. **options**: bit-coded: bit0 (1): reserved. Keep unset. bit1 (2): reserved. Keep unset. bit2: if set (4), then the found path will be visualized in yellow. bit3: if set (8), then some information will be output to the console. bit4: if set (16), then robot self-interferences will be ignored and calculations can drastically

	be sped-up. bit5: if set (32), then robot-environment interferences will be ignored and calculations can drastically be sped-up. bit6 (64): reserved. Keep unset. bit7 (128): reserved. Keep unset. bit8: if set (256), then the returned Cartesian space distances will ignore the orientational distance component. bit9: if set (512), then the specified waypoints will be followed by the end-effector in the Cartesian space. For that to happen, IK will be used too. If no waypoints are specified, then the end-effector will link the start to the goal configuration via a straight path in the Cartesian space. **select**: an optional array that describes the behaviour of specific joints during the operation, when bit9 of **options** is set. Can be NULL, in which case all joints are treated equally. The first value in the array indicates how many joint behaviour descriptions will follow. Then, joint behaviour descriptions are appended, with 2 values per joint: value 1: a joint handle. value 2: a value indicating how the joint will be handled during the path search operation: 0: the joint will be fixed, i.e. perform an exact interpolation between the start and goal configuration. 1-7: the weight of the joint during the IK operations: 1=weight is 0.5, 2=weight is 0.25, 3=weight is 0.1, 4=weight is 0.05, 5=weight is 0.025, 6=weight is 0.01, and 7=weight is 0.001 A good strategy for redundant manipulators with redundancy level n, is to fix n joints, if resolution is not successful by keeping the select argument NULL. **calcStepSize**: the maximum configuration space distance between individual collision-free phase2 nodes, during the interpolation phase. A distance calculation will use the weight specified for each joint in the motion planning properties. **maxOutCalcStepSize**: the maximum configuration space distance between individual collision-free phase2 nodes, after the IK calculation phase. Keep this distance larger than **calcStepSize**. A distance calculation will use the weight specified for each joint in the motion planning properties. **wayPointCnt**: the number of provided waypoints. Providing waypoints is optional, and only makes sense if bit9 of **options** is set. If no waypoints are provided, this should be 0. Otherwise this should be at least 2. **wayPoints**: an array that contains optional waypoints. If no waypoints are provided, this should be NULL. For n waypoints, this array should contain n*7 values: for each waypoint, provide the x,y,z coordinates, and the quaternion values (qx,qy,qz,qw). **outputConfigsCnt**: a pointer to an integer receiving the number of returned configurations. If a single configuration is returned, this means that the specified start and goal configurations are coincident. **auxIntParams**: reserved. Keep NULL. **auxFloatParams**: reserved. Keep NULL.
C return value	NULL in case of an error, or when the search failed. Otherwise a buffer of float values that the user is in charge of releasing with simReleaseBuffer. The returned buffer contains: **the found path** (x*n values): n configurations with each x values (x is the number of DoFs of the specified motion planning task). The configurations will include the start and the goal configuration, except when start and goal are coincident, in which case a single configuration is returned. **the configuration space distances** (n values): for each returned configuration, a distance to the start configuration (following the path). The last of the n values represents the length of the found path in the configuration space. The distance is calculated using the weight specified for each joint in the motion planning properties. **the end-effector positions** (3*n values): for each returned configuration, the position of the corresponding end-effector (x, y, z). **the end-effector quaternions** (4*n values): for each returned configuration, the quaternion of the corresponding end-effector (x, y, z, w). **the Cartesian space distances** (n values): for each returned configuration, a distance to the start pose (following the path). The last of the n values represents the length of the found path in the Cartesian space. The distance is calculated using the Cartesian space metric specified in the motion planning properties.
Lua synopsis	table path,table confSpaceLengths,table_3 tipPositions,table_4 tipQuaternions,table cartesianSpaceLengths=simGetMpConfigTransition(number motionPlanningObjectHandle,table startConfig,table goalConfig,number options,table select,number calcStepSize,number maxOutStepSize,table wayPoints=nil)
Lua parameters	Similar as C-function, except that the **select** table (if not nil) should not contain the first element that is required in the C-function
Lua return values	Similar as C-function

simGetNameSuffix / sim.getNameSuffix

Description	Returns the name suffix for an object name (e.g. "myRobot#42"'s name suffix is 42), or retrieves the name suffix set for the current script or for c/c++ API calls. See also sim.setNameSuffix, and read the section on accessing general-type objects.
C synopsis	simInt simGetNameSuffix(const simChar* nameWithSuffix)
C parameters	**nameWithSuffix**: full name (e.g. "myRobot#42"), or NULL to retrieve the name suffix for all c/c++ API calls
C return value	Name suffix of nameWithSuffix, or current name suffix for c/c++ API calls
Lua synopsis	(1) number nameSuffix,string name=sim.getNameSuffix(string nameWithSuffix): retieves the name suffix of nameWithSuffix (2) number nameSuffix=sim.getNameSuffix(nil): retrieves the name suffix set for current script
Lua parameters	**nameWithSuffix**: full name (e.g. "myRobot#42")
Lua return values	**nameSuffix**: name suffix (e.g. 42) of nameWithSuffix, or name suffix that is set for current script **name**: name without suffix (e.g. "myRobot") or nil if the sim.getNameSuffix argument was nil

simGetNavigationMode / sim.getNavigationMode

Description	Retrieves the navigation and selection mode for the mouse. See also sim.setNavigationMode.
C synopsis	simInt simGetNavigationMode()
C parameters	None
C return value	navigation mode if operation was successful, -1 otherwise
Lua synopsis	number navigationMode=sim.getNavigationMode()
Lua parameters	Same as C-function
Lua return values	Same as C-function. See also the boolean operators in Lua.

simGetObjectAssociatedWithScript / sim.getObjectAssociatedWithScript

Description	Retrives the handle of the object the script is attached to. See also sim.getScriptAssociatedWithObject, sim.getCustomizationScriptAssociatedWithObject and sim.associateScriptWithObject.
C synopsis	simInt simGetObjectAssociatedWithScript(simInt scriptHandle)
C parameters	**scriptHandle**: handle of the script
C return value	Handle of the object that is associated with the script, or -1 if no object is associated with the script, or in case of an error.
Lua synopsis	number objectHandle=sim.getObjectAssociatedWithScript (number scriptHandle)
Lua parameters	**scriptHandle**: handle of the script, or sim.handle_self for the handle of the current script
Lua return values	**objectHandle**: handle of the object that the script is associated with, or -1 if the script is not associated (e.g. main scripts or add-ons don't have associated objects) or in case of an error.

simGetObjectChild / sim.getObjectChild (remote API equivalent: simxGetObjectChild)

Description	Retrieves the handle of an object's child object. See also sim.getObjectParent and sim.getObjectsInTree.
C synopsis	simInt simGetObjectChild(simInt objectHandle,simInt index)
C parameters	**objectHandle**: handle of the object **index**: zero-based index of the child's position. To retrieve all children of an object, call the function by increasing the index until the return value is -1
C return value	handle of child object or -1 if the child doesn't exist at that index or in case of an error
Lua synopsis	number childHandle=sim.getObjectChild(number objectHandle,number index)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetObjectConfiguration / sim.getObjectConfiguration

Description	Retrieves configuration information for an object (object relative position/orientation, joint/path value). See also sim.setObjectConfiguration and sim.getConfigurationTree.
C synopsis	simChar* simGetObjectConfiguration(simInt objectHandle)
C parameters	**objectHandle**: handle of the object
C return value	Pointer to configuration data if operation was successful, NULL otherwise. The returned data should be deleted with simReleaseBuffer when not used anymore
Lua synopsis	number rawBufferHandle=sim.getObjectConfiguration(number objectHandle)
Lua parameters	Same as C-function
Lua return values	**rawBufferHandle**: handle to a raw data buffer, or -1 in case of an error. The raw buffer is attached to the script until the simulation ends, at which time it is automatically released. Alternatively, you can release that buffer with the simReleaseScriptRawBuffer-function

simGetObjectCustomData (DEPRECATED)

Description	DEPRECATED. See sim.readCustomDataBlock instead.

simGetObjectFloatParameter / sim.getObjectFloatParameter (remote API equivalent: simxGetObjectFloatParameter)

Description	Retrieves a floating-point parameter of a scene object or calculation object. See also sim.setObjectFloatParameter, sim.getObjectInt32Parameter and sim.getObjectStringParameter
C synopsis	simInt simGetObjectFloatParameter(simInt objectHandle,simInt parameterID,simFloat* parameter)
C parameters	**objectHandle**: handle of the object **parameterID**: identifier of the parameter to retrieve. See the list of all possible object parameter identifiers **parameter**: retrieved parameter
C return value	-1 in case of an error, 0 if the parameter could not be retrieved (e.g. because the parameterID doesn't exist, or because the specified object doesn't correspond to the correct type), or 1 if the operation was successful
Lua synopsis	number result,number parameter=sim.getObjectFloatParameter(number objectHandle,number parameterID)
Lua parameters	Same as C-function
Lua return values	**result**: -1 in case of an error, 0 if the parameter could not be retrieved (e.g. because the parameterID doesn't exist, or because the specified object doesn't correspond to the correct type), or 1 if the operation was successful **parameter**: retrieved parameter

simGetObjectHandle / sim.getObjectHandle (remote API equivalent: simxGetObjectHandle)

Description	Retrieves an object handle based on its name. The operation of this function depends on the current name suffix settings (see sim.getNameSuffix, sim.setNameSuffix, and the section on accessing general-type objects). See also sim.isHandleValid and sim.getObjectUniqueIdentifier.
C synopsis	simInt simGetObjectHandle(const simChar* objectName)
C parameters	**objectName**: name of object. If the name is appended by a "@alt" suffix, then the object handle based on the object's alternative name will be retrieved. If the name is appended by a "@silentError" suffix, then no error will be output if the object does not exist.
C return value	handle of object or -1 if operation was not successful
Lua synopsis	number objectHandle=sim.getObjectHandle(string objectName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetObjectInt32Parameter / sim.getObjectInt32Parameter (remote API equivalent: simxGetObjectIntParameter)

Description	Retrieves an int32 parameter of a scene object or calculation object. See also sim.setObjectInt32Parameter, sim.getObjectFloatParameter and sim.getObjectStringParameter
C synopsis	simInt simGetObjectInt32Parameter(simInt objectHandle,simInt parameterID,simInt* parameter)
C parameters	**objectHandle**: handle of the object **parameterID**: identifier of the parameter to retrieve. See the list of all possible object parameter identifiers **parameter**: retrieved parameter
C return value	-1 in case of an error, 0 if the parameter could not be retrieved (e.g. because the parameterID doesn't exist, or because the specified object doesn't correspond to the correct type), or 1 if the operation was successful
Lua synopsis	number result,number parameter=sim.getObjectInt32Parameter(number objectHandle,number parameterID)
Lua parameters	Same as C-function
Lua return values	**result**: -1 in case of an error, 0 if the parameter could not be retrieved (e.g. because the parameterID doesn't exist, or because the specified object doesn't correspond to the correct type), or 1 if the operation was successful **parameter**: retrieved parameter. See also the boolean operators in Lua.

simGetObjectIntParameter / sim.getObjectInt32Parameter (DEPRECATED)

Description	DEPRECATED. See sim.getObjectInt32Parameter instead.

simGetObjectMatrix / sim.getObjectMatrix

Description	Retrieves the transformation matrix of an object. See also sim.setObjectMatrix, sim.getObjectPosition, sim.getObjectOrientation and the other matrix/transformation functions.
C synopsis	simInt simGetObjectMatrix(simInt objectHandle,simInt relativeToObjectHandle,simFloat* matrix)
C parameters	**objectHandle**: handle of the object **relativeToObjectHandle**: indicates relative to which reference frame we want the matrix. Specify -1 to retrieve the absolute transformation matrix, sim_handle_parent to retrieve the transformation matrix relative to the object's parent, or an object handle relative to whose reference frame we want the transformation matrix. **matrix**: pointer to 12 simFloat values (the last row of the 4x4 matrix (0,0,0,1) is not needed) The x-axis of the orientation component is (matrix[0],matrix[4],matrix[8]) The y-axis of the orientation component is (matrix[1],matrix[5],matrix[9]) The z-axis of the orientation component is (matrix[2],matrix[6],matrix[10]) The translation component is (matrix[3],matrix[7],matrix[11])
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_12 matrix=sim.getObjectMatrix(number objectHandle,number relativeToObjectHandle)
Lua parameters	**objectHandle**: handle of the object **relativeToObjectHandle**: indicates relative to which reference frame we want the matrix. Specify -1 to retrieve the absolute transformation matrix, sim.handle_parent to retrieve the transformation matrix relative to the object's parent, or an object handle relative to whose reference frame we want the transformation matrix.
Lua return values	**matrix**: table of 12 numbers (the last row of the 4x4 matrix (0,0,0,1) is not returned), or nil in case of an error. Table values in Lua are indexed from 1, not 0!

simGetObjectName / sim.getObjectName

Description	Retrieves the name of an object based on its handle. See also sim.setObjectName.
C synopsis	simChar* simGetObjectName(simInt objectHandle)
C parameters	**objectHandle**: handle of the object. By adding *sim.handleflag_altname* to the object handle, the object alternative name can be retrieved.
C return value	Name (or alternative name) of the object if operation was successful, NULL otherwise. The user is in charge of destroying the returned buffer with simReleaseBuffer
Lua synopsis	string objectName=sim.getObjectName(number objectHandle)

Lua parameters	Same as C-function
Lua return values	Same as C-function (but nil instead of NULL, and simReleaseBuffer does not need to be called)

simGetObjectOrientation / sim.getObjectOrientation (remote API equivalent: simxGetObjectOrientation)

Description	Retrieves the orientation (Euler angles) of an object. See also sim.getObjectQuaternion, sim.setObjectOrientation, sim.getObjectPosition, sim.getObjectMatrix and the other matrix/transformation functions.
C synopsis	simInt simGetObjectOrientation(simInt objectHandle,simInt relativeToObjectHandle,simFloat* eulerAngles)
C parameters	**objectHandle**: handle of the object **relativeToObjectHandle**: indicates relative to which reference frame we want the orientation. Specify -1 to retrieve the absolute orientation, sim_handle_parent to retrieve the orientation relative to the object's parent, or an object handle relative to whose reference frame you want the orientation. **eulerAngles**: Euler angles (alpha, beta and gamma)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_3 eulerAngles=sim.getObjectOrientation(number objectHandle,number relativeToObjectHandle)
Lua parameters	Same as C-function
Lua return values	**eulerAngles**: table of 3 values (Euler angles) or nil in case of an error

simGetObjectParent / sim.getObjectParent (remote API equivalent: simxGetObjectParent)

Description	Retrieves the handle of an object's parent object. See also sim.setObjectParent and sim.getObjectChild.
C synopsis	simInt simGetObjectParent(simInt objectHandle)
C parameters	**objectHandle**: handle of the object
C return value	handle of parent or -1 if the parent doesn't exist or in case of an error
Lua synopsis	number parentHandle=sim.getObjectParent(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetObjectPosition / sim.getObjectPosition (remote API equivalent: simxGetObjectPosition)

Description	Retrieves the position of an object. See also sim.setObjectPosition, sim.getObjectOrientation, sim.getObjectMatrix and the other matrix/transformation functions.
C synopsis	simInt simGetObjectPosition(simInt objectHandle,simInt relativeToObjectHandle,simFloat* position)
C parameters	**objectHandle**: handle of the object **relativeToObjectHandle**: indicates relative to which reference frame we want the position. Specify -1 to retrieve the absolute position, sim_handle_parent to retrieve the position relative to the object's parent, or an object handle relative to whose reference frame we want the position. **position**: pointer to 3 values (x, y and z)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_3 position=sim.getObjectPosition(number objectHandle,number relativeToObjectHandle)
Lua parameters	Same as C-function
Lua return values	**position**: table of 3 values (x, y and z) or nil in case of an error

simGetObjectProperty / sim.getObjectProperty

Description	Retrieves the main properties of a scene object. See also sim.setObjectProperty, sim.getObjectSpecialProperty and sim.getModelProperty.
C synopsis	simInt simGetObjectProperty(simInt objectHandle)
C parameters	**objectHandle**: handle of the object

C return value	object property values, -1 if operation was not successful
Lua synopsis	number property=sim.getObjectProperty(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function. See also the boolean operators in Lua.

simGetObjectQuaternion / sim.getObjectQuaternion

Description	Retrieves the quaternion (x,y,z,w) of an object. See also sim.getObjectOrientation, sim.getObjectMatrix and the other matrix/transformation functions.
C synopsis	simInt simGetObjectQuaternion(simInt objectHandle,simInt relativeToObjectHandle,simFloat* quaternion)
C parameters	**objectHandle**: handle of the object **relativeToObjectHandle**: indicates relative to which reference frame we want the orientation. Specify -1 to retrieve the absolute orientation, sim_handle_parent to retrieve the orientation relative to the object's parent, or an object handle relative to whose reference frame you want the orientation. **quaternion**: the quaternion (x,y,z,w)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_4 quaternion=sim.getObjectQuaternion(number objectHandle,number relativeToObjectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetObjectSelection / sim.getObjectSelection (remote API equivalent: simxGetObjectSelection)

Description	Retrieves all selected object's handles. See also simGetObjectSelectionSize, sim.addObjectToSelection and sim.removeObjectFromSelection.
C synopsis	simInt simGetObjectSelection(simInt* objectHandles)
C parameters	**objectHandles**: pointer to object handles. Make sure to have at least simGetObjectSelectionSize simInts available
C return value	size of the selection (>=0) if operation was successful, -1 otherwise
Lua synopsis	table selectedObjectHandles=sim.getObjectSelection()
Lua parameters	None
Lua return values	**selectedObjectHandles**: table containing the handles of all selected objects, or nil if no object is selected or in case of an error

simGetObjectSelectionSize

Description	Retrieves the size of the object selection. See also sim.getObjectSelection.
C synopsis	simInt simGetObjectSelectionSize()
C parameters	None
C return value	size of the selection, or -1 if operation was not successful
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetObjectSizeFactor / sim.getObjectSizeFactor

Description	Retrieves the size factor of a scene object. The size factor is different from the real object size. Use this to be able to react to scaling operations. See also sim.getObjectSizeValues.
C synopsis	simFloat simGetObjectSizeFactor(simInt objectHandle)
C parameters	**objectHandle**: handle of the scene object
C return value	size factor or negative value in case of an error

Lua synopsis	number sizeFactor=sim.getObjectSizeFactor(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetObjectSizeValues / sim.getObjectSizeValues

Description	Retrieves the x, y and z size values of a scene object. The size values are different from the real object sizes. Use this to be able to react to scaling operations. See also sim.setObjectSizeValues and sim.getObjectSizeFactor.
C synopsis	simInt simGetObjectSizeValues(simInt objectHandle,simFloat* sizeValues)
C parameters	**objectHandle**: handle of the scene object **sizeValues (output)**: a pointer to 3 size values (x, y and z)
C return value	-1 in case of an error
Lua synopsis	table_3 sizeValues=sim.getObjectSizeValues(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetObjectSpecialProperty / sim.getObjectSpecialProperty

Description	Retrieves the special properties of a scene object. See also sim.setObjectSpecialProperty, sim.getObjectProperty and sim.getModelProperty.
C synopsis	simInt simGetObjectSpecialProperty(simInt objectHandle)
C parameters	**objectHandle**: handle of the object
C return value	object special property values, -1 if operation was not successful
Lua synopsis	number property=sim.getObjectSpecialProperty(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function. See also the boolean operators in Lua.

simGetObjectStringParameter / sim.getObjectStringParameter

Description	Retrieves a string parameter of a scene object or calculation object. See also sim.setObjectStringParameter, sim.getObjectInt32Parameter and sim.getObjectFloatParameter
C synopsis	simChar* simGetObjectStringParameter(simInt objectHandle,simInt parameterID,simInt* parameterLength)
C parameters	**objectHandle**: handle of the object **parameterID**: identifier of the parameter to retrieve. See the list of all possible object parameter identifiers **parameterLength**: the length of the retrieved parameter
C return value	A buffer containing the retrieved string, or NULL in case of an error. The user is in charge of releasing the returned buffer with simReleaseBuffer. The returned buffer might contain embedded zeros, and its length is specified by the parameterLength argument.
Lua synopsis	string parameter=sim.getObjectStringParameter(number objectHandle,number parameterID)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetObjectType / sim.getObjectType

Description	Retrieves the type of an object
C synopsis	simInt simGetObjectType(simInt objectHandle)
C parameters	**objectHandle**: handle of the object
C return value	type of the object (sim_object_shape_type, sim_object_joint_type, etc. (see the object types) or -1 in case of error

Lua synopsis	number objectType=sim.getObjectType(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetObjectUniqueIdentifier / sim.getObjectUniqueIdentifier

Description	Retrieves an object's unique identifier (an object handle is unique, but not across opened scenes. Additionally, if a huge amount of objects are created/destroyed (>2000000), then handles of destroyed objects will be reused. This is not the case with unique identifiers).
C synopsis	simInt simGetObjectUniqueIdentifier(simInt objectHandle,simInt* uniqueIdentifier)
C parameters	**objectHandle**: object handle, or sim_handle_all to retrieve all object identifiers **uniqueIdentifier**: pointer to the unique identifier, or if sim_handle_all is specified as the object handle, then the pointer points to several values. The user is in charge of reserving the buffer (size 1 if handle is specified, or number of objects in the scene if sim_handle_all is specified)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	(1) number uniqueId=sim.getObjectUniqueIdentifier(number objectHandle) (2) table uniqueIds=sim.getObjectUniqueIdentifier(sim.handle_all)
Lua parameters	Same as C-function
Lua return values	(1) **uniqueId**: the unique identifier or nil in case of an error (2) **uniqueIds**: a table containing the unique identifiers or nil in case of an error

simGetObjectVelocity / sim.getObjectVelocity (remote API equivalent: simxGetObjectVelocity)

Description	Retrieves the linear and/or angular velocity of an object, in absolute coordinates. The velocity is a measured velocity (i.e. from one simulation step to the next), and is available for all objects in the scene. See also sim.getVelocity.
C synopsis	simInt simGetObjectVelocity(simInt objectHandle,simFloat* linearVelocity,simFloat* angularVelocity)
C parameters	**objectHandle**: handle of a scene object. **linearVelocity**: pointer to 3 values that will receive the linear velocity. Can be NULL **angularVelocity**: pointer to 3 values that will receive the angular velocity. Can be NULL
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_3 linearVelocity,table_3 angularVelocity=sim.getObjectVelocity(number shapeHandle)
Lua parameters	**objectHandle**: same as C-function
Lua return values	**linearVelocity**: table containing 3 values that represent the linear velocity, or nil in case of an error **angularVelocity**: table containing 3 values that represent the angular velocity, or nil in case of an error

simGetObjects / sim.getObjects (remote API equivalent: simxGetObjects)

Description	Retrieves object handles. Use this in a loop where index starts at 0 and is incremented to get all object handles in the scene. See also sim.getObjectsInTree.
C synopsis	simInt simGetObjects(simInt index,simInt objectType)
C parameters	**index**: object index (not handle!). First object is located at index 0 **objectType**: object type (sim_object_shape_type, sim_object_joint_type, etc. (see the object types) or sim_handle_all for any type of object
C return value	handle of the object or -1 if no object is located at that index or in case of an error
Lua synopsis	number objectHandle=sim.getObjects(number index,number objectType)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetObjectsInTree / sim.getObjectsInTree

Description	Retrieves object handles in a given hierarchy tree. See also sim.getObjects.
C synopsis	simInt* simGetObjectsInTree(simInt treeBaseHandle,simInt objectType,simInt options,simInt* objectCount)
C parameters	**treeBaseHandle**: the handle of the object that describes the hierarchy tree, or sim_handle_scene for all objects in the scene. **objectType**: the object type to retrieve or sim_handle_all for any type of object in the tree **options**: bit-coded: bit0 set (1): exclude the tree base from the returned array bit1 set (2): include in the returned array only the object's first children. If **treeBaseHandle** is sim_handle_scene, then only parentless objects will be included. **objectCount (out value)**: the number of returned object handles
C return value	a pointer to an array containing object handles, or NULL in case of an error. The user is in charge of releasing the returned buffer with simReleaseBuffer.
Lua synopsis	table objects=sim.getObjectsInTree(number treeBaseHandle,number objectType=sim.handle_all, number options=0)
Lua parameters	Same as C-function
Lua return values	Similar as C-function

simGetOctreeVoxels / sim.getOctreeVoxels

Description	Retrieves voxel positions from an octree. See also the other octree related functions.
C synopsis	const float* simGetOctreeVoxels(simInt octreeHandle,simInt* ptCnt,simVoid* reserved)
C parameters	**octreeHandle**: the handle of the octree. See also simGetObjectHandle **ptCnt**: a pointer receiving the number of voxels contained in the returned pointer. **reserved**: reserved for future extensions. Set to NULL
C return value	NULL if operation was not successful or if the octree doesn't contain any voxels. Otherwise a pointer to the voxel X/Y/Z positions, relative to the octree reference frame
Lua synopsis	table voxels=sim.getOctreeVoxels(number octreeHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetOrientationOnPath / sim.getOrientationOnPath

Description	Retrieves the absolute interpolated orientation of a point along a path object. See also sim.getPositionOnPath, sim.getPathPosition and sim.getClosestPositionOnPath.
C synopsis	simInt simGetOrientationOnPath(simInt pathHandle,simFloat relativeDistance,simFloat* eulerAngles)
C parameters	**pathHandle**: handle of the path object **relativeDistance**: a value between 0 and 1, where 0 is the beginning of the path, and 1 the end of the path. Make sure you selected the appropriate path length calculation method (refer to the path position calculation method section). See also simGetPathLength. In order to retrieve the orientation that lies exactly on a specific path control point, specify following for *relativeDistance*: -ctrlPtIndex-1 (the value will be rounded appropriately). **eulerAngles**: Euler angles (alpha, beta and gamma)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_3 eulerAngles=sim.getOrientationOnPath (number pathHandle,number relativeDistance)
Lua parameters	**pathHandle**: handle of the path object **relativeDistance**: a value between 0 and 1, where 0 is the beginning of the path, and 1 the end of the path. Make sure you selected the appropriate path length calculation method. See also sim.getPathLength. In order to retrieve the orientation that lies exactly on a specific path control point, specify following for *relativeDistance*: -ctrlPtIndex-1 (the value will be rounded appropriately).
Lua return values	**eulerAngles**: table of 3 values (alpha, beta and gamma) or nil in case of an error

simGetPage / sim.getPage

Description	Retrieves the current page index (view). See also sim.setPage.
C synopsis	simInt simGetPage()

C parameters	None
C return value	page index or -1 in case of an error
Lua synopsis	number pageIndex=sim.getPage()
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetPathLength / sim.getPathLength

Description	Retrieves the length of a path object. The length is given in meters, but the actual returned length is dependent on the selected path length calculation method for the given path object. See also sim.getPathPosition and sim.setPathPosition.
C synopsis	simInt simGetPathLength(simInt objectHandle,simFloat* length)
C parameters	**objectHandle**: handle of the path object **length**: length of the path given in meters (but dependent on the selected path length calculation method)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number length=sim.getPathLength(number objectHandle)
Lua parameters	**objectHandle**: handle of the path object
Lua return values	**length**: length of the path given in meters (but dependent on the selected path length calculation method), or nil in case of an error

simGetPathPlanningHandle (DEPRECATED)

Description	DEPRECATED. See the OMPL library based path/motion planning functionality instead. Retrieves the handle of a path planning object. The operation of this function depends on the current name suffix settings (see sim.getNameSuffix, sim.setNameSuffix, and the section on accessing general-type objects). See also sim.isHandleValid.
C synopsis	simInt simGetPathPlanningHandle(const simChar* pathPlanningObjectName)
C parameters	**pathPlanningObjectName**: name of the path planning object
C return value	handle of the path planning object or -1 if operation was not successful
Lua synopsis	number pathPlanningObjectHandle=simGetPathPlanningObject(string pathPlanningObjectName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetPathPosition / sim.getPathPosition

Description	Retrieves the intrinsic position of a path object (a distance along the path). The position is given in meters, but the actual returned position is dependent on the selected path length calculation method for the given path object. See also sim.setPathPosition, sim.getPathLength, sim.getPositionOnPath and sim.getClosestPositionOnPath.
C synopsis	simInt simGetPathPosition(simInt objectHandle,simFloat* position)
C parameters	**objectHandle**: handle of the path object **position**: linear position on the path given in meters (but dependent on the selected path length calculation method)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number position=sim.getPathPosition(number objectHandle)
Lua parameters	**objectHandle**: handle of the path object
Lua return values	**position**: linear position on the path given in meters (but dependent on the selected path length calculation method), or nil in case of an error

simGetPointCloudOptions / sim.getPointCloudOptions

Description	Gets various properties of a point cloud. See also sim.setPointCloudOptions and the other point cloud related functions.
C synopsis	simInt simGetPointCloudOptions(simInt pointCloudHandle,simFloat* maxVoxelSize,simInt* maxPtCntPerVoxel,simInt* options,simFloat* pointSize,simVoid* reserved)
C parameters	**pointCloudHandle**: the handle of the point cloud. See also simGetObjectHandle **maxVoxelSize**: the maximum size of the octree voxels containing points **maxPtCntPerVoxel**: the maximum number of points allowed in a same octree voxel **options**: bit-coded: bit0 set (1): points have random colors bit1 set (2): show octree structure bit2 set (4): reserved. keep unset bit3 set (8): do not use an octree structure. When enabled, point cloud operations are limited, and point clouds will not be collidable, measurable or detectable anymore, but adding points will be much faster bit4 set (16): color is emissive **pointSize**: the size of the points, in pixels **reserved**: reserved for future extensions. Set to NULL
C return value	1 if operation was successful
Lua synopsis	number maxVoxelSize,number maxPtCntPerVoxel,number options,number pointSize=sim.getPointCloudOptions(number pointCloudHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetPointCloudPoints / sim.getPointCloudPoints

Description	Retrieves point positions from a point cloud. See also the other point cloud related functions.
C synopsis	const float* simGetPointCloudPoints(simInt pointCloudHandle,simInt* ptCnt,simVoid* reserved)
C parameters	**pointCloudHandle**: the handle of the point cloud. See also simGetObjectHandle **ptCnt**: a pointer receiving the number of points contained in the returned pointer. **reserved**: reserved for future extensions. Set to NULL
C return value	NULL if operation was not successful or if the point cloud doesn't contain any points. Otherwise a pointer to the point X/Y/Z positions, relative to the point cloud reference frame
Lua synopsis	table points=sim.getPointCloudPoints(number pointCloudHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetPositionOnPath / sim.getPositionOnPath

Description	Retrieves the absolute interpolated position of a point along a path object. See also sim.getOrientationOnPath, sim.getDataOnPath, sim.getPathPosition and sim.getClosestPositionOnPath.
C synopsis	simInt simGetPositionOnPath(simInt pathHandle,simFloat relativeDistance,simFloat* position)
C parameters	**pathHandle**: handle of the path object **relativeDistance**: a value between 0 and 1, where 0 is the beginning of the path, and 1 the end of the path. Make sure you selected the appropriate path length calculation method. See also simGetPathLength. In order to retrieve the position that lies exactly on a specific path control point, specify following for *relativeDistance*: -ctrlPtIndex-1 (the value will be rounded appropriately). **position**: pointer to 3 values (x, y and z)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_3 position=sim.getPositionOnPath (number pathHandle,number relativeDistance)
Lua parameters	**pathHandle**: handle of the path object **relativeDistance**: a value between 0 and 1, where 0 is the beginning of the path, and 1 the end of the path. Make sure you selected the appropriate path length calculation method. See also sim.getPathLength. In order to retrieve the position that lies exactly on a specific path control point, specify following for *relativeDistance*: -ctrlPtIndex-1 (the value will be rounded appropriately).
Lua return values	**position**: table of 3 values (x, y and z) or nil in case of an error

simGetQHull / sim.getQHull

Description	Retrieves the convex hull mesh from the specified vertices. See also sim.convexDecompose and sim.getDecimatedMesh.
C synopsis	simInt simGetQHull(const simFloat* inVertices,simInt inVerticesL,simFloat** verticesOut,simInt* verticesOutL,simInt** indicesOut,simInt* indicesOutL,simInt reserved1,const simFloat* reserved2)
C parameters	**inVertices**: a pointer to the input vertices (succession of x/y/z values). **inVerticesL**: the number of input vertices times 3. **verticesOut**: a pointer to a pointer to the output vertices. The output vertices are allocated by V-REP and the user is in charge of releasing the buffer via simReleaseBuffer. **verticesOutL**: a pointer to the number of output vertices times 3. **indicesOut**: a pointer to a pointer to the output indices. The output indices are allocated by V-REP and the user is in charge of releasing the buffer via simReleaseBuffer. **indicesOutL**: a pointer to the number of output indices (i.e. the number of triangles times 3). **reserved1**: reserved, set to 0. **reserved2**: reserved, set to NULL.
C return value	-1 or 0 if operation was not successful.
Lua synopsis	table verticesOut,table indicesOut=sim.getQHull(table verticesIn)
Lua parameters	**verticesIn**: a table containing the input vertices (succession of x/y/z values).
Lua return values	**verticesOut**: a table containing the output vertices (succession of x/y/z values). **indicesOut**: a table containing the output indices (3 values for each triangle).

simGetQuaternionFromMatrix / sim.getQuaternionFromMatrix

Description	Retrieves the quaternion from a transformation matrix. See also sim.getEulerAnglesFromMatrix and the other matrix/transformation functions.
C synopsis	simInt simGetQuaternionFromMatrix(const simFloat* matrix,simFloat* quaternion)
C parameters	**matrix**: pointer to 12 simFloat values (the last row of the 4x4 matrix (0,0,0,1) is not needed) The x-axis of the orientation component is (matrix[0],matrix[4],matrix[8]) The y-axis of the orientation component is (matrix[1],matrix[5],matrix[9]) The z-axis of the orientation component is (matrix[2],matrix[6],matrix[10]) The position component is (matrix[3],matrix[7],matrix[11]) **quaternion**: pointer to 4 simFloat values representing the quaternion in the matrix (x,y,z,w)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_4 quaternion=sim.getQuaternionFromMatrix(table_12 matrix)
Lua parameters	**matrix**: table to 12 numbers (the last row of the 4x4 matrix (0,0,0,1) is not needed). Table values in Lua are indexed from 1, not 0!
Lua return values	**quaternion**: table of 4 numbers representing the quaternion, or nil in case of an error

simGetRealTimeSimulation / sim.getRealTimeSimulation

Description	Indicates whether the simulation is real-time. See also simIsRealTimeSimulationStepNeeded and simAdjustRealTimeTimer.
C synopsis	simInt simGetRealTimeSimulation()
C parameters	None
C return value	1 if simulation is real-time, 0 if it is not, and -1 if the operation was not successful
Lua synopsis	number result=sim.getRealTimeSimulation()
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetReferencedHandles / sim.getReferencedHandles

Description	Retrieves a list of custom handles, linking a given object to other objects. See also

	sim.setReferencedHandles.
C synopsis	simInt simGetReferencedHandles(simInt objectHandle,simInt** referencedHandles,simInt** reserved1,simInt** reserved2)
C parameters	**objectHandle**: handle of the scene object that stores the list of handles **referencedHandles**: a pointer to a pointer that will be allocated and receive the list of handles. The user is in charge of releasing that buffer with simReleaseBuffer. Handles of following object types are supported: scene objects, collision objects, distance objects, IK groups, collections, and geometric constraint solver objects. **reserved1**: reserved for future extensions **reserved2**: reserved for future extensions
C return value	-1 in case of an error. Otherwise, the number of handles returned.
Lua synopsis	table referencedHandles=sim.getReferencedHandles(number objectHandle)
Lua parameters	Similar to the C-function
Lua return values	Similar to the C-function

simGetRotationAxis / sim.getRotationAxis

Description	Retrieves an axis and rotation angle that brings one transformation matrix onto another one. The translation part of the transformation matrices is ignored. This function, when used in combination with sim.rotateAroundAxis, can be used to build interpolations between transformation matrices. See also sim.getObjectMatrix, sim.setObjectMatrix and the other matrix/transformation functions.
C synopsis	simInt simGetRotationAxis(const simFloat* matrixStart,const simFloat* matrixGoal,simFloat* axis,simFloat* angle)
C parameters	**matrixStart**: the *start* transformation matrix **matrixGoal**: the *goal* transformation matrix **axis**: the returned rotation axis in absolute coordinates **angle**: the returned rotation angle
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_3 axis,number angle=sim.getRotationAxis(table_12 matrixStart,table_12 matrixGoal)
Lua parameters	Same as C-function
Lua return values	**axis**: the rotation axis in absolute coordinates, or nil in case of an error **angle**: the rotation angle, or nil in case of an error

simGetScaledImage / sim.getScaledImage

Description	Generates a scaled-up or scaled down version of the input image. See also sim.transformImage, sim.loadImage, sim.saveImage and sim.setVisionSensorCharImage.
C synopsis	simUChar* simGetScaledImage(const simUChar* imageIn,const simInt* resolutionIn,const simInt* resolutionOut,simInt options,simVoid* reserved)
C parameters	**imageIn**: a pointer to rgb or rgba values of the input image. **resolutionIn**: the resolution of the input image. **resolutionOut**: the desired resolution of the output image. The values will be replaced by the effective resolution of the output image **options**: bit-coded: bit0 set (1): the input image is rgba, otherwise it is rgb bit1 set (2): the returned image is rgba, otherwise it is rgb bit2-3: 0:ignore aspect ratio, 4:keep aspect ratio (the effective resolution of the returned image will be different), 8:keep aspect ratio by expanding (the effective resolution of the returned image will be different) bit4 set (16): no smooth transformation **reserved**: Reserved for future extension. Set to NULL.
C return value	NULL if operation was not successful, otherwise a buffer containing the output image data. The user is in charge of releasing the buffer with simReleaseBuffer.
Lua synopsis	string imageOut,table_2 effectiveRolutionOut=sim.getScaledImage(string imageIn,table_2 resolutionIn,table_2 desiredResolutionOut,number options)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetSceneCustomData (DEPRECATED)

Description	DEPRECATED. See sim.readCustomDataBlock instead.

simGetScript

Description	Retrieves script handles. Use this in a loop where index starts at 0 and is incremented to get all script handles. See also sim.getScriptHandle.
C synopsis	simInt simGetScript(simInt index)
C parameters	**index**: script index (not handle). First script is located at index 0
C return value	handle of a script if function was successful and a script exists at the given index, or -1 in case of an error
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetScriptAssociatedWithObject / sim.getScriptAssociatedWithObject

Description	Retrieves a child script's handle based on its associated object. See also sim.getObjectAssociatedWithScript, sim.getCustomizationScriptAssociatedWithObject and sim.associateScriptWithObject.
C synopsis	simInt simGetScriptAssociatedWithObject(simInt objectHandle)
C parameters	**objectHandle**: handle of the object that might have a child script associated
C return value	handle of the child script associated with the object, or -1 if the operation was not successful or the object doesn't have an associated script
Lua synopsis	number scriptHandle=sim.getScriptAssociatedWithObject(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetScriptAttribute / sim.getScriptAttribute

Description	Reads various script attributes or properties. See also sim.setScriptAttribute.
C synopsis	simInt simGetScriptAttribute(simInt scriptHandle,simInt attributeID,simFloat* floatVal,simInt* intOrBoolVal)
C parameters	**scriptHandle**: handle of a script **attributeID**: the script attributeID **floatVal**: pointer to a floating point value, receiving the floating point attribute (if applicable) **intOrBoolVal**: pointer to an integer value, receiving the integer or Boolean attribute (if applicable)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number/boolean attribute=sim.getScriptAttribute(number scriptHandle,number attributeID)
Lua parameters	Same as C-function
Lua return values	**attribute**: the requested attribute value

simGetScriptExecutionCount / sim.getScriptExecutionCount

Description	Retrieves the number of times the current script was called and returned. See also sim.getSimulationState.
C synopsis	-
C parameters	-

C return value	-
Lua synopsis	number executionCount=sim.getScriptExecutionCount()
Lua parameters	None
Lua return values	**executionCount**: number of times the current script was called and returned, or -1 in case of an error

simGetScriptHandle / sim.getScriptHandle

Description	Retrieves the handle of a script. The operation of this function depends on the current name suffix settings (see sim.getNameSuffix, sim.setNameSuffix, and the section on accessing general-type objects). A script doesn't directly have a name assigned, however the script inherits the name of its associated object, if it has one. See also sim.isHandleValid.
C synopsis	simInt simGetScriptHandle(const simChar* scriptName)
C parameters	**scriptName**: name of the script. The name of a child script. If the name is left blank, the handle of the main script is retrieved. In order to retrieve the handle of a customization script, use following scriptName prefix: *customization@* (for instance, *customization@objectName*)
C return value	handle of the script if operation was successful, -1 otherwise
Lua synopsis	number scriptHandle=sim.getScriptHandle(string scriptName)
Lua parameters	Same as C-function. Alternatively, scriptName can be nil or inexistent, in which case the current script's handle is returned
Lua return values	Same as C-function

simGetScriptName / sim.getScriptName

Description	Retrieves a script's name based on its handle. A script doesn't have a name assigned, however if the script is a child script and associated with a scene object, then this function will retrieve the name of the associated scene object. If the script is not a child script or is not associated with a scene object, then the returned value is NULL
C synopsis	simChar* simGetScriptName(simInt scriptHandle)
C parameters	**scriptHandle**: handle of the script
C return value	buffer to the script's name if function was successful and the name is valid, NULL otherwise. The user is in charge of releasing the returned buffer with simReleaseBuffer
Lua synopsis	string scriptName=sim.getScriptName(number scriptHandle)
Lua parameters	**scriptHandle**: handle of the script, or sim.handle_self for the handle of the current script
Lua return values	**scriptName**: name of the script if associated with a scene object, empty string if the script is the main script, or the name of the add-on if the script is an add-on.

simGetScriptProperty

Description	Retrieves properties relative to a script.
C synopsis	simInt simGetScriptProperty(simInt scriptHandle,simInt* scriptProperty,simInt* associatedObjectHandle)
C parameters	**scriptHandle**: handle of the script **scriptProperty**: pointer to a script property value (see the script type values)) **associatedObjectHandle**: pointer to the handle of an associated object if script is a child script, -1 otherwise (if the child script doesn't have an associated object, the value is -1 also)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetScriptRawBuffer

Description	Returns the raw data that is attached to a given script. See also simSetScriptRawBuffer and simReleaseScriptRawBuffer.
C synopsis	simChar* simGetScriptRawBuffer(simInt scriptHandle,simInt bufferHandle)
C parameters	**scriptHandle**: handle of the script **bufferHandle**: handle of the raw buffer
C return value	a pointer to the raw buffer (the buffer is owned by the simulator and will be released by the simulator or through a call to simReleaseScriptRawBuffer) or NULL if the buffer doesn't exist or in case of an error
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetScriptSimulationParameter / sim.getScriptSimulationParameter

Description	Retrieves a main script's or child script's parameter from its simulation parameter list. Useful for simple interaction with the user, or for simple parameter exchange with other scripts. See also sim.setScriptSimulationParameter, and the data packing/unpacking functions.
C synopsis	simChar* simGetScriptSimulationParameter(simInt scriptHandle,const simChar* parameterName,simInt* parameterLength)
C parameters	**scriptHandle**: handle of the main script or child script, or sim_handle_main_script or sim_handle_all. When scriptHandle is sim_handle_all, the function returns only one matching parameter encountered (other matching parameters might be different) **parameterName**: name of the parameter to retrieve **parameterLength**: the number of bytes that compose the value of the parameter (excluding the terminal zero)
C return value	value of the parameter or NULL if parameterName does not exist for the given script, or in case of an error. The user is in charge of releasing the returned value with simReleaseBuffer. The returned pointer points to parameterLength byte values, terminated by a terminal zero (the returned buffer may however contain several embedded zeros).
Lua synopsis	(1) boolean/number/string parameterValue=sim.getScriptSimulationParameter(number scriptHandle,string parameterName,boolean forceStringReturn=false) 2) table parameterValues,table scriptHandles=sim.getScriptSimulationParameter(number targetScripts,string parameterName,boolean forceStringReturn=false)
Lua parameters	(1) **scriptHandle**: handle of the script, or sim.handle_main_script or sim_handle_self. (2) **targetScripts**: sim.handle_all, sim.handle_tree or sim.handle_chain (with sim.handle_tree and sim.handle_chain the calling script is excluded). **parameterName**: name of the parameter to retrieve. **forceStringReturn**: forces the return of a string (i.e. raw value). False by default. If false, then the returned string will be converted to nil, false, true, a number or a string as appropriate (and in that order).
Lua return values	(1) **parameterValue**: value of the parameter, or nil in case of an error (or if that value is nil!). (2) **parameterValues**: table of parameter values or nil if no such parameter was found or in case of an error. **scriptHandles**: table of script handles associated with the parameter values (i.e. parameterValue[i] comes from the script with handle scriptHandles[i]) or nil if no such parameter was found or in case of an error. If the returned parameter value is a string, then it might contain any values (also embedded zeros)

simGetScriptText

Description	Returns the content of a script (i.e. Lua code). See also sim.setScriptText.
C synopsis	const simChar* simGetScriptText(simInt scriptHandle)
C parameters	**scriptHandle**: handle of a script
C return value	pointer to the script buffer (0-terminated buffer), or NULL in case of an error
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetShapeColor / sim.getShapeColor

Description	Retrieves the color of a shapes. See also sim.setShapeColor.
C synopsis	simInt simGetShapeColor(simInt shapeHandle,const simChar* colorName,simInt colorComponent,simFloat* rgbData)
C parameters	**shapeHandle**: handle of the shape **colorName**: name of a color. If a name is provided, a specific color component will be retrieved (e.g. if a shape is a compound shape. Can be NULL. **colorComponent**: a color component. **rgbData** (output): red, green and blue components of the color (3 values), or the transparency value (1 value)
C return value	-1 if operation was not successful. 0 if the color name was not found in the shape. Otherwise, the operation was successful
Lua synopsis	number result,table_3 rgbData=sim.getShapeColor(number shapeHandle,string colorName,number colorComponent)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetShapeGeomInfo / sim.getShapeGeomInfo

Description	Retrieves geometric information related to a shape. See also sim.getShapeMesh.
C synopsis	simInt simGetShapeGeomInfo(simInt shapeHandle,simInt* intData,simFloat* floatData,simVoid* reserved)
C parameters	**shapeHandle**: handle of the shape. See also simGetObjectHandle. **intData (output)**: pointer to 5 integer values: intData[0]: the pure type of the shape. Undefined if the shape is a compound shape. **floatData (output)**: pointer to 5 float values: floatData[0]: X-size or diameter of the pure shape. Undefined if the shape is a compound shape or not pure. floatData[1]: Y-size of the pure shape. Undefined if the shape is a compound shape or not pure. floatData[2]: Z-size or height of the pure shape. Undefined if the shape is a compound shape or not pure. floatData[3]: Inside scaling. Undefined if the shape is a compound shape or not pure. **reserved**: reserved for future extensions. Set to NULL.
C return value	-1 in case of an error, otherwise bit-coded: bit0 set (1): shape is a compound shape bit1 set (2): shape is pure bit2 set (4): shape is convex
Lua synopsis	number result,number pureType,table_4 dimensions=sim.getShapeGeomInfo(number shapeHandle)
Lua parameters	**shapeHandle**: handle of the shape. See also sim.getObjectHandle.
Lua return values	**result**: -1 in case of an error, otherwise bit-coded: bit0 set (1): shape is a compound shape bit1 set (2): shape is pure bit2 set (4): shape is convex **pureType**: the pure type of the shape. Undefined if the shape is a compound shape. **dimensions**: table to 4 values giving information about the shape's dimensions: dimensions[1]: X-size or diameter of the pure shape. Undefined if the shape is a compound shape or not pure. dimensions[2]: Y-size of the pure shape. Undefined if the shape is a compound shape or not pure. dimensions[3]: Z-size or height of the pure shape. Undefined if the shape is a compound shape or not pure. dimensions[4]: Inside scaling. Undefined if the shape is a compound shape or not pure.

simGetShapeMassAndInertia / sim.getShapeMassAndInertia

Description	Retrieves mass and inertia information from a shape. See also sim.setShapeMassAndInertia, sim.getObjectMatrix, sim.buildMatrix and sim.computeMassAndInertia.
C synopsis	simInt simGetShapeMassAndInertia(simInt shapeHandle,simFloat* mass,simFloat* inertiaMatrix,simFloat* centerOfMass,const simFloat* transformation)

C parameters	**shapeHandle**: handle of the shape object **mass**: the mass of the object **inertia matrix** (output): the inertia matrix or tensor (9 values), expressed relative to the center of mass. The returned matrix is relative to the orientational frame of **transformation** (see further below). **centerOfMass** (output): the position of the center of mass, relative to the specified **transformation** (see next item). **transformation**: the transformation matrix (12 values) relative to which we want the data. Can be NULL, in which case the returned data is relative to the absolute reference frame. See here to see how matrix transformations are specified in V-REP.
C return value	-1 in case of an error
Lua synopsis	number mass,table_9 inertiaMatrix,table_3 centerOfMass=sim.getShapeMassAndInertia(number shapeHandle,table_12 transformation=nil)
Lua parameters	See the C-function for details
Lua return values	See the C-function for details

simGetShapeMaterial (DEPRECATED)

Description	DEPRECATED. See sim.setShapeMaterial instead.

simGetShapeMesh / sim.getShapeMesh

Description	Retrieves a shape's mesh information. See also sim.getShapeViz, sim.createMeshShape and sim.exportMesh for a usage example.
C synopsis	simInt simGetShapeMesh(simInt shapeHandle,simFloat** vertices,simInt* verticesSize,simInt** indices,simInt* indicesSize,simFloat** normals)
C parameters	**shapeHandle**: handle of the shape **vertices**: receives the vertices. The user is in charge of destroying the array with simReleaseBuffer. See simExportMesh for a usage example. **verticesSize**: receives the size of the vertices array. See simExportMesh for a usage example. **indices**: receives the indices. The user is in charge of destroying the array with simReleaseBuffer. See simExportMesh for a usage example. **indicesSize**: receives the size of the indice array. See simExportMesh for a usage example. **normals**: receives the normals (3 times the size of indicesSize). The user is in charge of destroying the array with simReleaseBuffer. Can be NULL.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table vertices,table indices,table normals=sim.getShapeMesh(number shapeHandle)
Lua parameters	**shapeHandle**: handle of the shape. See sim.exportMesh for a usage example.
Lua return values	**vertices**: table of vertices, or nil in case of an error **indices**: table of indices, or nil in case of an error **normals**: table of normals, or nil in case of an error

simGetShapeTextureId / sim.getShapeTextureId

Description	Retrieves the texture ID of a texture that is applied to a specific shape. See also sim.getTextureId and sim.setShapeTexture.
C synopsis	simInt simGetShapeTextureId(simInt shapeHandle)
C parameters	**shapeHandle**: handle of the shape.
C return value	The texture ID, or -1 if the texture does not exist or in case of an error
Lua synopsis	number textureId=sim.getShapeTextureId(number shapeHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetShapeViz / sim.getShapeViz

Description	Retrieves a shape's visual information.
C synopsis	simInt simGetShapeViz(simInt shapeHandle,simInt index,struct SShapeVizInfo* info)
C parameters	**shapeHandle**: handle of the shape **index**: 0-based index of the shape element to retrieve (compound shapes contain more than one shape element) **info**: visual information about the shape: **vertices**: array of vertices. The user is in charge of releasing this buffer with simReleaseBuffer if the return value is > 0. **verticesSize**: the size of the vertices array **indices**: array of indices. The user is in charge of releasing this buffer with simReleaseBuffer if the return value is > 0. **indicesSize**: the size of the indices array **normals**: array of normals. The user is in charge of releasing this buffer with simReleaseBuffer if the return value is > 0. **shadingAngle**: the gouraud shading angle **colors[9]**: array of colors: RGB for ambient-diffuse, specular, and emission. **texture**: the RGBA texture (32bit/pixel). The user is in charge of releasing this buffer with simReleaseBuffer if the return value is > 1. **textureId**: a texture id (to identify textures shared among several shapes) **textureRes[2]**: the resolution of the texture **textureCoords**: the texture coordinates. The user is in charge of releasing this buffer with simReleaseBuffer if the return value is > 1. **textureApplyMode**: 0=modulate, 1=decal, 2=add **textureOptions**: bit-coded: bit0 set (1): repeat U bit1 set (2): repeat V bit2 set (4): interpolate colors
C return value	-1 if operation was not successful, 0 if there is no shape element at the given indexm, 1 if the shape element does not contain any texture, 2 if the shape element contains a texture
Lua synopsis	map data=sim.getShapeViz(number shapeHandle,number index)
Lua parameters	**shapeHandle**: handle of the shape. **index**: 0-based index of the shape element to retrieve (compound shapes contain more than one shape element)
Lua return values	In case of success, data contains following fields: **data.vertices**: array containing the vertices **data.indices**: array containing the indices **data.normals**: array containing the normals **data.colors**: array containing the colors: RGB for ambient-diffuse, specular, and emission. **data.shadingAngle**: the gouraud shading angle Additionally, if the shape element contains a texture, data contains following additional fields: **data.texture.texture**: the string-coded RGBA texture. See also sim.transformBuffer **data.texture.id**: a texture id (to identify textures shared among several shapes) **data.texture.resolution**: the resolution of the texture **data.texture.coordinates**: the texture coordinates **data.texture.applyMode**: 0=modulate, 1=decal, 2=add **data.texture.options**: bit-coded: bit0 set (1): repeat U bit1 set (2): repeat V bit2 set (4): interpolate colors

simGetSignalName / sim.getSignalName

Description	Returns the signal name at the given index. Use this function in a loop until return is NULL to read all set signals. Signals are values that are global to a given simulator scene and cleared at simulation start. See also the other signal functions.
C synopsis	simChar* simGetSignalName(simInt signalIndex,simInt signalType)
C parameters	**signalIndex**: zero based index **signalType**: signal type. 0 is for integer signals, 1 for float signals and 2 for string signals
C return value	NULL if operation was not successful or signal does not exist at this index, otherwise the name of the signal at the given index (the user is in charge of releasing the returned buffer with simReleaseBuffer)
Lua synopsis	string signalName=sim.getSignalName(number signalIndex,number signalType)

Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetSimulationPassesPerRenderingPass

Description	Returns the number of simulation passes (calculation passes) per frame (display). This value might not be constant for a given simulation. See also simSetSimulationPassesPerRenderingPass.
C synopsis	simInt simGetSimulationPassesPerRenderingPass()
C parameters	None
C return value	>0 if operation was successful (the number of simulation passes per rendering pass), -1 otherwise
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetSimulationState / sim.getSimulationState

Description	Retrieves current simulation state. See also the simulation state diagram.
C synopsis	simInt simGetSimulationState()
C parameters	None
C return value	The current state of the simulation (sim_simulation_stopped, sim_simulation_paused, etc. (see the simulation state values)), or -1 in case of an error
Lua synopsis	number simulationState=sim.getSimulationState()
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetSimulationTime / sim.getSimulationTime

Description	Retrieves the current simulation time
C synopsis	simFloat simGetSimulationTime()
C parameters	None
C return value	negative value (-1.0) if operation not successful, otherwise the simulation time
Lua synopsis	number simulationTime=getSimulationTime()
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetSimulationTimeStep / sim.getSimulationTimeStep

Description	Retrieves the simulation time step (the simulation time (i.e. not real-time) that passes at each main script simulation pass). This value might not be constant for a given simulation.
C synopsis	simFloat simGetSimulationTimeStep()
C parameters	None
C return value	negative value (-1.0) if operation not successful, otherwise the simulation time step
Lua synopsis	number timeStep=getSimulationTimeStep()
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetSimulatorMessage / sim.getSimulatorMessage

Description	Retrieves and removes the next message in the C/C++ or Lua message queues. Use this in a while-loop until all messages have been extracted. While the C/C++ interface has one single message queue, each Lua script has its own message queue. The C/C++ version of this function should only be called from the V-REP client application. A given message queue cannot hold more than 64 messages, unread messages will be discarded.
C synopsis	simChar* simGetSimulatorMessage(simInt* messageID,simInt* auxiliaryData,simInt* returnedDataSize)
C parameters	**messageID**: a simulator message (see the simulator messages) or -1 if no message is available or in case of an error **auxiliaryData**: table of 4 integers that can describe the returned message in more details **returnedDataSize**: size of the returned buffer
C return value	NULL if no buffer was returned, otherwise a buffer that should be released with simReleaseBuffer
Lua synopsis	number message,table_4 auxiliaryData,table auxiliaryData2=sim.getSimulatorMessage()
Lua parameters	None
Lua return values	**message**: a simulator message (see the simulator messages) or -1 if no message is available or in case of an error **auxiliaryData**: table of 4 numbers that can describe the returned message in more details **auxiliaryData2**: optional table of n numbers that hold more data related to the message

simGetStackBoolValue

Description	Tries to retrieve the value at the top of the stack, if that value is a Boolean. See also the other stack functions.
C synopsis	simInt simGetStackBoolValue(simInt stackHandle,simBool* boolValue)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **boolValue**: a pointer to a location receiving the bool value.
C return value	-1 in case of an error, 0 if the value is not a bool, 1 otherwise.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetStackDoubleTable

Description	Retrieves a double-precision float array from an array-type table at the top of the stack. Table values that are not numbers are converted to 0.0. See also simGetStackTableInfo and the other stack functions.
C synopsis	simInt simGetStackDoubleTable(simInt stackHandle,simDouble* array,simInt count)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **array**: a pointer to a location receiving the double values. Use simGetStackTableInfo to determine the number of values the table contains. **count**: the size of the array. If the array is bigger than the table, it will be padded with 0.0.
C return value	-1 in case of an error, 0 if item is not an array or does not contain only numbers, 1 otherwise.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetStackDoubleValue

Description	Tries to retrieve the value at the top of the stack, if that value is a number. See also the other stack functions.
C synopsis	simInt simGetStackDoubleValue(simInt stackHandle,simDouble* numberValue)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **numberValue**: a pointer to a location receiving the double value.
C return value	-1 in case of an error, 0 if the value is not a number, 1 otherwise.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetStackFloatTable

Description	Retrieves an float array from an array-type table at the top of the stack. Table values that are not numbers are converted to 0.0. See also simGetStackTableInfo and the other stack functions.
C synopsis	simInt simGetStackFloatTable(simInt stackHandle,simFloat* array,simInt count)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **array**: a pointer to a location receiving the float values. Use simGetStackTableInfo to determine the number of values the table contains. **count**: the size of the array. If the array is bigger than the table, it will be padded with 0.0.
C return value	-1 in case of an error, 0 if item is not an array or does not contain only numbers, 1 otherwise.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetStackFloatValue

Description	Tries to retrieve the value at the top of the stack, if that value is a number. See also the other stack functions.
C synopsis	simInt simGetStackFloatValue(simInt stackHandle,simFloat* numberValue)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **numberValue**: a pointer to a location receiving the float value.
C return value	-1 in case of an error, 0 if the value is not a number, 1 otherwise.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetStackInt32Table

Description	Retrieves an integer array from an array-type table at the top of the stack. Table values that are not numbers are converted to 0. See also simGetStackTableInfo and the other stack functions.
C synopsis	simInt simGetStackInt32Table(simInt stackHandle,simInt* array,simInt count)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **array**: a pointer to a location receiving the integer values. Use simGetStackTableInfo to determine the number of values the table contains. **count**: the size of the array. If the array is bigger than the table, it will be padded with 0.
C return value	-1 in case of an error, 0 if item is not an array or does not contain only numbers, 1 otherwise.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetStackInt32Value

Description	Tries to retrieve the value at the top of the stack, if that value is a number. See also the other stack functions.
C synopsis	simInt simGetStackInt32Value(simInt stackHandle,simInt* numberValue)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **numberValue**: a pointer to a location receiving the int32 value.
C return value	-1 in case of an error, 0 if the value is not a number, 1 otherwise.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetStackSize

Description	Returns the size of the stack. See also the other stack functions.
C synopsis	simInt simGetStackSize(simInt stackHandle)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack.
C return value	-1 in case of an error, otherwise the size of the stack.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetStackStringValue

Description	Tries to retrieve the value at the top of the stack, if that value is a string. See also the other stack functions.
C synopsis	simChar* simGetStackStringValue(simInt stackHandle,simInt* stringSize)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **stringSize**: a pointer to a location receiving the size of the string. Can be NULL if the string size is of no interest.
C return value	In case of an error, the return value is NULL and *stringSize* will be set to -1 (if *stringSize* is not NULL). If the stack item is not a string, the return value is NULL and *stringSize* will be set to 0 (if *stringSize* is not NULL). If the stack item is a string, the return value is not NULL, and *stringSize* will be the size of the string buffer (if *stringSize* is not NULL). In that case, the user is in charge of releasing the returned buffer with simReleaseBuffer.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetStackTableInfo

Description	Tries to retrieve information about a possible table at the top of the stack. See also the other stack functions.
C synopsis	simInt simGetStackTableInfo(simInt stackHandle,simInt infoType)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **infoType**: the type of information desired: **0**: whether we have a table, whether the table is an array-type or map-type, and the size of the array-type table **1**: whether the table contains only null values. **2**: whether the table contains only number values. **3**: whether the table contains only Boolean values. **4**: whether the table contains only string values. **5**: whether the table contains only table values.
C return value	-1 in case of an error, otherwise one of following: if *infoType* is 0: *sim_stack_table_not_table* (-3): the value at the top of the stack is not a table *sim_stack_table_map* (-2): the value at the top of the stack is a map-type table, containing value-key pairs, where the keys are not all numbers, not all consecutive, or not starting at 1. Use simUnfoldStackTable to read the content of the table. *sim_stack_table_empty* (0): the value at the top of the stack is an empty table. a value>0: the value at the top of the stack is an array-type table, containing value-key pairs, where all the keys are numbers, consecutive, and starting at 1. Use simUnfoldStackTable, simGetStackUInt8Table, simGetStackInt32Table, simGetStackFloatTable or simGetStackDoubleTable to read the content of the table. if *infoType* is 1: 1 if the table contains only null values if *infoType* is 2: 1 if the table contains only number values if *infoType* is 3: 1 if the table contains only Boolean values if *infoType* is 4: 1 if the table contains only string values

	if *infoType* is 5: 1 if the table contains only table values
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetStackUInt8Table

Description	Retrieves a uint8 array from an array-type table at the top of the stack. Table values that are not numbers are converted to 0. See also simGetStackTableInfo and the other stack functions.
C synopsis	simInt simGetStackUInt8Table(simInt stackHandle,simUChar* array,simInt count)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **array**: a pointer to a location receiving the uint8 values. Use simGetStackTableInfo to determine the number of values the table contains. **count**: the size of the array. If the array is bigger than the table, it will be padded with 0.
C return value	-1 in case of an error, 0 if item is not an array or does not contain only numbers, 1 otherwise.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetStringParameter / sim.getStringParameter (remote API equivalent: simxGetStringParameter)

Description	Retrieves a string value. See the string parameter identifiers. See also sim.getBoolParameter, sim.getInt32Parameter, sim.getArrayParameter and sim.getFloatParameter.
C synopsis	simChar* simGetStringParameter(simInt parameter)
C parameters	**parameter**: string parameter identifier
C return value	NULL if operation was not successful. Otherwise the string parameter. The user is in charge of releasing the returned buffer with simReleaseBuffer.
Lua synopsis	string parameterValue=sim.getStringParameter(number parameter)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetStringSignal / sim.getStringSignal (remote API equivalent: simxGetStringSignal)

Description	Gets the value of a string signal. Signals are cleared at simulation start. See also sim.setStringSignal, the other signal functions, the data packing/unpacking functions and sim.persistentDataRead.
C synopsis	simChar* simGetStringSignal(const simChar* signalName,simInt* stringLength)
C parameters	**signalName**: name of the signal **stringLength**: the size of the returned string, since it may contain any data (also embedded zeros).
C return value	NULL if operation was not successful or signal does not exist, otherwise the value of the string signal (which may contain any value, including embedded zeros). In that case the returned buffer should be released with simReleaseBuffer
Lua synopsis	string signalValue=sim.getStringSignal(string signalName)
Lua parameters	**signalName**: name of the signal
Lua return values	**signalValue**: value of the signal, or nil if operation was not successful or signal does not exist. The returned signal may contain any value, including embedded zeros.

simGetSystemTime / sim.getSystemTime

Description	Retrieves the system time. The system time is the time is seconds that elapsed since Windows or V-REP was started, depending on the system. See also sim.getSystemTimeInMs.
C synopsis	simFloat simGetSystemTime()
C parameters	None

C return value	system time in seconds, or a negative value (-1.0) in case of an error
Lua synopsis	number systemTime=sim.getSystemTime()
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetSystemTimeInMilliseconds (DEPRECATED)

Description	DEPRECATED. See sim.getSystemTimeInMs instead.

simGetSystemTimeInMs / sim.getSystemTimeInMs

Description	Retrieves a time in milliseconds.
C synopsis	simUInt simGetSystemTimeInMs(simInt previousTime)

C parameters	**previousTime**: value that indicates how the command should operate:
	>=0: the function returns a time difference with previousTime. PreviousTime must have been previously retrieved with the -1 argument below.
	-1: the function returns a time relative to an arbitrary time. Use this to measure time differences within V-REP
	-2: the function returns a time as follows:
	
	`// On Windows:`
	`returnedValue=TimeGetTime();`
	
	`// On MacOS / Linux:`
	`struct timeval now;`
	`gettimeofday(&now,NULL);`
	`returnValue=now.tv_sec*1000+now.tv_usec/1000;`

C return value	a time in milliseconds as described here above.
Lua synopsis	number systemTimeOrTimeDiff=sim.getSystemTimeInMs(number previousTime)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetTextureId / sim.getTextureId

Description	Retrieves the texture ID of a specific texture. See also sim.readTexture, sim.writeTexture and sim.createTexture.
C synopsis	simInt simGetTextureId(const simChar* textureName,simInt* resolution)
C parameters	**textureName**: the name of the texture ID to be retrieved. **resolution**: a pointer to 2 integer values representing the resolution of the texture. Can be NULL.
C return value	The texture ID, or -1 if the texture does not exist or in case of an error
Lua synopsis	number textureId,table_2 resolution=sim.getTextureId(string textureName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetThreadAutomaticSwitch / sim.getThreadAutomaticSwitch

Description	Queries whether the current thread will eventually automatically switch to another thread. If the current script doesn't run in a thread (i.e. if it runs in the application main thread), this function always return false. See also sim.setThreadAutomaticSwitch.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	boolean result=sim.getThreadAutomaticSwitch()

Lua parameters	-
Lua return values	**result**: if true, the thread is able to automatically switch to another thread.

simGetThreadId / sim.getThreadId

Description	Returns a thread id. See also simLockResources and simUnlockResources.
C synopsis	simInt simGetThreadId()
C parameters	-
C return value	-1 in case of an error, otherwise the thread id: 0 if the thread is the gui thread, 1 if the thread is the main simulation thread, n if the thread is an auxiliary simulation thread.
Lua synopsis	number threadId=sim.getThreadId()
Lua parameters	-
Lua return values	Same as C-function

simGetUIButtonLabel (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simGetUIButtonProperty (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simGetUIButtonSize (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simGetUIEventButton (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simGetUIHandle (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simGetUIPosition (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simGetUIProperty (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simGetUISlider (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simGetUInt64Parameter

Description	Retrieves an unsigned 64bit integer value. See the uint64 parameter identifiers.
C synopsis	simUInt64 simGetUInt64Parameter(simInt parameter,simUInt64* intState)
C parameters	**parameter**: uint64 parameter identifier **intState**: value of the parameter
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetVelocity / sim.getVelocity

Description	Retrieves the linear and/or angular velocity of the center of mass of a non-static shape object. See also sim.getObjectVelocity.
C synopsis	simInt simGetVelocity(simInt shapeHandle,simFloat* linearVelocity,simFloat* angularVelocity)
C parameters	**shapeHandle**: handle of a dynamically enabled shape **linearVelocity**: pointer to 3 values that will receive the linear velocity in absolute coordinates. Can be NULL **angularVelocity**: pointer to 3 values that will receive the angular velocity in absolute coordinates. Can be NULL
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_3 linearVelocity,table_3 angularVelocity=sim.getVelocity(number shapeHandle)
Lua parameters	**shapeHandle**: handle of a dynamically enabled shape
Lua return values	**linearVelocity**: table containing 3 values that represent the linear velocity in absolute coordinates, or nil in case of an error **angularVelocity**: table containing 3 values that represent the angular velocity in absolute coordinates, or nil in case of an error

simGetVisionSensorCharImage / sim.getVisionSensorCharImage (remote API equivalent: simxGetVisionSensorImage)

Description	Retrieves the rgb-image (or rgba, or a portion of it) of a vision sensor. The returned data doesn't make sense if sim.handleVisionSensor wasn't called previously (sim.handleVisionSensor is called by default in the main script if the vision sensor is not tagged as *explicit handling*). See also sim.getVisionSensorImage, sim.setVisionSensorCharImage and sim.saveImage.
C synopsis	simUChar* simGetVisionSensorCharImage(simInt sensorHandle,simInt* resolutionX,simInt* resolutionY)
C parameters	**sensorHandle**: handle of the vision sensor. Can be combined with sim_handleflag_greyscale (simply add sim_handleflag_greyscale to sensorHandle), if you wish to retrieve the grey scale equivalent. **resolutionX/resolutionY**: the returned vision sensor resolution
C return value	image buffer (buffer size is resolutionX*resolution*3 or resolutionX*resolutionY in case of a grey scale image retrieval) or NULL in case of an error. The user is in charge of releasing the returned buffer with simReleaseBuffer. Returned values are in the range of 0-255 (0=min. intensity, 255=max. intensity)
Lua synopsis	string imageBuffer,number resolutionX,number resolutionY=sim.getVisionSensorCharImage(number

	sensorHandle,number posX=0,number posY=0,number sizeX=0,number sizeY=0,number rgbaCutOff=0)
Lua parameters	**sensorHandle**: handle of the vision sensor. Can be combined with sim.handleflag_greyscale (simply add sim.handleflag_greyscale to sensorHandle), if you wish to retrieve the grey scale equivalent. **posX / posY**: position of the image portion to retrieve. Zero by default. **sizeX / sizeY**: size of the image portion to retrieve. Zero by default, which means that the full image should be retrieved **rgbaCutOff**: when different from zero, then an RGBA image is returned, where the alpha component will be 255 for all depth values below *rgbaCutOff*, and 0 for all depth values above *rgbaCutOff*. 0 corresponds to the near clipping plane, 1 to the far clipping plane. Zero by default.
Lua return values	**imageBuffer**: nil in case of an error. Otherwise a string containing rgb (or rgba) values (table size is sizeX*sizeY*3 (or sizeX*sizeY*4 in case of rgba), rgb(a) values in the range 0-255). In case of a grey scale image retrieval, the image buffer will contain grey values or grey+alpha values in the range 0-255.

simGetVisionSensorDepthBuffer / sim.getVisionSensorDepthBuffer (remote API equivalent: simxGetVisionSensorDepthBuffer)

Description	Retrieves the depth buffer (or a portion of it) of a vision sensor. Use sim.getVisionSensorResolution to know the resolution of the full depth buffer. The returned data doesn't make sense if sim.handleVisionSensor wasn't called previously (sim.handleVisionSensor is called by default in the main script if the vision sensor is not tagged as *explicit handling*).
C synopsis	simFloat* simGetVisionSensorDepthBuffer(simInt sensorHandle)
C parameters	**sensorHandle**: handle of the vision sensor
C return value	depth buffer (buffer size is resolutionX*resolutionY) or NULL in case of an error. The user is in charge of releasing the returned buffer with simReleaseBuffer. Returned values are in the range of 0-1 (0=closest to sensor (i.e. close clipping plane), 1=farthest from sensor (i.e. far clipping plane))
Lua synopsis	table depthBuffer=sim.getVisionSensorDepthBuffer(number sensorHandle,number posX=0,number posY=0,number sizeX=0,number sizeY=0)
Lua parameters	**sensorHandle**: handle of the vision sensor. Can be combined with sim.handleflag_codedstring (simply add sim.handleflag_codedstring to sensorHandle), if you wish to retrieve a string buffer (sim.buffer_float) instead of a table. In that case, refer also to sim.transformBuffer. **posX / posY**: position of the depth buffer portion to retrieve. Zero by default. **sizeX / sizeY**: size of the depth buffer portion to retrieve. Zero by default, which means that the full depth buffer should be retrieved
Lua return values	**depthBuffer**: table containing depth values (table size is sizeX*sizeY), a string containing coded depth values, or nil in case of an error. Returned values are in the range of 0-1 (0=closest to sensor (i.e. close clipping plane), 1=farthest from sensor (i.e. far clipping plane))

simGetVisionSensorFilter / sim.getVisionSensorFilter

Description	Retrieves the parameters and settings of a specific filter component of a vision sensor. See also sim.setVisionSensorFilter and the other vision sensor related API functions.
C synopsis	simInt simGetVisionSensorFilter(simInt visionSensorHandle,simInt filterIndex,simInt* options,simInt* pSizes,simUChar** bytes,simInt** ints,simFloat** floats,simUChar** custom)
C parameters	**visionSensorHandle**: handle of a vision sensor. See also simGetObjectHandle. **filterIndex**: the zero-based index of the filter position. **options**: bit-coded return value: bit 0 set (1): the component is enabled **pSizes**: a pointer to 4 integer values receiving the sizes of the returned buffers (see next 4 arguments). **bytes**: a buffer of bytes values representing the byte parameters of the filter component. The user is in charge of releasing that buffer with simReleaseBuffer. **ints**: a buffer of ints values representing the int parameters of the filter component. The user is in charge of releasing that buffer with simReleaseBuffer. **floats**: a buffer of floats values representing the float parameters of the filter component. The user is in charge of releasing that buffer with simReleaseBuffer. **custom**: a buffer of bytes values representing the custom parameters of the filter component. The user is in charge of releasing that buffer with simReleaseBuffer. USAGE EXAMPLE: `int options=0;`

```
int sizes[4]={0,0,0,0};
unsigned char* bytes;
int* ints;
float* floats;
unsigned char* custom;
int filterType=simGetVisionSensorFilter(handle,index,&options,sizes,&bytes,&ints,&floats,&custom);
if (filterType>0)
{
    // Modify options, bytes, ints, floats and custom
    // ...
    // Now write back the updated values:
    simSetVisionSensorFilter(handle,index,options,sizes,bytes,ints,floats,custom);
    // Destroy the buffers:
    simReleaseBuffer((simChar*)bytes);
    simReleaseBuffer((simChar*)ints);
    simReleaseBuffer((simChar*)floats);
    simReleaseBuffer((simChar*)custom);
}
```

C return value	-1 in case of an error, 0 if the *filterIndex* is not valid, otherwise the type of filter component pointed by the *filterIndex*.
Lua synopsis	number filterType,number options,table byteVals,table intVals,table floatVals,string customBuffer=sim.getVisionSensorFilter(number sensorHandle,number filterIndex)
Lua parameters	Same as C-function
Lua return values	Similar as C-function

USAGE EXAMPLE:

```
local filterType,options,bytes,ints,floats,buffer=sim.getVisionSensorFilter(handle,index)
if filterType>0 then
    -- Modify options, bytes, ints, floats and buffer
    -- ...
    -- Now write back the updated values:
    sim.setVisionSensorFilter(handle,index,options,bytes,ints,floats,buffer)
end
```

simGetVisionSensorImage / sim.getVisionSensorImage (remote API equivalent: simxGetVisionSensorImage)

Description	Retrieves the rgb-image (or a portion of it) of a vision sensor. Use sim.getVisionSensorResolution to know the resolution of the full image. The returned data doesn't make sense if sim.handleVisionSensor wasn't called previously (sim.handleVisionSensor is called by default in the main script if the vision sensor is not tagged as *explicit handling*). See also sim.getVisionSensorCharImage and sim.setVisionSensorImage.
C synopsis	simFloat* simGetVisionSensorImage(simInt sensorHandle)
C parameters	**sensorHandle**: handle of the vision sensor. Can be combined with sim_handleflag_greyscale (simply add sim_handleflag_greyscale to sensorHandle), if you wish to retrieve the grey scale equivalent.
C return value	image buffer (buffer size is resolutionX*resolutionY*3 or resolutionX*resolutionY in case of a grey scale image retrieval) or NULL in case of an error. The user is in charge of releasing the returned buffer with simReleaseBuffer. Returned values are in the range of 0-1 (0=min. intensity, 1=max. intensity)
Lua synopsis	table/string imageBuffer=sim.getVisionSensorImage(number sensorHandle,number posX=0,number posY=0,number sizeX=0,number sizeY=0,number returnType=0)
Lua parameters	**sensorHandle**: handle of the vision sensor. Can be combined with sim.handleflag_greyscale (simply add sim.handleflag_greyscale to sensorHandle), if you wish to retrieve the grey scale equivalent. **posX / posY**: position of the image portion to retrieve. Zero by default. **sizeX / sizeY**: size of the image portion to retrieve. Zero by default, which means that the full image should be retrieved **returnType**: the type of the returned buffer. 0 returns a table filled with rgb values in the range 0-1, 1 returns a string filled with rgb values in the range 0-255
Lua return values	**imageBuffer**: nil in case of an error. Otherwise a table containing rgb values (table size is sizeX*sizeY*3, rgb values in the range 0-1) or a string containing rgb values (table size is sizeX*sizeY*3, rgb values in the range 0-255). In case of a grey scale image retrieval, the image buffer will contain grey values or grey-alpha values.

simGetVisionSensorResolution / sim.getVisionSensorResolution

Description	Retrieves the resolution at which the given vision sensor operates (this might be different from what is indicated in the vision sensor dialog: should your graphic card model be rather old, then only resolutions a 2n will be supported). Useful in combination with sim.getVisionSensorImage/sim.getVisionSensorDepthBuffer
C synopsis	simInt simGetVisionSensorResolution(simInt sensorHandle,simInt* resolution)
C parameters	**sensorHandle**: handle of the vision sensor **resolution**: 2 values for the x and y component
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_2 resolution=sim.getVisionSensorResolution(number sensorHandle)
Lua parameters	**sensorHandle**: handle of the vision sensor
Lua return values	**resolution**: table containing the x and y resolution, or nil in case of an error

simGroupShapes / sim.groupShapes

Description	Groups (or merges) several shapes into a compound shape (or simple shape). See also sim.ungroupShape.
C synopsis	simInt simGroupShapes(const simInt* shapeHandles,simInt shapeCount)
C parameters	**shapeHandles**: the handles of the shapes you wish to group **shapeCount**: the size of the **shapeHandles** array. A negative number indicates that we want to merge the shapes instead of grouping them.
C return value	-1 if operation was not successful. Otherwise the handle of the resulting compound shape.
Lua synopsis	number shapeHandle=sim.groupShapes(table shapeHandles)
Lua parameters	Similar to C-function
Lua return values	Similar to C-function

simHandleChildScript (REPLACED)

Description	This function doesn't exist anymore since V-REP 3.1.3, and you should use sim.handleChildScripts instead. Scenes saved with V-REP versions prior to V-REP 3.1.3 are normally automatically adjusted.

simHandleChildScripts / sim.handleChildScripts

Description	Executes non-threaded child scripts located in the current scene hierarchy branch. Child scripts will be executed in a cascaded fashion and this command should only be called from the main script.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number executedScriptCount=sim.handleChildScripts(number systemCallbackType)
Lua parameters	**callType**: the desired system call type for the child scripts (e.g. *sysCall_actuation*).
Lua return values	executedScriptCount: number of executed non-threaded child scripts

simHandleCollision / sim.handleCollision

Description	Handles (check for collision, etc.) a registered collision object. Collision objects can be registered while editing a scene. See also sim.readCollision, sim.resetCollision, sim.checkCollision and sim.checkCollisionEx.
C synopsis	simInt simHandleCollision(simInt collisionObjectHandle)
C parameters	**collisionObjectHandle**: handle of the collision object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will handle all registered collision objects, while sim_handle_all_except_explicit will only handle those that are not marked as "explicit handling")
C return value	number of collisions or -1 if operation was not successful

Lua synopsis	number collisionCount,table_2 collidingObjectHandles=sim.handleCollision(number collisionObjectHandle)
Lua parameters	Same as C-function
Lua return values	**collisionCount**: number of collisions or -1 if operation was not successful. **collidingObjectHandles**: handles of the two colliding objects. This return value is only available when a collision object handle is provided (i.e. explicit handling).

simHandleDistance / sim.handleDistance

Description	Handles (measures distances, etc.) a registered distance object. Distance objects can be registered while editing a scene. See also sim.readDistance, sim.resetDistance and sim.checkDistance.
C synopsis	simInt simHandleDistance(simInt distanceObjectHandle,simFloat* smallestDistance)
C parameters	**distanceObjectHandle**: handle of the distance object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will handle all registered distance objects, while sim_handle_all_except_explicit will only handle those that are not marked as "explicit handling") **smallestDistance**: smallest measured distance. Can be NULL
C return value	1 if at least one distance was measured, 0 if no distance was measured, -1 in case of an error
Lua synopsis	number result,number smallestDistance=sim.handleDistance(number distanceObjectHandle)
Lua parameters	Same as C-function
Lua return values	**result**: 1 if at least one distance was measured, 0 if no distance was measured, -1 in case of an error **smallestDistance**: the smallest distance measured. Is nil if result is not 1

simHandleDynamics / sim.handleDynamics

Description	Handles the dynamics functionality in a scene. This function is not available to add-ons.
C synopsis	simInt simHandleDynamics(simFloat deltaTime)
C parameters	**deltaTime**: the time that passed since the command was called last. Typically simGetSimulationTimeStep()
C return value	-1 if operation was not successful. 0 if the sepcified physics engine could not be found, otherwise, the number of calculation steps performed by the physics engine.
Lua synopsis	number result=sim.handleDynamics(number deltaTime)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simHandleGeneralCallbackScript (DEPRECATED)

Description	DEPRECATED. Use customization scripts instead.

simHandleGraph / sim.handleGraph

Description	Handles a graph object (i.e. records current values of registered data streams). Graphs and data streams can be added/registered while editing a scene. See also sim.resetGraph.
C synopsis	simInt simHandleGraph(simInt graphHandle,simFloat simulationTime)
C parameters	**graphHandle**: handle of the graph object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will handle all graph objects, while sim_handle_all_except_explicit will only handle those that are not marked as "explicit handling") **simulationTime**: simulation time. Usually you want to record data stream at the end of a simulation pass to record actualized value: then set simulationTime to simGetSimulationTime()+simGetSimulationTimeStep().
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.handleGraph(number graphHandle,number simulationTime)
Lua parameters	Same as C-function

Lua return values	Same as C-function

simHandleIkGroup / sim.handleIkGroup

Description	Handles (solves) a registered IK group. IK groups can be registered while editing a scene. See also sim.checkIkGroup, sim.computeJacobian and sim.generateIkPath.
C synopsis	simInt simHandleIkGroup(simInt ikGroupHandle)
C parameters	**ikGroupHandle**: handle of the IK group or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will handle all IK groups, while sim_handle_all_except_explicit will only handle those that are not marked as "explicit handling"). See also simGetIkGroupHandle.
C return value	number of performed calculations (i.e. IK group calculation results are different from sim_ikresult_not_performed) if no specific IK group was specified, or a value of type IK result if a specific IK group was specified, -1 in case of an error (a failed IK group calculation is not considered as an error)
Lua synopsis	number calculationCountOrResult=sim.handleIkGroup(number ikGroupHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simHandleJoint (DEPRECATED)

Description	DEPRECATED. Instead of using this function for a joint in motion mode, set the joint into passive mode and handle its position update inside of a child script.

simHandleMainScript

Description	Handles (executes) the main script, i.e. the main simulation loop.
C synopsis	simInt simHandleMainScript()
C parameters	None
C return value	A main script execution result. If the return value contains sim_script_main_script_not_called, then the main script was not called (e.g. because a plugin hindered it when it received the sim_message_eventcallback_mainscriptabouttobecalled message). Otherwise, the main script was called and simAdvanceSimulationByOneStep should be executed.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simHandleMechanism / sim.handleMechanism

Description	Handles a mechanism registered with the geometric constraint solver functionality. Mechanisms can be registered while editing a scene.
C synopsis	simInt simHandleMechanism(simInt mechanismHandle)
C parameters	**mechanismHandle**: handle of the mechanism or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will handle all registered mechanisms, while sim_handle_all_except_explicit will only handle those that are not marked as "explicit handling")
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.handleMechanism(number mechanismHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simHandleMill / sim.handleMill

Description	Handles (performs cutting) a registered mill object. See also sim.resetMill.
C synopsis	simInt simHandleMill(simInt millHandle,simFloat* removedSurfaceAndVolume)
C parameters	**millHandle**: handle of a mill object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will handle all mill objects, while sim_handle_all_except_explicit will only handle those that are not marked as "explicit handling") **removedSurfaceAndVolume**: pointer to two floating point values indicating the total removed surface and volume by this call. Can be NULL
C return value	total number of cut objects, or -1 in case of an error
Lua synopsis	number cutCount,table_2 removedSurfaceAndVolume=sim.handleMill(number millHandle)
Lua parameters	Same as C-function
Lua return values	**cutCount**: total number of cut objects, or -1 in case of an error **removedSurfaceAndVolume**: table indicating the total removed surface and volume by this call

simHandleModule / sim.handleModule

Description	Handles a plugin. This function is only available from the Lua API. Plugins, next to their registered custom Lua functions, might need to perform operations on a regular basis and not when called from a threaded script (e.g. for synchronization purposes). They can do it when sim.handleModule is called (sim.openModule should however have been called previously). Refer to the messages relayed to plugins for more details. sim.handleModule can only be called from the main script and is not available in the C-API. Look at the default main script to get an idea about how to use sim.openModule, sim.handleModule and sim.closeModule.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	(1) number result=sim.handleModule(number sim.handle_all,boolean calledInSensingPart) (2) number result=sim.handleModule(string moduleName,boolean calledInSensingPart)
Lua parameters	**sim.handle_all**: indicates that all plugins should be handled (called) **moduleName**: the name of the plugin that should be handled (called) **calledInSensingPart**: set to false when called in the "actuation part". Set to true when called in the "sensing part"
Lua return values	**result**: -1 in case of an error, otherwise result is the number of plugins that executed the command.

simHandlePath (DEPRECATED)

Description	DEPRECATED. Instead of using this function, handle the path intrinsic position update inside of a child script.

simHandleProximitySensor / sim.handleProximitySensor

Description	Handles (performs sensing, etc. of) a registered proximity sensor object. See also sim.readProximitySensor, sim.checkProximitySensor, sim.checkProximitySensorEx and sim.resetProximitySensor.
C synopsis	simInt simHandleProximitySensor(simInt sensorHandle,simFloat* detectedPoint,simInt* detectedObjectHandle,simFloat* detectedSurfaceNormalVector)
C parameters	**sensorHandle**: handle of a proximity sensor object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will handle all proximity sensor objects, while sim_handle_all_except_explicit will only handle those that are not marked as "explicit handling") **detectedPoint**: coordinates of the closest detected point (x, y and z: detectedPoint[0]-detectedPoint[2]) relative to the sensor reference frame, and distance to the detected point (1 value: detectedPoint[3]). Can be NULL **detectedObjectHandle**: handle of the object that was detected. Can be NULL **detectedSurfaceNormalVector**: normal vector (normalized) of the detected surface. Relative to the sensor reference frame. Can be NULL

	When several proximity sensors are handled at the same time (e.g. with the sim_handle_all argument), then the output values are relative to the closest detection distance
C return value	0 if nothing was detected, -1 in case of an error. In a future release, a more detailed return value might be available
Lua synopsis	number result,number distance,table_3 detectedPoint,number detectedObjectHandle,table_3 detectedSurfaceNormalVector=sim.handleProximitySensor(number sensorHandle)
Lua parameters	**sensorHandle**: handle of a proximity sensor object or sim.handle_all or sim_handle_all_except_explicit. (sim.handle_all will handle all proximity sensor objects, while sim.handle_all_except_explicit will only handle those that are not marked as "explicit handling")
Lua return values	**result**: 0 if nothing was detected, -1 in case of an error. In a future release, a more detailed return value might be available **distance**: distance to the detected point if result is >0, nil otherwise **detectedPoint**: table of 3 numbers indicating the relative coordinates of the detected point if result is >0, nil otherwise **detectedObjectHandle**: handle of the object that was detected if result is >0, nil otherwise **detectedSurfaceNormalVector**: normal vector (normalized) of the detected surface. Relative to the sensor reference frame. Is nil if result is <1 When several proximity sensors are handled at the same time (e.g. with the sim.handle_all argument), then the return values are relative to the closest detection distance

simHandleSensingChildScripts (REPLACED)

Description	This function has been replaced with sim.handleChildScripts, and is not a valid API function anymore since V-REP 3.1.3

simHandleSensingStart / sim.handleSensingStart

Description	Handles various functionality (e.g. camera tracking during simulation, object velocity calculation, etc.). Should only be called from the main script, as the first instruction in the *sensing* section. See also sim.handleSimulationStart.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result=sim.handleSensingStart()
Lua parameters	none
Lua return values	-1 in case of an error.

simHandleSimulationStart / sim.handleSimulationStart

Description	Initializes various functionality (e.g. camera tracking during simulation, object velocity calculation, etc.). Should only be called from the main script, as the first instruction in the *initialization* section. See also sim.handleSensingStart.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result=sim.handleSimulationStart()
Lua parameters	none
Lua return values	-1 in case of an error.

simHandleVarious (DEPRECATED)

Description	DEPRECATED. See sim.handleSimulationStart and sim.handleSensingStart instead.

simHandleVisionSensor / sim.handleVisionSensor

Description	Handles (performs sensing, etc. of) a registered vision sensor object. See also sim.readVisionSensor, sim.checkVisionSensor, sim.checkVisionSensorEx and sim.resetVisionSensor.
C synopsis	simInt simHandleVisionSensor(simInt visionSensorHandle,simFloat** auxValues,simInt** auxValuesCount)

C parameters	**visionSensorHandle**: handle of a vision sensor object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will handle all vision sensor objects, while sim_handle_all_except_explicit will only handle those that are not marked as "explicit handling")
	auxValues: auxiliary values returned from the applied filters (refer to the documentation for details). By default V-REP returns one packet of 15 auxiliary values:the minimum of {intensity, red, green, blue, depth value}, the maximum of {intensity, red, green, blue, depth value}, and the average of {intensity, red, green, blue, depth value}. If additional filter components return values, then they will be appended as packets to the first packet. AuxValues can be NULL. The user is in charge of releasing the auxValues buffer with simReleaseBuffer(*auxValues). If visionSensorHandle is sim_handle_all or sim_handle_all_except_explicit, nothing is returned in auxValues.
	auxValuesCount: contains information about the number of auxiliary value packets and packet sizes returned in auxValues. The first value is the number of packets, the second is the size of packet1, the third is the size of packet2, etc. Can be NULL if auxValues is also NULL. The user is in charge of releasing the auxValuesCount buffer with simReleaseBuffer(*auxValuesCount).
	
	USAGE EXAMPLE:
	
	`float* auxValues=NULL;`
	`int* auxValuesCount=NULL;`
	`float averageColor[3]={0.0f,0.0f,0.0f};`
	`if (simHandleVisionSensor(visionSensorHandle,&auxValues,&auxValuesCount)>=0)`
	`{`
	`  if ((auxValuesCount[0]>0)||(auxValuesCount[1]>=15))`
	`  {`
	`  averageColor[0]=auxValues[11];`
	`  averageColor[1]=auxValues[12];`
	`  averageColor[2]=auxValues[13];`
	`  }`
	`  simReleaseBuffer((char*)auxValues);`
	`  simReleaseBuffer((char*)auxValuesCount);`
	`}`

C return value	number of detections (number of vision sensors that triggered a detection), -1 in case of an error
Lua synopsis	number detectionCount,table auxiliaryValuePacket1,table auxiliaryValuePacket2, etc.=sim.handleVisionSensor(number visionSensorHandle)
Lua parameters	Same as C-function
Lua return values	**detectionCount**: number of detections (number of vision sensors that triggered a detection), -1 in case of an error **auxiliaryValuePacket1**: default auxiliary value packet (same as for the C-function) **auxiliaryValuePacket2**: additional auxiliary value packet (e.g. from a filter component) **auxiliaryValuePacket3**: etc. (the function returns as many tables as there are auxiliary value packets)

simImportMesh / sim.importMesh

Description	Imports a mesh from a file. See also sim.exportMesh, sim.importShape and sim.createMeshShape
C synopsis	simInt simImportMesh(simInt fileformat,const simChar* pathAndFilename,simInt options,simFloat identicalVerticeTolerance,simFloat scalingFactor,simFloat*** vertices,simInt** verticesSizes,simInt*** indices,simInt** indicesSizes,simFloat*** reserved,simChar*** names)
C parameters	**fileformat**: the fileformat to import from. 0 for OBJ format, 1 for DXF format, 2 for 3DS format, 3 for ASCII STL format and 4 for BINARY STL format **pathAndFilename**: the location of the file to import. **options**: bit-coded: bit0 set (1): keep identical vertices, bit1 set (2): keep identical triangles, bit2 set (4): don't correct triangle windings **identicalVerticeTolerance**: the distance from which two distinct vertices will be merged. Bit0 of options should be cleared for this to have an effect **scalingFactor**: the scaling factor to apply to the imported vertices

vertices: an array to vertice arrays. The import operation may generate several meshes depending on the fileformat. The user is in charge of releasing the memory. See the example below
verticesSizes: an array indicating the individual vertice array sizes. The user is in charge of releasing the memory. See the example below
indices: an array to indice arrays. The import operation may generate several meshes depending on the fileformat. The user is in charge of releasing the memory. Can be NULL. See the example below
indicesSizes: an array indicating the individual indice array sizes. The user is in charge of releasing the memory. Can be NULL if indices is also NULL. See the example below
reserved: reserved for future extensions. Keep at NULL.
names: an array to mesh names extracted from the file. The import operation may generate several meshes depending on the fileformat. The user is in charge of releasing the memory. See the example below

USAGE EXAMPLE:

```
simFloat** vertices;
simInt* verticesSizes;
simInt** indices;
simInt* indicesSizes;
simChar** names;
simInt elementCount=simImportMesh(1,"d:\\example.dxf",0,0.0001f,1.0f,&vertices,
                            &verticesSizes,&indices,&indicesSizes,NULL,&names);
if (elementCount>0)
{
    const float grey[3]={0.5f,0.5f,0.5f};
    for (int i=0;i<elementCount;i++)
    {
        simInt shapeHandle=simCreateMeshShape(2,20.0f*3.1415f/180.0f,vertices[i],
                                verticesSizes[i],indices[i],indicesSizes[i],NULL);
        simSetObjectName(shapeHandle,names[i]);
        simSetShapeColor(shapeHandle,"",sim_colorcomponent_ambient,grey);
        simReleaseBuffer(names[i]);
        simReleaseBuffer((simChar*)indices[i]);
        simReleaseBuffer((simChar*)vertices[i]);
    }
    simReleaseBuffer((simChar*)names);
    simReleaseBuffer((simChar*)indicesSizes);
    simReleaseBuffer((simChar*)indices);
    simReleaseBuffer((simChar*)verticesSizes);
    simReleaseBuffer((simChar*)vertices);
}
```

C return value	Number of imported meshes, or 0 or -1 if the operation was not successful
Lua synopsis	table_of_table vertices,table_of_table indices,nil,table names=sim.importMesh(number fileformat,string pathAndFilename,number options,number identicalVerticeTolerance,number scalingFactor)
Lua parameters	Same as C-function

Lua return values	**vertices**: a table to vertice tables, or nil if operation was not successful. The import operation may generate several meshes depending on the fileformat. See the example below
	indices: a table to indice tables, or nil if operation was not successful. The import operation may generate several meshes depending on the fileformat. See the example below
	nil: return value is reserved for future extensions
	names: a table to mesh names extracted from the file, or nil if operation was not successful. The import operation may generate several meshes depending on the fileformat. See the example below
	
	USAGE EXAMPLE (e.g. in a customization script):
	
	`if (importButtonPressed) then`
	`  vertices,indices,reserved,names=sim.importMesh(1,"d:\\example.dxf",0,0.0001,1)`
	`  if (vertices) then`
	`  for i=1,#vertices,1 do`
	`  h=sim.createMeshShape(2,20*math.pi/180,vertices[i],indices[i])`
	`  sim.setShapeColor(h,"",sim.colorcomponent_ambient,{0.5,0.5,0.5})`
	`  sim.setObjectName(h,names[i])`
	`  end`
	`  end`
	`end`

simImportShape / sim.importShape

Description	Imports a shape from a file (first imports meshes, then groups/merges them into a shape). See also sim.importMesh.

C synopsis	simInt simImportShape(simInt fileformat,const simChar* pathAndFilename,simInt options,simFloat identicalVerticeTolerance,simFloat scalingFactor)
C parameters	**fileformat**: the fileformat to import from. 0 for OBJ format, 1 for DXF format, 2 for 3DS format, 3 for ASCII STL format, 4 for BINARY STL format and 5 for COLLADA format (in that case, make sure the collada plugin is available and correctly loaded). **pathAndFilename**: the location of the file to import. **options**: bit-coded: bit0 set (1): keep identical vertices bit1 set (2): keep identical triangles bit2 set (4): reserved. keep at 0. bit3 set (8): do not preserve colors (only for COLLADA format for now) bit4 set (16): tries to preserve textures (OBJ format only). When this bit is set and several shapes are imported, they will be grouped. If the bit is not set and several shapes are imported, they will be merged. **identicalVerticeTolerance**: the distance from which two distinct vertices will be merged. Bit0 of options should be cleared for this to have an effect **scalingFactor**: the scaling factor to apply to the imported vertices
C return value	The handle of the imported shape, or -1 if the operation was not successful
Lua synopsis	number shapeHandle=sim.importShape(number fileformat,string pathAndFilename,number options,number identicalVerticeTolerance,number scalingFactor)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simInitializePathSearch (DEPRECATED)

Description	DEPRECATED. See the OMPL library based path/motion planning functionality instead. Initializes a temporary path search object that can be used for stepped path planning calculations. Useful in conjunction with simPerformPathSearchStep when path planning calculations need to be performed in several steps so as to keep the simulator unblocked. When operating from a threaded child script, rather use the simSearchPath function. See also simGetPathPlanningHandle
C synopsis	simInt simInitializePathSearch(simInt pathPlanningObjectHandle,simFloat maximumSearchTime,simFloat searchTimeStep)
C parameters	**pathPlanningObjectHandle**: handle of the path planning object **maximumSearchTime**: maximum search time in seconds **searchTimeStep**: the duration of each search step (when calling simPerformPathSearchStep)
C return value	-1 if operation was not successful, otherwise a handle to a temporary path search object
Lua synopsis	number temporaryPathSearchObjectHandle=simInitializePathSearch(number pathPlanningObjectHandle,number maximumSearchTime,number searchTimeStep)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simInsertDataIntoStackTable

Description	Inserts data into a table on the stack. The function expects a *value* at the top of the stack, a *key* one position below, and a table below that. The value and its associated key will be inserted into the table and removed from the stack. If successive values are inserted with consecutive number keys starting at 1, then the table values can be accessed via number indices in a script, and the table can be seen as an array. Otherwise, the table can be seen as a map or associative array. See also the other stack functions.
C synopsis	simInt simInsertDataIntoStackTable(simInt stackHandle)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack.
C return value	-1 in case of an error.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simInsertObjectIntoOctree / sim.insertObjectIntoOctree

Description	Inserts an object into an octree, as voxels. Each voxel will store a color and a tag value. See also sim.subtractObjectFromOctree, sim.insertVoxelsIntoOctree and the other octree related functions.
C synopsis	simInt simInsertObjectIntoOctree(simInt octreeHandle,simInt objectHandle,simInt options,const simUChar* color,simUInt tag,simVoid* reserved)
C parameters	**octreeHandle**: the handle of the octree. See also simGetObjectHandle **objectHandle**: the handle of the object to insert. Only potentially collidable objects are supported **options**: reserved. Set to 0 **color**: a pointer to one RGB triple, specifying the red, green and blue color components (0-255). Can be NULL. **tag**: a uint32 value, which is user-defined. **reserved**: reserved for future extensions. Set to NULL
C return value	-1 if operation was not successful, otherwise the total number of voxels in the octree
Lua synopsis	number totalVoxelCnt=sim.insertObjectIntoOctree(number octreeHandle,number objectHandle,number options,table color=nil,number tag=0)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simInsertObjectIntoPointCloud / sim.insertObjectIntoPointCloud

Description	Inserts an object into a point cloud, as points. See also sim.insertPointsIntoPointCloud and the other point cloud related functions.
C synopsis	simInt simInsertObjectIntoPointCloud(simInt pointCloudHandle,simInt objectHandle,simInt options,simFloat gridSize,const simUChar* color,simVoid* reserved)
C parameters	**pointCloudHandle**: the handle of the point cloud. See also simGetObjectHandle **objectHandle**: the handle of the object to insert. Only potentially collidable objects are supported **options**: reserved. Set to 0 **gridSize**: when a shape is inserted, it will first be converted to an octree with a given grid or voxel size. **color**: a pointer to one RGB triple, specifying the red, green and blue color components (0-255). Can be NULL. **optionalValues**: can be used to specify additional parameters, or set to NULL for default parameter values: ((simInt*)optionalValues)[0]: an integer value that is bit coded. Each bit indicates which additional parameter will be taken into account: ((simFloat*)optionalValues)[1]: **duplicateTolerance**: a minimum distance tolerance value that is used to avoid duplicate points. To have this parameter taken into account, set bit0 to 1 in ((simInt*)optionalValues)[0]. Point insertion is slower when the duplicate tolerance is > then 0.0
C return value	-1 if operation was not successful, otherwise the total number of points in the point cloud
Lua synopsis	number totalPointCnt=sim.insertObjectIntoPointCloud(number pointCloudHandle,number objectHandle,number options,number gridSize,table color=nil,number duplicateTolerance=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simInsertPathCtrlPoints / sim.insertPathCtrlPoints

Description	Inserts one or several control points into a path object. See also sim.cutPathCtrlPoints and sim.createPath.
C synopsis	simInt simInsertPathCtrlPoints(simInt pathHandle,simInt options,simInt startIndex,simInt ptCnt,const simVoid* ptData)
C parameters	**pathHandle**: the handle of the path. Refer also to simGetObjectHandle. **options**: bit-coded: bit0: if set (1), then the path will be closed (given that there are enough control points in the path) bit1: if set (2), then the expected value count (valCnt) for each point will be 16 instead of 11 (see further down) **startIndex**: the zero-based index where the first new control point should be inserted. **ptCnt**: the number of control points to insert. **ptData (input)**: a buffer of ptCnt*valCnt values (float or int). ValCnt is 16 if bit1 of options is set,

	otherwise 11. Each new control point should have its properties described with following valCnt values: ptData[0]-ptData[2] (float values): the position of the control point (x,y,z), relative to the path object ptData[3]-ptData[5] (float values): the orientation of the control point in Euler angles (alpha,beta,gamma), relative to the path object ptData[6] (float value): the relative velocity at the control point ptData[7] (float value): the virtual distance at the control point ptData[8] (int value): the number of Bezier points at the control point ptData[9] (float value): the Bezier interpolation factor 1 at the control point ptData[10] (float value): the Bezier interpolation factor 2 at the control point ptData[11] (int value): the auxiliary flags at the control point ptData[12]-ptData[15] (float values): the 4 auxiliary values at the control point
C return value	-1 if operation was not successful.
Lua synopsis	number result=sim.insertPathCtrlPoints(number pathHandle,number options,number startIndex,number ptCnt,table ptData)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simInsertPointsIntoPointCloud / sim.insertPointsIntoPointCloud

Description	Inserts points into a point cloud. See also sim.removePointsFromPointCloud and the other point cloud related functions.
C synopsis	simInt simInsertPointsIntoPointCloud(simInt pointCloudHandle,simInt options,const simFloat* pts,simInt ptCnt,const simUChar* color,simVoid* reserved)
C parameters	**pointCloudHandle**: the handle of the point cloud. See also simGetObjectHandle **options**: bit-coded: bit0 set (1): specified points are relative to the point cloud reference frame, otherwise they are relative to the world reference frame bit1 set (2): the color array contains one RGB triple per point. Otherwise it contains a single RGB triple **pts**: a pointer to the point positions specified as X/Y/Z coordinates **ptCnt**: the number of point coordinates contained in pts **color**: a pointer to one or several RGB triples, specifying the red, green and blue color components (0-255). Can be NULL. **optionalValues**: can be used to specify additional parameters, or set to NULL for default parameter values: ((simInt*)optionalValues)[0]: an integer value that is bit coded. Each bit indicates which additional parameter will be taken into account: ((simFloat*)optionalValues)[1]: **duplicateTolerance**: a minimum distance tolerance value that is used to avoid duplicate points. To have this parameter taken into account, set bit0 to 1 in ((simInt*)optionalValues)[0]. Point insertion is slower when the duplicate tolerance is > then 0.0
C return value	-1 if operation was not successful, otherwise the total number of points in the point cloud
Lua synopsis	number totalPointCnt=sim.insertPointsIntoPointCloud(number pointCloudHandle,number options,table points,table color=nil,number duplicateTolerance=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simInsertVoxelsIntoOctree / sim.insertVoxelsIntoOctree

Description	Inserts voxels into an octree. Each voxel will store a color and a tag value. See also sim.removeVoxelsFromOctree and the other octree related functions.
C synopsis	simInt simInsertVoxelsIntoOctree(simInt octreeHandle,simInt options,const simFloat* pts,simInt ptCnt,const simUChar* color,const simUInt* tag,simVoid* reserved)
C parameters	**octreeHandle**: the handle of the octree. See also simGetObjectHandle **options**: bit-coded: bit0 set (1): specified points are relative to the octree reference frame, otherwise they are relative to the world reference frame bit1 set (2): the color array contains one RGB triple per point, and the tag array contains one value per point. Otherwise it the color array contains a single RGB triple, and the tag array contains a single value. **pts**: a pointer to the voxel positions specified as X/Y/Z coordinates

	ptCnt: the number of point coordinates contained in pts **color**: a pointer to one or several RGB triples, specifying the red, green and blue color components (0-255). Can be NULL. **tag**: a pointer to one or several uint32 values, which are user-defined values. Can be NULL, and should be NULL if *color* is NULL. **reserved**: reserved for future extensions. Set to NULL
C return value	-1 if operation was not successful, otherwise the total number of voxels in the octree
Lua synopsis	number totalVoxelCnt=sim.insertVoxelsIntoOctree(number octreeHandle,number options,table points,table color=nil,table tag=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simInterpolateMatrices / sim.interpolateMatrices

Description	Computes the interpolated transformation matrix between matrixIn1 and matrixIn2. Quaternions are used internally. See also the other matrix/transformation functions.
C synopsis	simInt simInterpolateMatrices(const simFloat* matrixIn1,const simFloat* matrixIn2,simFloat interpolFactor,simFloat* matrixOut)
C parameters	**matrixIn1**: the first input matrix **matrixIn2**: the second input matrix **interpolFactor**: the interpolation factor, a value between 0.0 and 1.0 (0.0--> matrixOut=matrixIn1, 1.0--> matrixOut=matrixIn2) **matrixOut**: the output matrix (the result of the interpolation). A transformation matrix contains 12 values (the last row (0,0,0,1) is omitted): The x-axis of the orientation component is (matrix[0],matrix[4],matrix[8]) The y-axis of the orientation component is (matrix[1],matrix[5],matrix[9]) The z-axis of the orientation component is (matrix[2],matrix[6],matrix[10]) The position component is (matrix[3],matrix[7],matrix[11])
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_12 resultMatrix=sim.interpolateMatrices(table_12 matrixIn1,table_12 matrixIn2,number interpolFactor)
Lua parameters	**matrixIn1**: the first input matrix (a table containing 12 values (the last row (0,0,0,1) is not required)) **matrixIn2**: the second input matrix (a table containing 12 values (the last row (0,0,0,1) is not required)) **interpolFactor**: the interpolation factor, a value between 0.0 and 1.0 (0.0--> resultMatrix=matrixIn1, 1.0--> resultMatrix=matrixIn2)
Lua return values	**resultMatrix**: the result matrix (a table containing 12 values (the last row (0,0,0,1) is omitted)). Table values in Lua are indexed from 1, not 0!

simIntersectPointsWithPointCloud / sim.intersectPointsWithPointCloud

Description	Removes points from a point cloud, that do not intersect with the provided points (i.e. the result in the point cloud will be the intersection between the two sets of points). When a point cloud doesn't use an octree calculation structure, then this operation cannot be performed. See also sim.insertPointsIntoPointCloud, sim.setPointCloudOptions and the other point cloud related functions.
C synopsis	simInt simIntersectPointsWithPointCloud(simInt pointCloudHandle,simInt options,const simFloat* pts,simInt ptCnt,simFloat tolerance,simVoid* reserved)
C parameters	**pointCloudHandle**: the handle of the point cloud. See also simGetObjectHandle **options**: bit-coded: bit0 set (1): specified points are relative to the point cloud reference frame, otherwise they are relative to the world reference frame **pts**: a pointer to the point positions specified as X/Y/Z coordinates. **ptCnt**: the number of point coordinates contained in pts **tolerance**: a distance used as a tolerance value **reserved**: reserved for future extensions. Set to NULL
C return value	-1 if operation was not successful, otherwise the total number of points in the point cloud
Lua synopsis	number totalPointCnt=sim.intersectPointsWithPointCloud(number pointCloudHandle,number options,table points,number tolerance)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simInvertMatrix / sim.invertMatrix

Description	Inverts a transformation matrix. See also the other matrix/transformation functions.
C synopsis	simInt simInvertMatrix(simFloat* matrix)
C parameters	**matrix**: pointer to 12 simFloat values representing the matrix that should be inverted (the last row of the 4x4 matrix (0,0,0,1) is not needed) The x-axis of the orientation component is (matrix[0],matrix[4],matrix[8]) The y-axis of the orientation component is (matrix[1],matrix[5],matrix[9]) The z-axis of the orientation component is (matrix[2],matrix[6],matrix[10]) The position component is (matrix[3],matrix[7],matrix[11])
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.invertMatrix(table_12 matrix)
Lua parameters	same as C-function
Lua return values	same as C-function

simIsHandleValid / sim.isHandleValid

Description	Checks whether a general object handle is still valid. When a general object is destroyed (e.g. programmatically or via the user interface), then its related handle is not valid anymore and will trigger an error when used. Use this function to avoid triggering an error. See also sim.getObjectHandle, sim.getCollectionHandle, sim.getCollisionHandle, sim.getDistanceHandle, sim.getMechanismHandle, sim.getIkGroupHandle, sim.getScriptHandle and sim.getObjectUniqueIdentifier.
C synopsis	simInt simIsHandleValid(simInt generalObjectHandle,simInt generalObjectType)
C parameters	**generalOjectHandle**: handle of a general-type object (e.g. scene object, collision object, distance object, etc.) **generalOjectType**: type of the general object. Refer to the general object types. Can be -1, in which case the specified handle is checked for validity in all types (handles of different types never overlap)
C return value	-1 if operation was not successful, 0 if the handle is not valid anymore, or 1 if the handle is still valid.
Lua synopsis	number result=sim.isHandleValid(number generalObjectHandle,number generalObjectType=-1)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simIsObjectInSelection / sim.isObjectInSelection

Description	Checks whether an object is selected. See also sim.getObjectSelection, sim.removeObjectFromSelection and sim.addObjectToSelection.
C synopsis	simInt simIsObjectInSelection(simInt objectHandle)
C parameters	**objectHandle**: handle of the object
C return value	3 if object is the last selection,1 if object is in selection, 0 if not, -1 if operation was not successful
Lua synopsis	number selectionState=sim.isObjectInSelection(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simIsRealTimeSimulationStepNeeded

Description	Indicates whether a call to simAdvanceSimulationByOneStep is needed (only useful during real-time simulations).
C synopsis	simInt simIsRealTimeSimulationStepNeeded()
C parameters	None
C return value	1 if call is needed, 0 if not needed, and -1 if the operation was not successful

Lua synopsis	-
Lua parameters	-
Lua return values	-

simIsScriptExecutionThreaded / sim.isScriptExecutionThreaded

Description	Checks whether the current script runs threaded
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result=sim.isScriptExecutionThreaded()
Lua parameters	None
Lua return values	**result**: -1 if operation was not successful, 0 if script execution is not threaded, >0 if script execution is threaded

simIsScriptRunningInThread / sim.isScriptRunningInThread (DEPRECATED)

Description	DEPRECATED. Refer to sim.isScriptExecutionThreaded instead.

simIsStackValueNull

Description	Tests whether the value at the top of the stack is Null. See also the other stack functions.
C synopsis	simInt simIsStackValueNull(simInt stackHandle)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack.
C return value	-1 in case of an error, 0 if the value is not null, 1 otherwise.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simJointGetForce (DEPRECATED)

Description	DEPRECATED. Refer to sim.getJointForce instead.

simLaunchExecutable / sim.launchExecutable

Description	Launches an executable. Similar to os.execute or io.popen, but is system independent.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result=sim.launchExecutable(string filename,string parameters='',number showStatus=1)
Lua parameters	**filename**: file name of the executable. If the filename starts with '@', then it will be considered as a system command, otherwise the current directory might be automatically prepended to the filename if it makes sense. **parameters**: optional input arguments **showStatus**: 0 to hide the application's window, 1 to show it. Works only with Windows OS.
Lua return values	**result**: -1 if operation was not successful. Under Windows OS, if the application could not be launched, return value is -1. Under Mac OS or Linux, return value might be different from -1 even if the application could not be launched.

simLaunchThreadedChildScripts / sim.launchThreadedChildScripts

Description	Starts or restarts (if appropriately flagged) all threaded child scripts. This command should only be called from the main script. Refer also to following related functions: sim.switchThread, sim.setThreadSwitchTiming, sim.setThreadAutomaticSwitch, sim.setThreadIsFree, sim.setThreadResumeLocation and sim.resumeThreads.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number launchCount=sim.launchThreadedChildScripts()
Lua parameters	-
Lua return values	launchCount: number of launched threaded child scripts

simLoadImage / sim.loadImage

Description	Loads an image from file or memory. See also sim.saveImage, sim.getScaledImage, sim.transformImage and sim.setVisionSensorCharImage.
C synopsis	simUChar* simLoadImage(simInt* resolution,simInt options,const simChar* filenameOrBuffer,simVoid* reserved)
C parameters	**resolution**: a pointer that will accept the image resolution. **options**: bit-coded. If bit0 is set (1), then the returned image is rgba, otherwise it is rgb. **filename**: the name of the file to read. The file extension indicates the format. If you wish to load an image from a memory buffer, then have filename point to that memory buffer. **reserved**: Set to NULL if you load an image from file. If you load an image from a memory buffer, this should be to an integer pointer where the first pointed integer is the size of the memory buffer.
C return value	NULL if operation was not successful, otherwise a buffer containing the image data. The user is in charge of releasing the buffer with simReleaseBuffer.
Lua synopsis	string image,table_2 resolution=sim.loadImage(number options,string filenameOrBuffer)
Lua parameters	Similar as C-function. To load an image from a string buffer, simply prefix the buffer with "@mem".
Lua return values	Similar as C-function

simLoadModel / sim.loadModel (remote API equivalent: simxLoadModel)

Description	Loads a previously saved model, and selects it. See also sim.saveModel, sim.loadScene, and sim.setBoolParameter with sim.boolparam_scene_and_model_load_messages.
C synopsis	simInt simLoadModel(const simChar* filename)
C parameters	**filename**: model filename. The filename extension is required ("ttm")
C return value	-1 if operation was not successful. Otherwise the handle of the model base object.
Lua synopsis	a) number objectHandle=sim.loadModel(string filename) b) number objectHandle=sim.loadModel(string buffer) c) string rgbaImage=sim.loadModel(string filename,bool onlyThumbnail) d) string rgbaImage=sim.loadModel(string buffer,bool onlyThumbnail)
Lua parameters	a) **filename**: model filename. The filename extension is required ("ttm") b) **buffer**: a buffer containing the model c)&d) **onlyThumbnail**: when true, then only the thumbnail image of the model will be loaded and returned
Lua return values	a)&b) -1 if operation was not successful. Otherwise the handle of the model base object. c)&d) nil if operation was not successful. Otherwise the model thumbnail image (128x128x4, rgba).

simLoadModule / sim.loadModule

Description	Loads a V-REP plugin. This should usually be done in the main client application, just after simRunSimulator was called. Alternatively, you can also dynamically load/unload a plugin, but

	depending on the plugin function, this might not work/lead to a crash. In case the dynamically loaded plugin registers custom Lua functions, those functions cannot be used in scripts that were already initialized (except for the script that called *sim.loadModule*). Normally, all plugins of type *v_repExtXXX.dll* (or *libv_repExtXXX.so* or *libv_repExtXXX.dylib*) in the V-REP directory are loaded at application start. Plugins that are meant to be dynamically loaded should use a different name, or a different directory. See also simSendModuleMessage and sim.unloadModule.
C synopsis	simInt simLoadModule(const simChar* filenameAndPath,const simChar* pluginName)
C parameters	**filenameAndPath**: file name and path of the plugin **pluginName**: name of the plugin. If the file name is *v_repExtXXX.dll*, then the name should be *XXX*
C return value	handle of the plugin if value is 0 or positive. otherwise: -3: plugin could not be loaded -2: plugin is missing entry points -1: plugin could not initialize
Lua synopsis	number pluginHandle=sim.loadModule(string filenameAndPath,string pluginName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simLoadScene / sim.loadScene (remote API equivalent: simxLoadScene)

Description	Loads a previously saved scene. If current scene is not empty, a new scene will be created before switching to it and loading the scene. See also sim.saveScene, sim.loadModel, simCloseScene and sim.setBoolParameter with sim.boolparam_scene_and_model_load_messages.
C synopsis	simInt simLoadScene(const simChar* filename)
C parameters	**filename**: scene filename. The filename extension is required ("ttt")
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.loadScene(string filename)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simLoadUI (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simLockInterface (DEPRECATED)

Description	DEPRECATED. Has no effect anymore.

simLockResources

Description	Prepares V-REP for access to resources. The V-REP API functions normally automatically protect resources appropriately: when the non-GUI thread reads from or writes to resources, they will be protected (the lock always succeeds). When the GUI thread reads from or writes to resources, the protection might fail (i.e the lock will fail if the non-GUI thread has already locked resources for a longer time), in which case the API function will return with a fail error code. This can be troublesome in case a long succession of API calls is planned. In that case, you should additionally protect the API function calls with one initial call to simLockResources. See also simUnlockResources and sim.getThreadId.
C synopsis	simInt simLockResources(simInt lockType,simInt reserved)
C parameters	**lockType**: the lock type. **reserved**: reserved. Set to -1.
C return value	-1 if the lock has failed. Otherwise a lock handle. A locked should always eventually be released, otherwise you will experience a dead-lock or crash.

Lua synopsis	-
Lua parameters	-
Lua return values	-

simModifyGhost / sim.modifyGhost

Description	Modifies or removes a ghost object previously added with sim.addGhost.
C synopsis	simInt simModifyGhost(simInt ghostGroup,simInt ghostId,simInt operation,simFloat floatValue,simInt options,simInt optionsMask,const simFloat* colorOrTransformation)
C parameters	**ghostGroup**: an identifier that allows, together with **ghostId**, identifying which ghosts to modify. **ghostId**: an identifier that allows, together with **ghostGroup**, identifying which ghosts to modify. If -1, then all ghosts that match **ghostGroup** will be modified. **operation**: a value indicating the operation to perform: **0**: no operation is performed, and the return value indicates the number of ghosts that match the identifiers. **1**: removes the specified ghosts. The return value indicates the number of removed ghosts. **2**: sets the start time (via **floatValue**) of the specified ghosts. The return value indicates the number of modified ghosts. **3**: sets the end time (via **floatValue**) of the specified ghosts. The return value indicates the number of modified ghosts. **4**: shifts the start time (via **floatValue**) of the specified ghosts. The return value indicates the number of modified ghosts. **5**: shifts the end time (via **floatValue**) of the specified ghosts. The return value indicates the number of modified ghosts. **6**: shifts the start and end times (via **floatValue**) of the specified ghosts. The return value indicates the number of modified ghosts. **7**: scales the start time (via **floatValue**) of the specified ghosts. The return value indicates the number of modified ghosts. **8**: scales the end time (via **floatValue**) of the specified ghosts. The return value indicates the number of modified ghosts. **9**: scales the start and end times (via **floatValue**) of the specified ghosts. The return value indicates the number of modified ghosts. **10**: modifies the attributes (via **options** and **optionsMask**) of the specified ghosts. The return value indicates the number of modified ghosts. **11**: pre-multiplies (via **colorOrTransformation**) the transformations of the specified ghosts. The return value indicates the number of modified ghosts. **12**: post-multiplies (via **colorOrTransformation**) the transformations of the specified ghosts. The return value indicates the number of modified ghosts. **13**: modifies (via **colorOrTransformation**) the color of the specified ghosts. The return value indicates the number of modified ghosts. **14**: sets the transparency factor (via **floatValue**) of the specified ghosts. Only ghosts with custom colors will be influenced. The return value indicates the number of modified ghosts. **floatValue**: a floating point value that is used to modify the specified ghosts. See **operation** here above. **options**: the attributes used to modify the specified ghosts (see **operation** here above). Attributes are bit-coded: **bit0** reserved **bit1** set (2)=the provided start- and end-times will be played-back in real-time **bit2** set (4)=preserve the original colors **bit3** reserved **bit4** set (16)=create an invisible ghost **bit5** set (32)=backface culling for the ghost (only when using custom colors) **optionsMask**: a mask allowing to select options to set. **colorOrTransformation**: a pointer to 7 or 12 float values: when **operation** is 11 or 12, 7 values indicating a transformation should be provided (x,y,z,qx,qy,qz,qw). when **operation** is 13, 12 values indicating a color should be provided (ambient_diffuse RGB, 3 reserved values (set to zero), specular RGB and emissive RGB). Can be NULL when operation is different from 11, 12 or 13.
C return value	-1 if operation was not successful, otherwise a value depending on the **operation** performed.
Lua synopsis	number result=sim.modifyGhost(number ghostGroup,number ghostId,number operation,number floatValue,number options=nil,number optionsMask=nil,table colorOrTransformation=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simModifyPointCloud / sim.modifyPointCloud (DEPRECATED)

Description	DEPRECATED. See point cloud object and point cloud related functions instead.

simMoveStackItemToTop

Description	Removes the top item in the stack, effectively reducing the stack size by one. See also simPopStackItem and the other stack functions.
C synopsis	simInt simMoveStackItemToTop(simInt stackHandle,simInt cIndex)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **cIndex**: the zero-based index of the item to move.
C return value	-1 in case of an error.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simMoveToJointPositions / sim.moveToJointPositions (DEPRECATED)

Description	DEPRECATED. See sim.rmlMoveToJointPositions instead.

simMoveToObject / sim.moveToObject

Description	Moves an object to the position/orientation of another moving object (target object) by performing interpolations (i.e. the object will effectiviely follow the target object). If the target object is a path, a position on the path can be specified. If the target object is not moving, use rather sim.getObjectPosition, sim.getObjectOrientation or sim.getPositionOnPath, sim.getOrientationOnPath in conjunction with sim.rmlMoveToPosition. This function can only be called from child scripts running in a thread (since this is a blocking operation) and is not available from the C-API. See also sim.followPath.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number deltaTimeLeft=sim.moveToObject(number objectHandle,number targetObjectHandle,number positionAndOrOrientation,number relativeDistanceOnPath,number velocity,number acceleration)
Lua parameters	**objectHandle**: handle of the object to be moved **targetObjectHandle**: handle of the object to be followed **positionAndOrOrientation**: a value between 1 and 3 (1: only position is modified, 2: only orientation is modified, 3: position and orientation is modified). Can be nil in which case 3 is applied. **relativeDistanceOnPath**: a value between 0 and 1, where 0 is the beginning of the path, and 1 the end of the path. Make sure you selected the appropriate path length calculation method. Can be nil in which case the path object is treated as a regular object. **velocity**: movement velocity expressed in 1/s (i.e. the inverse of the velocity indicates the time that will be taken to reach the target object) **acceleration**: the acceleration/deceleration expressed in 1/s^2. Can be nil in which case an infinite acceleration is applied.
Lua return values	**deltaTimeLeft**: if the time needed to reach the target object is not a multiple of the simulation time step, then deltatimeLeft is the execution time left at current simulation time. deltaTimeLeft is also memorized internally on a thread-basis and used as compensation or correction factor in subsequent blocking commands. deltaTimeLeft is nil in case of an error.

simMoveToPosition / sim.moveToPosition (DEPRECATED)

Description	DEPRECATED. Use sim.rmlMoveToPosition instead.

simMsgBox / sim.msgBox

Description	Opens a modal message box for interaction with the user. See also sim.fileDialog and sim.displayDialog
C synopsis	simInt simMsgBox(simInt dlgType,simInt buttons,const simChar* title,const simChar* message)
C parameters	**dlgType**: the message box type. **buttons**: the buttons to display. **title**: title of the dialog **message**: the message to display
C return value	a message box return value
Lua synopsis	number returnValue=sim.msgBox(number dlgType,number buttons,string title,string message)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simMultiplyMatrices / sim.multiplyMatrices

Description	Multiplies two transformation matrices. See also the other matrix/transformation functions.
C synopsis	simInt simMultiplyMatrices(const simFloat* matrixIn1,const simFloat* matrixIn2,simFloat* matrixOut)
C parameters	**matrixIn1**: the first input matrix **matrixIn2**: the second input matrix **matrixOut**: the output matrix (the result of the multiplication: matrixIn1*matrixIn2). A transformation matrix contains 12 values (the last row (0,0,0,1) is omitted): The x-axis of the orientation component is (matrix[0],matrix[4],matrix[8]) The y-axis of the orientation component is (matrix[1],matrix[5],matrix[9]) The z-axis of the orientation component is (matrix[2],matrix[6],matrix[10]) The position component is (matrix[3],matrix[7],matrix[11])
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_12 resultMatrix=sim.multiplyMatrices(table_12 matrixIn1,table_12 matrixIn2)
Lua parameters	Same as C-function
Lua return values	**resultMatrix**: the result matrix (a table containing 12 values (the last row (0,0,0,1) is omitted)). Table values in Lua are indexed from 1, not 0!

simMultiplyVector / sim.multiplyVector

Description	Multiplies a vector with a transformation matrix (v=m*v). See also the other matrix/transformation functions.
C synopsis	See simTransformVector
C parameters	-
C return value	-
Lua synopsis	table_3 resultVector=sim.multiplyVector(table_12 matrix,table_3 vector)
Lua parameters	**matrix**: the transformation matrix (a table containing 12 values (the last row (0,0,0,1) is not required)) **vector**: the original vector (a table containing 3 values (the last element (1) of the homogeneous coordinates is not required)
Lua return values	**resultVector**: the result vector (a table containing 3 values (the last element (1) of the homogeneous coordinates is omitted))

simOpenModule / sim.openModule

Description	"Opens" a plugin (allowing it to reserve resources at the start of a simulation). This command can only be called from the main script. Call it from the main script in the first simulation pass (usually with sim.handle_all argument). sim.openModule is not available in the C-API. Look at the default main script to get an idea about how to use sim.openModule, sim.handleModule and sim.closeModule.
C synopsis	-

C parameters	-
C return value	-
Lua synopsis	(1) number result=sim.openModule(number sim.handle_all) (2) number result=sim.openModule(string moduleName)
Lua parameters	sim.handle_all: indicates that all plugins should be opened moduleName: the name of the plugin that should be opened
Lua return values	**result**: -1 in case of an error, otherwise result is the number of plugins that executed the command.

simOpenTextEditor / sim.openTextEditor

Description	Opens a modal text edition window.
C synopsis	simChar* simOpenTextEditor(const simChar* initText,const simChar* xml,simVoid* reserved)

C parameters	**initText**: a pointer to the initial text to be displayed.
	xml: a pointer to an XML description of the text editor's properties. Can be NULL for default properties. Following is a valid content:
	
	`<editor title="Window title" editable="true" searchable="true"`
	`  tabWidth="4" textColor="50 50 50" backgroundColor="190 190 190"`
	`  selectionColor="128 128 255" size="800 600" position="100 100" >`
	
	`  <keywords1 color="152 0 0" >`
	`  <item word="simGetObjectHandle" autocomplete="true"`
	`  calltip="number handle=simGetObjectHandle(number objectName)" />`
	`  <item word="simGetObjectPosition" autocomplete="true"`
	`  calltip="table_3 pos=simGetObjectPosition(number handle,`
	`  number relHandle)" />`
	`  </keywords1>`
	`  <keywords2 color="220 80 20" >`
	`  <item word="simGetObjectOrientation" autocomplete="true"`
	`  calltip="table_3 euler=simGetObjectOrientation(number handle,`
	`  number relHandle)" />`
	`  </keywords2>`
	`</editor>`
	
	Other *editor* attributes with their default values are:
	
	`isLua="false"`
	`useVrepKeywords="false"`
	`commentColor="0 140 0"`
	`numberColor="220 0 220"`
	`stringColor="255 255 0"`
	`characterColor="255 255 0"`
	`operatorColor="0 0 0"`
	`preprocessorColor="0 128 128"`
	`identifierColor="64 64 64"`
	`wordColor="0 0 255"`
	`word4Color="152 64 0"`
	
	reserved: reserved for future extension. Set to NULL.

C return value	NULL in case of an error. Otherwise a pointer to the text. The user is in charge of releasing the returned buffer with simReleaseBuffer.
Lua synopsis	string text=sim.openTextEditor(string initText,string xml=nil)
Lua parameters	Similar to C-function
Lua return values	Similar to C-function

simPackBytes / sim.packUInt8Table (DEPRECATED)

Description	DEPRECATED. Refer to sim.packUInt8Table instead.

simPackDoubleTable / sim.packDoubleTable

Description	Packs a table of double floating-point numbers into a string. See also sim.unpackDoubleTable and the other packing/unpacking functions.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	string data=sim.packDoubleTable(table doubleNumbers,number startDoubleIndex=0,number doubleCount=0)
Lua parameters	**doubleNumbers**: a table containing double floating-point numbers. Non-numbers will be packed as zero values **startDoubleIndex**: the zero-based index from which on data should be packed. Can be omitted in which case 0 is used **doubleCount**: the amount of doubles that should be packed. Can be omitted in which case 0 is used (which indicates that the maximum available doubles should be packed from the indicated startDoubleIndex)
Lua return values	**data**: a string (values between 0 and 255) that contains packed double floating-point numbers from the table if operation was successful, nil otherwise

simPackDoubles / sim.packDoubleTable (DEPRECATED)

Description	DEPRECATED. Refer to sim.packDoubleTable instead.

simPackFloatTable / sim.packFloatTable

Description	Packs a table of floating-point numbers into a string. See also sim.unpackFloatTable and the other packing/unpacking functions.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	string data=sim.packFloatTable(table floatingNumbers,number startFloatIndex=0,number floatCount=0)
Lua parameters	**floatingNumbers**: a table containing floating-point numbers. Non-numbers will be packed as zero values **startFloatIndex**: the zero-based index from which on data should be packed. Can be omitted in which case 0 is used **floatCount**: the amount of floats that should be packed. Can be omitted in which case 0 is used (which indicates that the maximum available floats should be packed from the indicated startFloatIndex)
Lua return values	**data**: a string (values between 0 and 255) that contains packed floating-point numbers from the table if operation was successful, nil otherwise

simPackFloats / sim.packFloatTable (DEPRECATED)

Description	DEPRECATED. Refer to sim.packFloatTable instead.

simPackInt32Table / sim.packInt32Table

Description	Packs a table of int32 numbers into a string. See also sim.unpackInt32Table and the other packing/unpacking functions.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	string data=sim.packInt32Table(table int32Numbers,number startInt32Index=0,number int32Count=0)
Lua parameters	**int32Numbers**: a table containing int32 numbers. Non-numbers will be packed as zero values

	startInt32Index: the zero-based index from which on data should be packed. Can be omitted in which case 0 is used **int32Count**: the amount of int32s that should be packed. Can be omitted in which case 0 is used (which indicates that the maximum available int32s should be packed from the indicated startInt32Index)
Lua return values	**data**: a string (values between 0 and 255) that contains packed int32 numbers from the table if operation was successful, nil otherwise

simPackInts / sim.packInt32Table (DEPRECATED)

Description	DEPRECATED. Refer to sim.packInt32Table instead.

simPackTable / sim.packTable

Description	Packs a table into a buffer. The table may contain other nested tables, nil, boolean, number or string values. All other types (e.g. functions) will be considered as nil values. See also sim.unpackTable, the other stack functions and the other packing/unpacking functions.
C synopsis	simChar* simPackTable(simInt stackHandle,simInt* bufferSize)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. There must be a table located at the top of the stack. **bufferSize**: the size of the returned buffer.
C return value	NULL in case of an error, otherwise a data buffer. The user is in charge of releasing the returned buffer with simReleaseBuffer.
Lua synopsis	string buffer=sim.packTable(table aTable)
Lua parameters	**aTable**: a script table.
Lua return values	**buffer**: a string buffer.

simPackUInt16Table / sim.packUInt16Table

Description	Packs a table of uint16 numbers into a string. See also sim.unpackUInt16Table and the other packing/unpacking functions.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	string data=sim.packUInt16Table(table uint16Numbers,number startUint16Index=0,number uint16Count=0)
Lua parameters	**uint16Numbers**: a table containing uint16 numbers. Invalid uint16 numbers will be packed in an undefined manner. **startUint16Index**: the zero-based index from which on data should be packed. Can be omitted in which case 0 is used **uint16Count**: the amount of uint16s that should be packed. Can be omitted in which case 0 is used (which indicates that the maximum available uint16s should be packed from the indicated startUint16Index)
Lua return values	**data**: a string (values between 0 and 255) that contains packed uint16 numbers from the table if operation was successful, nil otherwise

simPackUInt32Table / sim.packUInt32Table

Description	Packs a table of uint32 numbers into a string. See also sim.unpackUInt32Table and the other packing/unpacking functions.
C synopsis	-
C parameters	-
C return value	-

Lua synopsis	string data=sim.packUInt32Table(table uint32Numbers,number startUint32Index=0,number uint32Count=0)
Lua parameters	**uint32Numbers**: a table containing uint32 numbers. Non-numbers will be packed as zero values **startUint32Index**: the zero-based index from which on data should be packed. Can be omitted in which case 0 is used **uint32Count**: the amount of uint32s that should be packed. Can be omitted in which case 0 is used (which indicates that the maximum available uint32s should be packed from the indicated startUint32Index)
Lua return values	**data**: a string (values between 0 and 255) that contains packed uint32 numbers from the table if operation was successful, nil otherwise

simPackUInt8Table / sim.packUInt8Table

Description	Packs a table of uint8 numbers into a string. See also sim.unpackUInt8Table and the other packing/unpacking functions.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	string data=sim.packUInt8Table(table uint8Numbers,number startUint8Index=0,number uint8count=0)
Lua parameters	**uint8Numbers**: a table containing uint8 numbers. Invalid byte number will be packed in an undefined manner. **startUint8Index**: the zero-based index from which on data should be packed. Can be omitted in which case 0 is used **uint8count**: the amount of uint8s that should be packed. Can be omitted in which case 0 is used (which indicates that the maximum available uint8s should be packed from the indicated startUint8Index)
Lua return values	**data**: a string (values between 0 and 255) that contains the uint8 numbers from the table if operation was successful, nil otherwise

simPackUInts / sim.packUInt32Table (DEPRECATED)

Description	DEPRECATED. Refer to sim.packUInt32Table instead.

simPackWords / sim.packUInt16Table (DEPRECATED)

Description	DEPRECATED. Refer to sim.packUInt16Table instead.

simPauseSimulation / sim.pauseSimulation (remote API equivalent: simxPauseSimulation)

Description	Requests a pause of a simulation. See also sim.startSimulation, sim.stopSimulation and sim.getSimulationState. See also the simulation state diagram.
C synopsis	simInt simPauseSimulation()
C parameters	None
C return value	-1 in case of an error, 0 if the operation could not be performed. >0 in case of success.
Lua synopsis	number result=sim.pauseSimulation()
Lua parameters	Same as C-function
Lua return values	Same as C-function

simPerformPathSearchStep (DEPRECATED)

Description	DEPRECATED. See the OMPL library based path/motion planning functionality instead. Performs a path search in several sub-steps. simInitializePathSearch has to be called previously. Call this function in a loop until the return value is different from -2. When operating from a threaded child script, rather use the simSearchPath function
C synopsis	simInt simPerformPathSearchStep(simInt temporaryPathSearchObject,simBool abortSearch)
C parameters	**temporaryPathSearchObject**: handle of a temporary path search object (returned by simInitializePathSearch) **abortSearch**: 0 to continue the search, any other value to abort
C return value	-2 if nothing was found but search can continue (by calling this function again), -1 in case of an error, 0 if no path was found, 1 if a partial path was found, and 2 if a full path was found. Unless the return value is -2, the temporary path search object will be destroyed
Lua synopsis	number result=simPerformPathSearchStep(number temporaryPathSearchObjectHandle,boolean abortSearch)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simPersistentDataRead / sim.persistentDataRead

Description	Reads a block of persistent data. See also sim.persistentDataWrite, sim.getStringSignal, sim.getIntegerSignal, sim.getFloatSignal and sim.readCustomDataBlock.
C synopsis	simChar* simPersistentDataRead(const simChar* dataName,simInt* dataLength)
C parameters	**dataName**: name of the data block **dataLength**: the size of the returned data block, since it may contain any data (also embedded zeros).
C return value	NULL if operation was not successful or data block does not exist, otherwise the data block (which may contain any value, including embedded zeros). In that case the returned buffer should be released with simReleaseBuffer
Lua synopsis	string dataValue=sim.persistentDataRead(string dataName)
Lua parameters	**dataName**: name of the data block
Lua return values	**dataValue**: value of the data block, or nil if operation was not successful or data block does not exist. The returned data block may contain any value, including embedded zeros.

simPersistentDataWrite / sim.persistentDataWrite

Description	Writes a persistent data block. Persistent data, valid across all opened simulator scenes, remains until the simulator ends, or until it is cleared by writing an empty data block. If the options flag is set appropriately, then persistent data can also be stored on file, and be automatically reloaded next time V-REP starts. See also sim.persistentDataRead, sim.setStringSignal, sim.setIntegerSignal, sim.setFloatSignal and sim.writeCustomDataBlock.
C synopsis	simInt simPersistentDataWrite(const simChar* dataName,const simChar* dataValue,simInt dataLength,simInt options)
C parameters	**dataName**: name of the data block **dataValue**: content of the data block (which may contain any value, including embedded zeros). If dataValue is an empty string, then the data block is cleared (if present). **dataLength**: the size of the data block. **options**: bit-coded. If bit 0 is set (1), then the data is also stored on file ("system/persistentData.dat"), and automatically reloaded next time V-REP start.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.persistentDataWrite(string dataName,string dataValue,number options=0)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simPopStackItem

Description	Removes the top item in the stack, effectively reducing the stack size by one. See also simMoveStackItemToTop and the other stack functions.

C synopsis	simInt simPopStackItem(simInt stackHandle,simInt count)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **count**: the number of items to pop, or 0 to clear the stack.
C return value	-1 in case of an error, otherwise the new size of the stack.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simPushBoolOntoStack

Description	Pushes a Boolean value onto the stack. The value will then be located at the top of the stack. See also the other stack functions.
C synopsis	simInt simPushBoolOntoStack(simInt stackHandle,simBool value)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **value**: the value to push.
C return value	-1 in case of an error.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simPushDoubleOntoStack

Description	Pushes a double precision value (i.e. a Lua number) onto the stack. The value will then be located at the top of the stack. See also the other stack functions.
C synopsis	simInt simPushDoubleOntoStack(simInt stackHandle,simDouble value)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **value**: the value to push.
C return value	-1 in case of an error.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simPushDoubleTableOntoStack

Description	Pushes a double-precision array onto the stack, as a table filled with Lua numbers. The table will then be located at the top of the stack. See also the other stack functions.
C synopsis	simInt simPushDoubleTableOntoStack(simInt stackHandle,const simDouble* values,simInt valueCnt)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **values**: a double-precision array. **valueCnt**: the size of the array.
C return value	-1 in case of an error.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simPushFloatOntoStack

Description	Pushes a float value (i.e. a Lua number) onto the stack. The value will then be located at the top of the stack. See also the other stack functions.
C synopsis	simInt simPushFloatOntoStack(simInt stackHandle,simFloat value)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **value**: the value to push.

C return value	-1 in case of an error.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simPushFloatTableOntoStack

Description	Pushes a float array onto the stack, as a table filled with Lua numbers. The table will then be located at the top of the stack. See also the other stack functions.
C synopsis	simInt simPushFloatTableOntoStack(simInt stackHandle,const simFloat* values,simInt valueCnt)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **values**: a float array. **valueCnt**: the size of the array.
C return value	-1 in case of an error.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simPushInt32OntoStack

Description	Pushes an int32 value (i.e. a Lua number) onto the stack. The value will then be located at the top of the stack. See also the other stack functions.
C synopsis	simInt simPushInt32OntoStack(simInt stackHandle,simInt value)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **value**: the value to push.
C return value	-1 in case of an error.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simPushInt32TableOntoStack

Description	Pushes an int32 array onto the stack, as a table filled with Lua numbers. The table will then be located at the top of the stack. See also the other stack functions.
C synopsis	simInt simPushInt32TableOntoStack(simInt stackHandle,const simInt* values,simInt valueCnt)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **values**: an int32 array. **valueCnt**: the size of the array.
C return value	-1 in case of an error.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simPushNullOntoStack

Description	Pushes the value Null (or nil) onto the stack. The value will then be located at the top of the stack. See also the other stack functions.
C synopsis	simInt simPushNullOntoStack(simInt stackHandle)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack.
C return value	-1 in case of an error.
Lua synopsis	-
Lua parameters	-

Lua return values	-

simPushStringOntoStack

Description	Pushes a string onto the stack. The string may contain any values, including embedded zeros. The string will then be located at the top of the stack. See also the other stack functions.
C synopsis	simInt simPushStringOntoStack(simInt stackHandle,const simChar* value,simInt stringSize)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **value**: the string. **stringSize**: the length of the string. If you specify 0, the string is a text string and its length will be automatically determined.
C return value	-1 in case of an error.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simPushTableOntoStack

Description	Pushes an empty table onto the stack. The table will then be located at the top of the stack. See also the other stack functions.
C synopsis	simInt simPushTableOntoStack(simInt stackHandle)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack.
C return value	-1 in case of an error.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simPushUInt8TableOntoStack

Description	Pushes a uint8 array onto the stack, as a table filled with Lua numbers. The table will then be located at the top of the stack. See also the other stack functions.
C synopsis	simInt simPushUInt8TableOntoStack(simInt stackHandle,const simUChar* values,simInt valueCnt)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. **values**: a uint8 array. **valueCnt**: the size of the array.
C return value	-1 in case of an error.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simQuitSimulator / sim.quitSimulator

Description	Triggers a quit signal that will eventually quits the application. See also simRunSimulator.
C synopsis	simVoid simQuitSimulator(simBool doNotDisplayMessages)
C parameters	**doNotDisplayMessages**: when true, will force quit.
C return value	-
Lua synopsis	sim.quitSimulator(boolean doNotDisplayMessages)
Lua parameters	Same as C-function
Lua return values	-

simRMLMoveToJointPositions / sim.rmlMoveToJointPositions

Description	Moves (actuates) several joints at the same time using the Reflexxes Motion Library type II or IV. This function can only be called from child scripts running in a thread (since this is a blocking operation) and is not available from the C-API. See also sim.rmlMoveToPosition, and sim.rmlPos.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result,table newPos,table newVel,table newAccel,number timeLeft=sim.rmlMoveToJointPositions(table jointHandles,number flags,table currentVel,table currentAccel,table maxVel,table maxAccel,table maxJerk,table targetPos,table targetVel,table direction)
Lua parameters	**jointHandles**: handles of the joints to actuate **flags**: RML flags. -1 for default flags. **currentVel**: the current velocity of the joints. Can be nil in which case a velocity vector of 0 is used. **currentAccel**: the current acceleration of the joints. Can be nil in which case an acceleration vector of 0 is used. **maxVel**: the maximum allowed velocity of the joints **maxAccel**: the maximum allowed acceleration of the joints **maxJerk**: the maximum allowed jerk of the joints. With the RML type II plugin, the max. jerk will however always be infinite. **targetPos**: the desired target positions of the joints **targetVel**: the desired velocity of the joints at the target. Can be nil in which case a velocity vector of 0 is used. **direction**: the desired rotation direction for cyclic revolute joints: 0 for the shortest distance, -x for a movement towards negative values, +x for a movement towards positive values (n=(x-1) represents the number of additional turns). Can be nil or omitted, in which case a value of 0 is used for all joints.
Lua return values	**result**: 1 if the function call was successful **newPos**: the new positions of the joints **newVel**: the new velocities of the joints **newAccel**: the new accelerations of the joints **timeLeft**: the time left for additional calculations in current simulation time step

simRMLMoveToPosition / sim.rmlMoveToPosition

Description	Moves an object to a given position and/or orientation using the Reflexxes Motion Library type II or IV. This function can only be called from child scripts running in a thread (since this is a blocking operation) and is not available from the C-API. See also sim.rmlMoveToJointPositions, sim.rmlPos, sim.moveToObject and sim.followPath.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result,table_3 newPos,table_4 newQuaternion,table_4 newVel,table_4 newAccel,number timeLeft=sim.rmlMoveToPosition(number objectHandle,number relativeToObjectHandle,number flags,table_4 currentVel,table_4 currentAccel,table_4 maxVel,table_4 maxAccel,table_4 maxJerk,table_3 targetPosition,table_4 targetQuaternion,table_4 targetVel)
Lua parameters	**objectHandle**: handle of the object to be moved **relativeToObjectHandle**: indicates relative to which reference frame the movement data is specified. Specify -1 for a movement relative to the absolute reference frame, sim.handle_parent for a movement relative to the object's parent frame, or an object handle relative to whose reference frame the movement should be performed. **flags**: RML flags. -1 for default flags. **currentVel**: the current velocity of the object (velX, velY, velZ, velAngle). Can be nil in which case a velocity vector of 0 is used. **currentAccel**: the current acceleration of the object (accelX, accelY, accelZ, accelAngle). Can be nil in which case an acceleration vector of 0 is used. **maxVel**: the maximum allowed velocity of the object (maxVelX, maxVelY, maxVelZ, maxVelAngle) **maxAccel**: the maximum allowed acceleration of the object (maxAccelX, maxAccelY, maxAccelZ, maxAccelAngle) **maxJerk**: the maximum allowed jerk of the object (maxJerkX, maxJerkY, maxJerkZ, maxJerkAngle). With the RML type II plugin, the max. jerk will however always be infinite. **targetPosition**: the desired target position of the object (expressed relative to *relativeToObjectHandle*). Can be nil, in which case the position of the object will stay constant **targetQuaternion**: the desired target orientation of the object (expressed relative to *relativeToObjectHandle*). Can be nil, in which case the orientation of the object will stay constant **targetVel**: the desired velocity of the object at the target (targetVelX, targetVelY, targetVelZ,

	targetVelAngle). Can be nil in which case a velocity vector of 0 is used.
Lua return values	**result**: 1 if the function call was successful **newPos**: the new relative position of the object **newQuaternion**: the new relative orientation of the object **newVel**: the new velocity vector (velX, velY, velZ, velAngle) **newAccel**: the new acceleration vector (accelX, accelY, accelZ, accelAngle) **timeLeft**: the time left for additional calculations in current simulation time step

simRMLPos / sim.rmlPos

Description	Executes a call to the Reflexxes Motion Library type II or IV. The Reflexxes Motion Library provides instantaneous trajectory generation capabilities for motion control systems. This function prepares a position-based trajectory generation object, that can then be calculated with sim.rmlStep. When this object is not needed anymore, remove it with sim.rmlRemove. See also sim.rmlVel, sim.rmlMoveToPosition and sim.rmlMoveToJointPositions.
C synopsis	simInt simRMLPos(simInt dofs,simDouble smallestTimeStep,simInt flags,const simDouble* currentPosVelAccel,const simDouble* maxVelAccelJerk,const simBool* selection,const simDouble* targetPosVel,simVoid* auxData)
C parameters	**dofs**: the number of degrees of freedom (N). **smallestTimeStep**: the smallest expected cycle time. Use a value of 0.0001 (0.1ms). **flags**: RML flags. -1 for default flags. **currentPosVelAccel**: the current position, velocity and acceleration. Arrange values as {pos1,pos2,..,posN,vel1,vel2,..,velN,accel1,accel2,..,accelN} **maxVelAccelJerk**: the maximum allowed velocity, acceleration and jerk. Arrange values as {vel1,vel2,..,velN,accel1,accel2,..,accelN,jerk1,jerk2,..,jerkN}. With the RML type II plugin, the max. jerk will however always be infinite. **selection**: the selection vector (one value for each DoF). For a default behaviour, fill the vector with non-zero values. **targetPosVel**: the target position and velocity. Arrange values as {pos1,pos2,..,posN,vel1,vel2,..,velN} **auxData**: can be NULL. Otherwise in/out extension data. The first byte indicates how many additional in/out values we wish to set/get. Following auxiliary values can be set/get: value 1 (input): Bytes 2-5 (int): set to 1 if you wish this object to be automatically destroyed at simulation end (which is the default case when called from the main script or a child script).
C return value	-1 in case of an error, otherwise the handle of the created object.
Lua synopsis	number handle=sim.rmlPos(number dofs,number smallestTimeStep,number flags,table currentPosVelAccel,table maxVelAccelJerk,table selection,table targetPosVel) If you wish to use this function in a blocking mode, consider using sim.rmlMoveToPosition or sim.rmlMoveToJointPositions instead.
Lua parameters	Refer to the C-function documentation
Lua return values	Refer to the C-function documentation

simRMLPosition (DEPRECATED)

Description	DEPRECATED. See sim.rmlPos instead.

simRMLRemove / sim.rmlRemove

Description	Removes an object previously created via sim.rmlPos or sim.rmlVel.
C synopsis	simInt simRMLRemove(simInt handle)
C parameters	**handle**: the handle of the object created via simRMLPos or simRMLVel.
C return value	1 in case of success
Lua synopsis	number result=sim.rmlRemove(number handle)
Lua parameters	Refer to the C-function documentation
Lua return values	Refer to the C-function documentation

simRMLStep / sim.rmlStep

Description	Executes a call to the Reflexxes Motion Library type II or IV. The Reflexxes Motion Library provides instantaneous trajectory generation capabilities for motion control systems. This function steps forward a trajectory generation algorithm previously prepared via sim.rmlPos or sim.rmlVel.
C synopsis	simInt simRMLStep(simInt handle,simDouble timeStep,simDouble* newPosVelAccel,simVoid* auxData,simVoid* reserved)
C parameters	**handle**: the handle of the object created via simRMLPos or simRMLVel. **timeStep**: the cycle time **newPosVelAccl**: the new position, velocity and acceleration (output values). Values are arranged as {pos1,pos2,..,posN,vel1,vel2,..,velN,accel1,accel2,..,accelN} **auxData**: can be NULL. Otherwise in/out extension data. The first byte indicates how many additional in/out values we wish to set/get. Following auxiliary values can be set/get: value 1 (output): Bytes 2-9 (double): returns the **synchronization time** (the time needed to reach the desired state. This time does not include the cycle time of the current call to simRMLStep) **reserved**: reserved for future extensions. Set to NULL.
C return value	-1 in case of an error, otherwise the return value of function RMLPosition or RMLVelocity in the Reflexxes Motion Library: 1: final state reached 0: final state not yet reached -100: RML_ERROR_INVALID_INPUT_VALUES -101: RML_ERROR_EXECUTION_TIME_CALCULATION -102: RML_ERROR_SYNCHRONIZATION -103: RML_ERROR_NUMBER_OF_DOFS -104: RML_ERROR_NO_PHASE_SYNCHRONIZATION -105: RML_ERROR_NULL_POINTER -106: RML_ERROR_EXECUTION_TIME_TOO_BIG
Lua synopsis	number result,table newPosVelAccel,number synchronizationTime=sim.rmlStep(number handle,number timeStep) If you wish to use this function in a blocking mode, consider using sim.rmlMoveToPosition or sim.rmlMoveToJointPositions instead.
Lua parameters	Refer to the C-function documentation
Lua return values	Refer to the C-function documentation

simRMLVel / sim.rmlVel

Description	Executes a call to the Reflexxes Motion Library type II or IV. The Reflexxes Motion Library provides instantaneous trajectory generation capabilities for motion control systems. This function prepares a velocity-based trajectory generation object, that can then be calculated with sim.rmlStep. When this object is not needed anymore, remove it with sim.rmlRemove. See also sim.rmlPos.
C synopsis	simInt simRMLVel(simInt dofs,simDouble smallestTimeStep,simInt flags,const simDouble* currentPosVelAccel,const simDouble* maxAccelJerk,const simBool* selection,const simDouble* targetVel,simVoid* auxData)
C parameters	**dofs**: the number of degrees of freedom (N). **smallestTimeStep**: the smallest expected cycle time. Use a value of 0.0001 (0.1ms). **flags**: RML flags. -1 for default flags. **currentPosVelAccel**: the current position, velocity and acceleration. Arrange values as {pos1,pos2,..,posN,vel1,vel2,..,velN,accel1,accel2,..,accelN} **maxAccelJerk**: the maximum allowed acceleration and jerk. Arrange values as {accel1,accel2,..,accelN,jerk1,jerk2,..,jerkN}. With the RML type II plugin, the max. jerk will however always be infinite. **selection**: the selection vector (one value for each DoF). For a default behaviour, fill the vector with non-zero values. **targetVel**: the target velocity (one value for each DoF) **auxData**: can be NULL. Otherwise in/out extension data. The first byte indicates how many additional in/out values we wish to set/get. Following auxiliary values can be set/get: value 1 (input): Bytes 2-5 (int): set to 1 if you wish this object to be automatically destroyed at simulation end (which is the default case when called from the main script or a child script).
C return value	-1 in case of an error, otherwise the handle of the created object.

Lua synopsis	number handle=sim.rmlVel(number dofs,number smallestTimeStep,number flags,table currentPosVelAccel,table maxAccelJerk,table selection,table targetVel)
Lua parameters	Refer to the C-function documentation
Lua return values	Refer to the C-function documentation

simRMLVelocity (DEPRECATED)

Description	DEPRECATED. See sim.rmlVel instead.

simReadCollision / sim.readCollision (remote API equivalent: simxReadCollision)

Description	Reads the collision state of a registered collision object. This function doesn't perform collision detection, it merely reads the result from a previous call to sim.handleCollision (sim.handleCollision is called in the default main script). See also sim.resetCollision, sim.checkCollision and sim.checkCollisionEx.
C synopsis	simInt simReadCollision(simInt collisionObjectHandle)
C parameters	**collisionObjectHandle**: handle of the collision object
C return value	collision state (0 or 1), or -1 if operation was not successful.
Lua synopsis	number collisionState, table_2 collidingObjectHandles=sim.readCollision(number collisionObjectHandle)
Lua parameters	Same as C-function
Lua return values	**collisionState**: collision state (0 or 1), or -1 if operation was not successful. **collidingObjectHandles**: handles of the two colliding objects.

simReadCustomDataBlock / sim.readCustomDataBlock

Description	Reads custom data that is stored inside of an object, a script or a scene (i.e. the data is part of the object, the script or the scene). Reads also custom data for the application's currrent session. See also sim.writeCustomDataBlock, sim.readCustomDataBlockTags and the data packing/unpacking functions.
C synopsis	simChar* simReadCustomDataBlock(simInt objectHandle,const simChar* tagName,simInt* dataSize)
C parameters	**objectHandle**: handle of the object or script where the data is stored, or sim_handle_scene if the data is stored in the scene, or sim_handle_app if the data is stored in the application's current session. **tagName**: a string that identifies the data. **dataSize**: a pointer to an integer receiving the size of the returned buffer.
C return value	the custom data block, or nil in case of an error (or if the data is not present). The user is in charge of releasing the returned buffer with simReleaseBuffer.
Lua synopsis	string customDataBlock=sim.readCustomDataBlock(number objectHandle,string tagName)
Lua parameters	Same as C-function. In addition, you can specify sim.handle_self for the **objectHandle** argument, if your target is the current script.
Lua return values	Same as C-function.

simReadCustomDataBlockTags / sim.readCustomDataBlockTags

Description	Reads the tags of all custom data that is stored inside of an object, a script or a scene (i.e. the data is part of the object, the script or the scene). Reads also all custom data that is stored inside of the application's current session. See also sim.readCustomDataBlock.
C synopsis	simChar* simReadCustomDataBlockTags(simInt objectHandle,simInt* tagCount)
C parameters	**objectHandle**: handle of the object or script where the data is stored, or sim_handle_scene if the data is stored in the scene, or sim_handle_app if the data is stored in the application's current session. **tagCount**: a pointer to an integer receiving the number of tag strings contained in the returned buffer.
C return value	the tags (each followed by the zero-char) or nil in case of an error (or if no tags are present). The user is in charge of releasing the returned buffer with simReleaseBuffer.
Lua synopsis	table tags=sim.readCustomDataBlockTags(number objectHandle)
Lua parameters	Same as C-function. In addition, you can specify sim.handle_self for the **objectHandle** argument, if

	your target is the current script.
Lua return values	Similar to C-function.

simReadDistance / sim.readDistance (remote API equivalent: simxReadDistance)

Description	Reads the distance of a registered distance object. This function doesn't perform distance measurement, it merely reads the result from a previous call to sim.handleDistance (sim.handleDistance is called in the default main script). See also sim.resetDistance and sim.checkDistance.
C synopsis	simInt simReadDistance(simInt distanceObjectHandle,simFloat* smallestDistance)
C parameters	**distanceObjectHandle**: handle of the distance object **smallestDistance**: smallest distance (valid only if return value is different from -1)
C return value	different from -1 if distance was read, -1 in case of an error.
Lua synopsis	number result,number smallestDistance=sim.readDistance(number distanceObjectHandle)
Lua parameters	Same as C-function
Lua return values	**result**: 1 if distance was read, -1 in case of an error, and 0 if sim.handleDistance was never called, or if sim.resetDistance was previously called. **smallestDistance**: the smallest distance. Is nil if result is not 1

simReadForceSensor / sim.readForceSensor (remote API equivalent: simxReadForceSensor)

Description	Reads the force and torque applied to a force sensor (filtered values are read), and its current state ('unbroken' or 'broken'). See also sim.breakForceSensor and sim.getJointForce.
C synopsis	simInt simReadForceSensor(simInt objectHandle,simFloat* forceVector,simFloat* torqueVector)
C parameters	**objectHandle**: handle of the object (must be a force sensor). Can be combined with sim_handleflag_rawvalue (simply add sim_handleflag_rawvalue to objectHandle), if you wish to access the raw values generated by each individual dynamic simulation step (by default, there are 10 dynamic simulation steps for each simulation step). Raw values can only be accessed from inside a joint callback function. **forceVector**: pointer to 3 values (applied forces along the sensor's x, y and z-axes). Can be NULL **torqueVector**: pointer to 3 values (applied torques about the sensor's x, y and z-axes). Can be NULL
C return value	-1 in case of an error, otherwise bit-coded: bit 0 set (1): force and torque data is available, otherwise it is not (yet) available (e.g. when not enough values are present for the filter) bit 1 set (2): force sensor is broken, otherwise it is still intact ('unbroken')
Lua synopsis	number result, table_3 forceVector,table_3 torqueVector=sim.readForceSensor(number objectHandle)
Lua parameters	Same as C-function
Lua return values	**result**: -1 in case of an error, otherwise bit-coded (same as for the C-function return value) **forceVector**: table holding 3 values (applied forces along the sensor's x, y and z-axes) **torqueVector**: table holding 3 values (applied torques about the sensor's x, y and z-axes)

simReadProximitySensor / sim.readProximitySensor (remote API equivalent: simxReadProximitySensor)

Description	Reads the state of a proximity sensor. This function doesn't perform detection, it merely reads the result from a previous call to sim.handleProximitySensor (sim.handleProximitySensor is called in the default main script). See also sim.checkProximitySensor, sim.checkProximitySensorEx and sim.resetProximitySensor.
C synopsis	simInt simReadProximitySensor(simInt sensorHandle,simFloat* detectedPoint,simInt* detectedObjectHandle,simFloat* detectedSurfaceNormalVector)
C parameters	**sensorHandle**: handle of a proximity sensor object **detectedPoint**: coordinates of the closest detected point (x, y and z: detectedPoint[0]-detectedPoint[2]) relative to the sensor reference frame, and distance to the detected point (1 value: detectedPoint[3]). Can be NULL **detectedObjectHandle**: handle of the object that was detected. Can be NULL **detectedSurfaceNormalVector**: normal vector (normalized) of the detected surface. Relative to the sensor reference frame. Can be NULL
C return value	detection state (0 or 1), or -1 in case of an error, or if simHandleProximitySensor was never called, or if

	simResetProximitySensor was previously called.
Lua synopsis	number result,number distance,table_3 detectedPoint,number detectedObjectHandle,table_3 detectedSurfaceNormalVector=sim.readProximitySensor(number sensorHandle)
Lua parameters	Same as C-function
Lua return values	**result**: detection state (0 or 1), or -1 in case of an error, or if sim.handleProximitySensor was never called, or if sim.resetProximitySensor was previously called. **distance**: distance to the detected point if result is 1, nil otherwise **detectedPoint**: table of 3 numbers indicating the relative coordinates of the detected point if result is 1, nil otherwise **detectedObjectHandle**: handle of the object that was detected if result is 1, nil otherwise **detectedSurfaceNormalVector**: normal vector (normalized) of the detected surface. Relative to the sensor reference frame. Is nil if result is different from 1

simReadTexture / sim.readTexture

Description	Retrieves the RGB data (or a portion of it) related to a specific texture. See also sim.getTextureId, sim.writeTexture and sim.createTexture.
C synopsis	simChar* simReadTexture(simInt textureId,simInt options,simInt posX,simInt posY,simInt sizeX,simInt sizeY)
C parameters	**textureId**: the ID of the texture. See also simGetTextureId. **options**: reserved for future functionality. Set to zero. **posX / posY**: the x/y position of the texture portion to retrieve. Set to 0/0 to retrieve the full texture **sizeX / sizeY**: the x/y size of the texture portion to retrieve. Set to 0/0 to retrieve the full texture
C return value	The texture data, or NULL in case of an error. The texture data contains RGB values between 0-255 (3 bytes per pixel). The user is in charge of releasing the returned buffer with simReleaseBuffer.
Lua synopsis	string textureData=sim.readTexture(number textureId,number options,number posX=0,number posY=0,number sizeX=0,number sizeY=0)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simReadVisionSensor / sim.readVisionSensor (remote API equivalent: simxReadVisionSensor)

Description	Reads the state of a vision sensor. This function doesn't perform detection, it merely reads the result from a previous call to sim.handleVisionSensor (sim.handleVisionSensor is called in the default main script). See also sim.checkVisionSensor, sim.checkVisionSensorEx and sim.resetVisionSensor.
C synopsis	simInt simReadVisionSensor(simInt visionSensorHandle,simFloat** auxValues,simInt** auxValuesCount)
C parameters	**visionSensorHandle**: handle of a vision sensor object **auxValues**: auxiliary values returned from the applied filters. By default V-REP returns one packet of 15 auxiliary values:the minimum of {intensity, red, green, blue, depth value}, the maximum of {intensity, red, green, blue, depth value}, and the average of {intensity, red, green, blue, depth value}. If additional filter components return values, then they will be appended as packets to the first packet. AuxValues can be NULL. The user is in charge of releasing the auxValues buffer with simReleaseBuffer(*auxValues). **auxValuesCount**: contains information about the number of auxiliary value packets and packet sizes returned in auxValues. The first value is the number of packets, the second is the size of packet1, the third is the size of packet2, etc. Can be NULL if auxValues is also NULL. The user is in charge of releasing the auxValuesCount buffer with simReleaseBuffer(*auxValuesCount). See simHandleVisionSensor for a usage example
C return value	detection state (0 or 1), or -1 in case of an error, or if simHandleVisionSensor was never called, or if simResetVisionSensor was previously called.
Lua synopsis	number result,table auxiliaryValuePacket1,table auxiliaryValuePacket2, etc.=sim.readVisionSensor(number visionSensorHandle)
Lua parameters	**visionSensorHandle**: handle of a vision sensor object
Lua return values	**result**: detection state (0 or 1), or -1 in case of an error, or if sim.handleVisionSensor was never called, or if sim.resetVisionSensor was previously called. **auxiliaryValuePacket1**: default auxiliary value packet (same as for the C-function) **auxiliaryValuePacket2**: additional auxiliary value packet (e.g. from a filter component) **auxiliaryValuePacket3**: etc. (the function returns as many tables as there are auxiliary value packets)

simReceiveData / sim.receiveData

Description	Receives wireless data (in a simulation). See also sim.sendData and sim.tubeOpen. Cannot be called from add-ons.
	
	Wireless receptions can be visualized globally via the environment dialog, or individually as in following example:
	
	`sim.setBoolParameter(sim.boolparam_force_show_wireless_reception,true)`
	`data=sim.receiveData(...)`
	`sim.setBoolParameter(sim.boolparam_force_show_wireless_reception,false)`

C synopsis	simReceiveData(simInt dataHeader,const simChar* dataName,simInt antennaHandle,simInt index,simInt* dataLength,simInt* senderID,simInt* dataHeaderR,simChar** dataNameR)
C parameters	**dataHeader**: number indicating who "designed" the communication message. Can also be -1, in which case messages with any dataHeader will be retrieved (not recommended, unless index is different from -1). **dataName**: name indicating the type of message. Can be nil, in which case messages with any dataName will be retrieved (not recommended, unless index is different from -1) **antennaHandle**: handle of the scene object that should operate as the antenna for this reception. If sim_handle_default is specified, a reception antenna at (0,0,0) is simulated. **index**: zero-based index of the message to read. If -1 is indicated, the first message that matches the dataHeader and dataName is read and removed. Otherwise messages are just read. **dataLength**: length of the received data (if returned pointer is not NULL) **senderID**: identifier of the sender. Can be the handle of a script if the message was sent from a script, or can be 0 if the message was sent from the non-Lua API. Can be NULL. **dataHeaderR**: dataHeader of the data that was read. Can be NULL. **dataNameR**: dataName of the data that was read. Can be NULL. The user is in charge of releasing the buffer with simReleaseBuffer(*dataNameR).
C return value	pointer to the received data, or NULL if no data is available or in case of an error. The user is in charge of releasing the returned buffer with simReleaseBuffer.
Lua synopsis	string data,number senderID,number dataHeader,string dataName=sim.receiveData(number dataHeader=-1,string dataName=nil,number antennaHandle=sim.handle_self,number index=-1)
Lua parameters	**dataHeader**: number indicating who "designed" the communication message. Can also be -1, in which case messages with any dataHeader will be retrieved (not recommended, unless index is different from -1). This value can be omitted (-1 will be used). **dataName**: name indicating the type of message. Can be nil, in which case messages with any dataName will be retrieved (not recommended, unless index is different from -1). This value can be omitted (nill will be used) **antennaHandle**: handle of the scene object that should operate as the antenna for this reception. If sim.handle_default is specified, a reception antenna at (0,0,0) is simulated. If sim.handle_self is specified, the object associated with the current child script is used as the antenna. This value can be omitted (sim.handle_self will be used) **index**: zero-based index of the message to read. If -1 is indicated, the first message that matches the dataHeader and dataName is read and removed. Otherwise messages are just read. This value can be omitted (-1 will be used)
Lua return values	**data**: string containing the received data, or nil in case of an error or if no data is present. If received data is packed, see also the data packing/unpacking functions **senderID**: identifier of the sender. Can be the handle of a script if the message was sent from a script, or can be 0 if the message was sent from the non-Lua API. **dataHeader**: dataHeader of the data that was read. **dataName**: dataName of the data that was read.

simRefreshDialogs / sim.refreshDialogs

Description	Refreshes V-REP's internal dialogs. Calling sim.refreshDialogs will not trigger a sim_message_eventcallback_refreshdialogs message
C synopsis	simInt simRefreshDialogs(simInt refreshDegree)
C parameters	**refreshDegree**: refresh degree (0=light, 1=medium, 2=full)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available

Lua synopsis	number result=sim.refreshDialogs(number refreshDegree)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simRegisterContactCallback (DEPRECATED)

Description	DEPRECATED.

simRegisterCustomLuaFunction (DEPRECATED)

Description	This function is deprecated. Use simRegisterScriptCallbackFunction instead.

simRegisterCustomLuaVariable (DEPRECATED)

Description	DEPRECATED. See simRegisterScriptVariable instead.

simRegisterJointCtrlCallback (DEPRECATED)

Description	DEPRECATED.

simRegisterScriptCallbackFunction

Description	Registers a customized script function. This function is useful for plugins (or the main client application) that wish to provide their own or customized script functions. See also simRegisterScriptVariable. Data exchange between a script and the plugin happens via a stack. Reading and writing arguments from/to the stack gives you a maximum of flexibility, and you wil be able to exchange also complex data structures. But it can also be tedious, if your data structures are anyway relatively simple. In that case you can use the helper classes *CScriptFunctionData* and *CScriptFunctionDataItem* located in *programming/common* and *programming/include*: they will greatly simplify the task. Use following 4 functions in the helper class: *readDataFromStack*, *getInDataPtr*, *pushOutData* and *writeDataToStack*.
C synopsis	simInt simRegisterScriptCallbackFunction(const simChar* funcNameAtPluginName,const simChar* callTips,simVoid(*callBack)(struct SScriptCallBack* cb))
C parameters	**functNameAtPluginName**: name of the function, combined with the plugin name: functionName@pluginName. Avoid using too simple function names, otherwise they might clash with other plugins. Also, always use the *simXX.* prefix (e.g. *simMyPlugin.myCustomFunction*) for the function name. The plugin name should be the exact same name used while loading the plugin via simLoadModule (if the plugin name is *v_repExtMyPlugin.dll*, this should be *MyPlugin*). **callTips**: call tips: string (or several strings separated by '@') that indicates the input/output argument type/size. Call tips appear in the script editor when the function was typed followed by "(". callTips Can be NULL, in which case no call tips will be displayed, nor syntax highlighting used. **callback**: callback address that is called when the "functName" function is called from Lua. Can be NULL, in which case the command will only register the function for call tips and syntax highlighting. See further down for a simple way to call above function, using a helper class. The callback's first argument is a SScriptCallBack structure that holds: **simInt objectID**: handle of the object that the calling script is attached to, or -1 if the calling script is not a child script **simInt scriptID**: handle of the calling script **simInt stackID**: a stack handle. The stack is used to read arguments from the script, and to return

	data to the script. See also the available stack functions. **simChar waitUntilZero**: this value can be used when threaded scripts call a custom Lua function in a plugin that shouldn't return until a condition is met (e.g. until the robot movement finished). For that purpose, the plugin should write a value different from zero to indicate a "wait" state. When the callback returns, the control is not given back to the script until some other thread calling the plugin writes zero to that location. Once zero was written, the memory location should not be used anymore (because it might be released anytime by the simulator). Also, when the user stops a simulation before zero was written to that location, the wait state is aborted. In that case however the memory location stays valid (i.e. writing zero will not result in a crash) until the simulation ended. **simChar* raiseErrorWithMessage**: max. 256 bytes are available here for an error message. This buffer is allocated by V-REP and initially contains a zero length string. If the length of the string is not zero, a script error will be raised and the message displayed in the status bar.
C return value	1 if function was registered, 0 if function was replaced (when that function name already existed), -1 in case of an error
Lua synopsis	-
Lua parameters	-
Lua return values	-

simRegisterScriptVariable

Description	Registers a script variable. Each time a script is run for the first time, registered variables will be set. Can also be used for more complex operations as in following example: simRegisterScriptVariable("simUi","require('customUi')",0); which is equivalent with an implicit simUi=require('customUi') command in the initialization phase of every script. See also simRegisterScriptCallbackFunction and sim.setScriptVariable.
C synopsis	simInt simRegisterScriptVariable(const simChar* varName,const simChar* varValue)
C parameters	**varName**: name of the variable **varValue**: value of the variable. Can be NULL, in which case the value of the variable will be the top item of the provided stack. **stackHandle**: a stack handle obtained with simCreateStack. Set to 0 if *varValue* is not NULL. Set to -1 if *varValue* is not NULL and you do not want the variable to appear in the script editor auto-completion list.
C return value	1 if the variable was registered, 0 if the variable was replaced because it already existed, -1 in case of an error
Lua synopsis	-
Lua parameters	-
Lua return values	-

simReleaseBuffer (remote API equivalent: simxReleaseBuffer)

Description	Releases a buffer previously created with simCreateBuffer or a buffer returned by the simulator. You will have to cast some buffer pointers to (simChar*).
C synopsis	simInt simReleaseBuffer(simChar* buffer)
C parameters	**buffer**: buffer to be released
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	-
Lua parameters	-
Lua return values	-

simReleaseScriptRawBuffer

Description	Removes (destroys) a script's raw buffer. All raw buffers are automatically released at the end of a simulation, but depending on the extent of use of the raw buffers, it might be a good idea to release them when not needed anymore. See also simGetScriptRawBuffer and simSetScriptRawBuffer.

C synopsis	simInt simReleaseScriptRawBuffer(simInt scriptHandle,simInt bufferHandle)
C parameters	**scriptHandle**: handle of the script. Can be sim_handle_main_script or sim_handle_all (in that case the bufferHandle is automatically also sim_handle_all) **bufferHandle**: handle of the raw buffer or sim_handle_all to release all raw buffers of the given script
C return value	-1 if operation was not successful (an unexisting buffer will not trigger an error). In a future release, a more differentiated return value might be available
Lua synopsis	-
Lua parameters	-
Lua return values	-

simReleaseStack

Description	Releases a stack object. See also the other stack functions.
C synopsis	simInt simReleaseStack(simInt stackHandle)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack.
C return value	1 in case of success.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simRemoveBanner / sim.removeBanner

Description	Removes a previously added banner. See also sim.addBanner
C synopsis	simInt simRemoveBanner(simInt bannerID)
C parameters	**bannerID**: handle of a previously added banner. *sim_handle_all* removes all banners from the scene. If you or-combine the bannerID with *sim_handleflag_togglevisibility*, then you can toggle the banner's visibility without removing it: in that case, the return value is the new visibility stae of the banner.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.removeBanner(number bannerID)
Lua parameters	**bannerID**: handle of a previously added banner. sim.handle_all removes all banners from the scene, that were created in a script (banners created from the C interface are not removed)
Lua return values	Same as C-function

simRemoveCollection / sim.removeCollection

Description	Removes a collection from the scene, including the objects it contains. If you just want to remove the collection, but not the objects it contains, call sim.emptyCollection beforehand. See also sim.createCollection and sim.addObjectToCollection.
C synopsis	simInt simRemoveCollection(simInt collectionHandle)
C parameters	**collectionHandle**: handle of the collection to remove. sim_handle_all removes all collections from the scene
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.removeCollection(number collectionHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simRemoveDrawingObject / sim.removeDrawingObject

Description	Removes a previously added drawing object. See also sim.addDrawingObject and sim.addDrawingObjectItem

C synopsis	simInt simRemoveDrawingObject(simInt objectHandle)
C parameters	**objectHandle**: handle of a previously added drawing object. sim_handle_all removes all drawing objects from the scene
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.removeDrawingObject(number objectHandle)
Lua parameters	**objectHandle**: handle of a previously added drawing object. sim.handle_all removes all drawing objects from the scene, that were created in a script (drawing objects created from the C interface are not removed)
Lua return values	Same as C-function

simRemoveIkGroup / sim.removeIkGroup

Description	Removes an IK group. See also sim.getIkGroupHandle and sim.createIkGroup.
C synopsis	simInt simRemoveIkGroup(simInt ikGroupHandle)
C parameters	**ikGroupHandle**: handle of an IK group.
C return value	-1 if operation was not successful.
Lua synopsis	number result=sim.removeIkGroup(number ikGroupHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simRemoveModel / sim.removeModel (remote API equivalent: simxRemoveModel)

Description	Removes a model from the scene. See also sim.removeObject. Threaded child scripts can only destroy models that do not contain other scripts attached than itself. Object destruction always tries to destroy attached scripts before destroying the object itself. If a script tries to destroy the object it is attached to, then the object will first be destroyed, and the script destruction will be delayed.
C synopsis	simInt simRemoveModel(simInt objectHandle)
C parameters	**objectHandle**: handle of the model (i.e. object tagged as model) to remove.
C return value	-1 if operation was not successful, otherwise the number of removed objects (a model might contain several objects)
Lua synopsis	number removedCnt=sim.removeModel(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simRemoveMotionPlanning (DEPRECATED)

Description	DEPRECATED. See the OMPL library based path/motion planning functionality instead. Removes a motion planning task. See also simGetMotionPlanningHandle and simCreateMotionPlanning.
C synopsis	simInt simRemoveMotionPlanning(simInt motionPlanningHandle)
C parameters	**motionPlanningHandle**: handle of a motion planning task.
C return value	-1 if operation was not successful.
Lua synopsis	number result=simRemoveMotionPlanning(number motionPlanningHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simRemoveObject / sim.removeObject (remote API equivalent: simxRemoveObject)

Description	Removes an object from the scene. See also sim.removeModel. Threaded child scripts can only destroy objects that do not contain other scripts attached than itself.

	Object destruction always tries to destroy attached scripts before destroying the object itself. If a script tries to destroy the object it is attached to, then the object will first be destroyed, and the script destruction will be delayed.
C synopsis	simInt simRemoveObject(simInt objectHandle)
C parameters	**objectHandle**: handle of the object to remove. sim_handle_all removes all objects from the scene
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.removeObject(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simRemoveObjectFromSelection / sim.removeObjectFromSelection

Description	Removes an object from the selection. See also sim.addObjectToSelection, sim.isObjectInSelection and sim.getObjectSelection.
C synopsis	simInt simRemoveObjectFromSelection(simInt what,simInt objectHandle)
C parameters	**What**: indicates what we wish to remove from the selection. Valid values are sim_handle_single (removes one object from the selection), sim_handle_all (removes all objects from the selection), sim_handle_tree (removes the tree with base objectHandle (inclusive) from the selection, sim_handle_chain (removes the chain with tip objectHandle (inclusive) from the selection. **objectHandle**: handle of the object to remove from the selection. If sim_handle_all is specifies in "what", then objectHandle is ignored
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	(1) number result=sim.removeObjectFromSelection(number what,number objectHandle) (2) number result=sim.removeObjectFromSelection(table objectHandles)
Lua parameters	(1) Same as C-function. The second argument can be omitted if "what" is sim.handle_all (2) objectHandles: table containing the handles of objects to remove. Can be nil, in which case nothing happens
Lua return values	Same as C-function

simRemoveParticleObject / sim.removeParticleObject

Description	Removes a previously added particle object. See also sim.addParticleObject and sim.addParticleObjectItem
C synopsis	simInt simRemoveParticleObject(simInt objectHandle)
C parameters	**objectHandle**: handle of a previously added particle object. sim_handle_all removes all particle objects from the scene
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.removeParticleObject(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simRemovePointsFromPointCloud / sim.removePointsFromPointCloud

Description	Removes points from a point cloud. When a point cloud doesn't use an octree calculation structure, then individual points cannot be removed, only all points can be removed in that case. See also sim.insertPointsIntoPointCloud, sim.setPointCloudOptions and the other point cloud related functions.
C synopsis	simInt simRemovePointsFromPointCloud(simInt pointCloudHandle,simInt options,const simFloat* pts,simInt ptCnt,simFloat tolerance,simVoid* reserved)
C parameters	**pointCloudHandle**: the handle of the point cloud. See also simGetObjectHandle **options**: bit-coded: bit0 set (1): specified points are relative to the point cloud reference frame, otherwise they are relative to the world reference frame **pts**: a pointer to the point positions specified as X/Y/Z coordinates. Set to NULL to remove all points **ptCnt**: the number of point coordinates contained in pts

	tolerance: a distance used as a tolerance value **reserved**: reserved for future extensions. Set to NULL
C return value	-1 if operation was not successful, otherwise the total number of points in the point cloud
Lua synopsis	number totalPointCnt=sim.removePointsFromPointCloud(number pointCloudHandle,number options,table points,number tolerance)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simRemoveScript / sim.removeScript

Description	Removes a script. Not all script types can be removed, and it will also depend on whether simulation is running or not. See also sim.addScript.
C synopsis	simInt simRemoveScript(simInt scriptHandle)
C parameters	**scriptHandle**: handle of the script to remove or sim_handle_all to remove all scripts
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.removeScript(number scriptHandle)
Lua parameters	Same as C-function. Additionally you can specify sim.handle_self to have the script remove itself.
Lua return values	Same as C-function

simRemoveUI (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simRemoveVoxelsFromOctree / sim.removeVoxelsFromOctree

Description	Removes voxels from an octree. See also sim.insertVoxelsIntoOctree and the other octree related functions.
C synopsis	simInt simRemoveVoxelsFromOctree(simInt octreeHandle,simInt options,const simFloat* pts,simInt ptCnt,simVoid* reserved)
C parameters	**octreeHandle**: the handle of the octree. See also simGetObjectHandle **options**: bit-coded: bit0 set (1): specified points are relative to the octree reference frame, otherwise they are relative to the world reference frame **pts**: a pointer to the voxel positions specified as X/Y/Z coordinates. Set to NULL to remove all voxels **ptCnt**: the number of point coordinates contained in pts **reserved**: reserved for future extensions. Set to NULL
C return value	-1 if operation was not successful, otherwise the total number of voxels in the octree
Lua synopsis	number totalVoxelCnt=sim.removeVoxelsFromOctree(number octreeHandle,number options,table points)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simReorientShapeBoundingBox / sim.reorientShapeBoundingBox

Description	Reorients the bounding box of a shape.
C synopsis	simInt simReorientShapeBoundingBox(simInt shapeHandle,simInt relativeToHandle,simInt reservedSetToZero)
C parameters	**shapeHandle**: the handle of the shape that will be reoriented. See also simGetObjectHandle. **relativeToHandle**: the handle of an object relative to which the shape should be reoriented, or -1 to align the bounding box with the world, or sim_handle_self to build the smallest bounding box around the object. **reservedSetToZero**: reserved for future extensions. Set to 0.

C return value	-1 if operation was not successful. 0 if the bounding box could not be reoriented (the bounding box of pure shapes cannot be reoriented), otherwise 1.
Lua synopsis	number result=sim.reorientShapeBoundingBox(number shapeHandle,number relativeToHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simResetCollision / sim.resetCollision

Description	Clears the collision state, colors, intersections, etc. for a registered collision object. See also sim.handleCollision.
C synopsis	simInt simResetCollision(simInt collisionObjectHandle)
C parameters	**collisionObjectHandle**: handle of the collision object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will reset all registered collision objects, while sim_handle_all_except_explicit will only reset those that are not marked as "explicit handling")
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.resetCollision(number collisionObjectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simResetDistance / sim.resetDistance

Description	Clears the distance state, the distance segment, etc. for a registered distance object. See also sim.handleDistance.
C synopsis	simInt simResetDistance(simInt distanceObjectHandle)
C parameters	**distanceObjectHandle**: handle of the distance object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will reset all registered distance objects, while sim_handle_all_except_explicit will only reset those that are not marked as "explicit handling")
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.resetDistance(number distanceObjectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simResetDynamicObject / sim.resetDynamicObject

Description	Dynamically resets an object that is dynamically simulated. This means that the object representation in the dynamics engine is removed, and added again. This can be useful when the set-up of a dynamically simulated chain needs to be modified during simulation (e.g. joint or shape attachement position/orientation changed). It should be noted that calling this on a dynamically simulated object might slightly change its position/orientation relative to its parent (since the object will be disconnected from the dynamics world in its current position/orientation), so the user is in charge of rectifying for that.
C synopsis	simInt simResetDynamicObject(simInt objectHandle)
C parameters	**objectHandle**: handle of the object or sim_handle_all (to reset all dynamic content in the scene). **objectHandle** can be combined with sim_handleflag_model, if you wish to reset all objects in a model (where objectHandle would be the model base).
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.resetDynamicObject(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simResetGraph / sim.resetGraph

Description	Clears a graph object (resets all its data streams). See also sim.handleGraph.
C synopsis	simInt simResetGraph(simInt graphHandle)
C parameters	**graphHandle**: handle of the graph object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will reset all graph objects, while sim_handle_all_except_explicit will only reset those that are not marked as "explicit handling")
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.resetGraph(number graphHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simResetJoint (DEPRECATED)

Description	DEPRECATED.

simResetMill / sim.resetMill

Description	Clears all previously set mill calculation results (list of cut objects, surface and volume that were cut). See also sim.handleMill.
C synopsis	simInt simResetMill(simInt millHandle)
C parameters	**millHandle**: handle of the mill object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will reset all mill objects, while sim_handle_all_except_explicit will only reset those that are not marked as "explicit handling")
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.resetMill(number millHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simResetMilling / sim.resetMilling

Description	Resets a cuttable object (e.g. a shape) to its initial shape (before it was milled), thus cancelling milling changes. The calculation structure linked to the object is removed and an updated calculation structure might be calculated (might take some calculation time). See also sim.applyMilling, sim.handleMill and sim.resetMill.
C synopsis	simInt simResetMilling(simInt objectHandle)
C parameters	**objectHandle**: handle of the cut object or sim_handle_all to reset all cut objects.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.resetMilling(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simResetPath (DEPRECATED)

Description	DEPRECATED.

simResetProximitySensor / sim.resetProximitySensor

Description	Clears the detection state, detection color, detection segments, etc. of a proximity sensor object. See also sim.handleProximitySensor.
C synopsis	simInt simResetProximitySensor(simInt sensorHandle)
C parameters	**sensorHandle**: handle of the proximity sensor object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will reset all proximity sensor objects, while sim_handle_all_except_explicit will only reset those that are not marked as "explicit handling")
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.resetProximitySensor(number sensorHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simResetScript

Description	Resets a script interpreter linked to a specific script.
C synopsis	simInt simResetScript(simInt scriptHandle)
C parameters	**scriptHandle**: handle of the script to reset or sim_handle_all to reset all scripts
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	-
Lua parameters	-
Lua return values	-

simResetVisionSensor / sim.resetVisionSensor

Description	Clears the detection state, etc. of a proximity sensor object. See also sim.handleVisionSensor.
C synopsis	simInt simResetVisionSensor(simInt sensorHandle)
C parameters	**sensorHandle**: handle of the vision sensor object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will reset all vision sensor objects, while sim_handle_all_except_explicit will only reset those that are not marked as "explicit handling")
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.resetVisionSensor(number sensorHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simResumeThreads / sim.resumeThreads

Description	In conjunction with sim.setThreadResumeLocation, sim.resumeThreads allows specifying when and in which order threads are resumed. By default, V-REP doesn't use "regular" threads, but something similar to hybrid threads (which behave like coroutines, but can also behave like regular threads). This allows much more flexibility and execution control of the threads. Once a thread switched to another thread, it will resume execution at the beginning of next simulation pass by default. In order to also have full synchronization control between threads, you can assign a resume location and order to each thread. When sim.resumeThreads(x) is called, all threads that were assigned a resume location of x will be resumed. See also sim.setThreadResumeLocation, sim.setThreadSwitchTiming, sim.switchThread and sim.setThreadIsFree. This function can only be called in the main script.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number count=sim.resumeThreads(number location)
Lua parameters	**location**: indicates a location value associated with threads (through the sim.setThreadResumeLocation function). Only threads with the same location value will be resumed.

Lua return values	**result**: number of resumed threads, or -1 in case of an error.

simRotateAroundAxis / sim.rotateAroundAxis

Description	Rotates a transformation matrix around a random axis in space. This function, when used in combination with sim.getRotationAxis, can be used to build interpolations between transformation matrices. See also sim.getObjectMatrix, sim.setObjectMatrix and the other matrix/transformation functions.
C synopsis	simInt simRotateAroundAxis(const simFloat* matrixIn,const simFloat* axis,const simFloat* axisPos,simFloat angle,simFloat* matrixOut)
C parameters	**matrixIn**: the transformation matrix to rotate **axis**: the axis vector in absolute coordinates to rotate around **axisPos**: the position of the rotation axis in absolute coordinates **angle**: the amount of rotation to perform **matrixOut**: the returned transformed (rotated) matrix
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_12 matrixOut=sim.rotateAroundAxis(table_12 matrixIn,table_3 axis,table_3 axisPos,number angle)
Lua parameters	Same as C-function
Lua return values	**matrixOut**: the transformed (rotated) matrix, or nil in case of an error

simRunSimulator

Description	Runs the simulator. Should be the first and last command run. This will launch the main simulator loop. See also sim.quitSimulator and the section on the main client application.
C synopsis	simInt simRunSimulator(const simChar* applicationName,simInt options,simVoid(*initCallBack)(),simVoid(*loopCallBack)(),simVoid(*deinitCallBack)())
C parameters	**applicationName**: name of the application **options**: start-up options (combine them with the OR operator) **initCallBack**: the call-back address of the initialization routine. The initialization routine will be called just once, and should be used to load plugins for instance. Can be NULL **loopCallBack**: the call-back address of the main simulator loop. That routine is called continuously in a loop, and should react to simulator messages (simGetSimulatorMessage), and handle running simulations. Can be NULL **deinitCallBack**: the call-back address of the deinitialization routine. The deinitialization routine will be called just once, before the simulation ends, and should be used to unload plugins for instance. Can be NULL
C return value	1 if the command was successfull
Lua synopsis	-
Lua parameters	-
Lua return values	-

simSaveImage / sim.saveImage

Description	Saves an image to file or to memory. See also sim.loadImage, sim.getScaledImage and sim.getVisionSensorCharImage.
C synopsis	simSaveImage(const simUChar* image,const simInt* resolution,simInt options,const simChar* filename,simInt quality,simVoid* reserved)
C parameters	**image**: a pointer to rgb or rgba values. **resolution**: the x/y resolution of the provided image. **options**: bit-coded. If bit0 is set (1), then the provided image is rgba, otherwise it is rgb. **filename**: the name of the file to write. The file extension indicates the format. **quality**: the quality of the written image: 0 for best compression, 100 for largest file. Use -1 for default behaviour. **reserved**: Reserved for future extension. Set to NULL.
C return value	-1 if operation was not successful.

Lua synopsis	1) number result=sim.saveImage(string image,table_2 resolution,number options,string filename,number quality) 2) string imgBuffer=sim.saveImage(string image,table_2 resolution,number options,string filename,number quality)
Lua parameters	**image**: a pointer to rgb or rgba values. **resolution**: the x/y resolution of the provided image. **options**: bit-coded. If bit0 is set (1), then the provided image is rgba, otherwise it is rgb. **filename**: the name of the file to write. The file extension indicates the format. If the filename only contains '.xxx', where xxx represents the file format, then the image will be saved to memory **quality**: the quality of the written image: 0 for best compression, 100 for largest file. Use -1 for default behaviour.
Lua return values	1) **result**: -1 if operation was not successful 2) **imgBuffer**: a buffer containing the image in packed format (e.g. png, jpg, etc.)

simSaveModel / sim.saveModel

Description	Saves a model (an object marked as "Object is model base" and all other objects in its hierarchy tree). Any existing file with same name will be overwritten. See also sim.loadModel, and sim.saveScene.
C synopsis	simInt simSaveModel(int baseOfModelHandle,const simChar* filename)
C parameters	**baseOfModelHandle**: handle of an object marked as "Object is model base" **filename**: model filename. The filename extension is required ("ttm")
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	a) number result=sim.saveModel(number baseOfModelHandle,string filename) b) string buffer=sim.saveModel(number baseOfModelHandle)
Lua parameters	**baseOfModelHandle**: handle of an object marked as "Object is model base" **filename**: model filename. The filename extension is required ("ttm"). If this argument is omitted, then the model will be saved to a buffer which will be returned
Lua return values	a) -1 if operation was not successful b) nil if operation was not successful, or a buffer containing the saved model

simSaveScene / sim.saveScene

Description	Saves a scene. Any existing file with same name will be overwritten. See also sim.loadScene, simCloseScene, and sim.saveModel.
C synopsis	simInt simSaveScene(const simChar* filename)
C parameters	**filename**: scene filename. The filename extension is required ("ttt")
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.saveScene(string filename)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSaveUI (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simScaleObject / sim.scaleObject

Description	Scales specified objects in a non-isometric fashion, if possible (i.e. some objects can be fully isometrically scaled, others have 2 or all 3 axes linked). See also sim.scaleObjects for isometric scaling.
C synopsis	simInt simScaleObject(simInt objectHandle,simFloat xScale,simFloat yScale,simFloat zScale,simInt options)
	objectHandle: the handle of the object to scale.

C parameters	**xScale/yScale/zScale**: the scaling factors along the object's x, y and z-axis. **options**: reserved for future extension. Keep at 0.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.scaleObject(number objectHandle,number xScale,number yScale,number zScale,number options=0)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simScaleObjects / sim.scaleObjects

Description	Scales (in dimensions) specified objects. All related values are automatically scaled appropriately (e.g. masses, forces, etc.). See also sim.scaleObject for non-isometric scaling.
C synopsis	simInt simScaleObjects(const simInt* objectHandles,simInt objectCount,simFloat scalingFactor,simBool scalePositionsToo)
C parameters	**objectHandles**: an array containing the handles of the objects to scale. If an object is a model base, all its child objects will also be scaled. **objectCount**: the number of handles in the objectHandles array. **scalingFactor**: the scaling factor **scalePositionsToo**: if true, selected object's positions will also be scaled
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.scaleObjects(table objectHandles,number scalingFactor,boolean scalePositionsToo)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simScaleSelectedObjects (DEPRECATED)

Description	DEPRECATED. See sim.scaleObjects instead.

simSearchPath (DEPRECATED)

Description	DEPRECATED. See the OMPL library based path/motion planning functionality instead. Searches for a path. A registered path planning object is required (path planning objects can be registered in the scene editor). When called from C or from a non-threaded script, then this function will be blocking. When called from a threaded child script, this function is still blocking, but will switch to other threads at the specified time interval (subTimeStep). See also simGetPathPlanningHandle, simInitializePathSearch and simPerformPathSearchStep.
C synopsis	simInt simSearchPath(simInt pathPlanningObjectHandle,simFloat maxSearchTime)
C parameters	**pathPlanningObjectHandle**: handle of the path planning object **maxSearchTime**: maximum search time in seconds
C return value	-1 if operation was not successful (error), 0 if no path was found, 1 if a partial path was found or 2 if a full path was found (check the path planning task settings dialog)
Lua synopsis	number result=simSearchPath(number pathPlanningObjectHandle,number maxSearchTime,subTimeStep)
Lua parameters	**pathPlanningObjectHandle**: handle of the path planning object **maxSearchTime**: maximum search time in seconds **subTimeStep**: the delay after which the thread will switch to other threads (meaningful only for threaded child scripts). Can be nil or ignored, in which case a default value of 0.05 is used.
Lua return values	-1 if operation was not successful (error), 0 if no path was found, 1 if a partial path was found or 2 if a full path was found (check the path planning task settings dialog)

simSendData / sim.sendData

Description	Sends (or broadcasts) wireless data (in a simulation). See also sim.receiveData and sim.tubeOpen. Cannot be called from add-ons.
	
	Wireless emissions can be visualized globally via the environment dialog, or individually as in following example:
	
	`sim.setBoolParameter(sim.boolparam_force_show_wireless_emission,true)`
	`sim.sendData(...)`
	`sim.setBoolParameter(sim.boolparam_force_show_wireless_emission,false)`

C synopsis	simInt simSendData(simInt targetID,simInt dataHeader,const simChar* dataName,const simChar* data,simInt dataLength,simInt antennaHandle,simFloat actionRadius,simFloat emissionAngle1,simFloat emissionAngle2,simFloat persistence)
C parameters	**targetID**: indicates what receivers will receive the message. Can be sim_handle_all, or the handle of a script **dataHeader**: number indicating who "designed" the communication message. Always use the same header (because only you will know the meaning of the message) and stick to it. The best is to use the serial number of your V-REP copy (check the "Help" menu, in the "About" item for the serial number). Otherwise, you risk collision with other developer's messages which might use the same header as yours. **dataName**: name indicating the type of message. dataHeader and dataName will be used to filter out all unwanted messages when trying to receive a specific message (see simReceiveData) **data**: data to transmit **dataLength**: length of the data to transmit **antennaHandle**: handle of the scene object that should operate as the antenna for this transmission. Use sim_handle_default to simulate an antenna coinciding with the world reference frame **actionRadius**: radius of the transmission area. If the sender's antenna and receiver's antenna are farther apart than the actionRadius, the receiver can't receive the data **emissionAngle1**: opening angle of the transmission area, vertically (along the antenna's z-axis). Value can vary between 0 and pi. If pi, and emissionAngle2 is 2pi, then the transmission area is spherical. **emissionAngle2**: opening angle of the transmission area, horizontally (along the antenna's x/y-axis). Value can vary between 0 and 2pi. If 2pi, and emissionAngle1 is pi, then the transmission area is spherical. **persistence**: the simulation time duration after which the data is not available to receivers anymore, or 0.0 for a persistence of 1.5*simulationTimeStep (default)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.sendData(number targetID,number dataHeader,string dataName,string data,number antennaHandle=sim.handle_self,number actionRadius=100,number emissionAngle1=3.1415,number emissionAngle2=6.283,number persistence=0)
Lua parameters	**targetID**: indicates what receivers will receive the message. Can be sim.handle_all, sim.handle_tree, sim.handle_chain, or the handle of a script. If sim.handle_tree is specified, then only child scripts built on top of current script's hierarchy will be able to receive the message. If sim.handle_chain is specified, then only child scripts parented with current script (or the main script) will be able to receive the message **dataHeader**: number indicating who "designed" the communication message. Always use the same header (because only you will know the meaning of the message) and stick to it. The best is to use the serial number of your V-REP copy (check the "Help" menu, in the "About" item for the serial number). Otherwise, you risk collision with other developer's messages which might use the same header as yours. **dataName**: name indicating the type of message. dataHeader and dataName will be used to filter out all unwanted messages when trying to receive a specific message (see sim.receiveData) **data**: data to transmit (each character can have values between 0 and 255). See also the data packing/unpacking functions. **antennaHandle**: handle of the scene object that should operate as the antenna for this transmission. Use sim.handle_default to simulate an antenna coinciding with the world reference frame, or sim.handle_self to use the object associated with the child script as antenna. Can be omitted (in that case, sim.handle_self is used) **actionRadius**: same as C-function. Can be omitted (in that case 100 is used) **emissionAngle1**: same as C-function. Can be omitted (in that case, pi is used) **emissionAngle2**: same as C-function. Can be omitted (in that case, 2pi is used) **persistence**: same as C-function. Can be omitted (in that case, 0.0 is used)
Lua return values	Same as C-function

simSendModuleMessage

Description	Sends a message to plugins. This function should normally only be used from the main client application side. See also sim.loadModule and simBroadcastMessage.
C synopsis	simVoid* simSendModuleMessage(simInt message,simInt* auxiliaryData,simVoid* customData,simInt* replyData)
C parameters	**message**: the message to send. Refer to sim_message_eventcallback_-type messages. **auxiliaryData**: pointer to 4 integers. auxiliaryData[0] should be a unique identifier different from 0. Use a large random identifier, or better, your V-REP serial number if the message is yours. Otherwise, use the identifier of some other module. auxiliaryData[1] could be the messageID of the message you wish to send to another module. auxiliaryData[2] and auxiliaryData[3] can be any values specific to your application. **customData**: customData of your application (the broadcaster is in charge to release that buffer). Can be NULL. **replyData**: pointer to 4 integers that can be used by a module to reply to a broadcasted message. Can be NULL. If not NULL, all 4 values are automatically initialized to -1. Broadcasted messages can be intercepted in a plugin's "v_repMessage"-function.
C return value	Pointer to custom reply data that can be used by a module to reply to a broadcasted message. The module that replies is in charge of allocating the data with simCreateBuffer and the original broadcaster is in charge of releasing that data with simReleaseBuffer. A reply to a broadcasted message is triggered by a module that writes a value different from -1 into auxiliaryData[0]-auxiliaryData[3], thus aborting further broadcast of the original message and returning to the broadcaster. If the return value is different from NULL, the broadcast is also interrupted.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simSerialCheck / sim.serialCheck

Description	Reads how many bytes are waiting to be read on a serial port (RS-232). See also sim.serialRead.
C synopsis	simInt simSerialCheck(simInt portHandle)
C parameters	**portHandle**: the handle returned by the simSerialOpen function
C return value	-1 if operation was not successful, otherwise the number of bytes that are waiting to be read
Lua synopsis	number result=sim.serialCheck(number portHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSerialClose / sim.serialClose

Description	Closes a serial port (RS-232). See also sim.serialOpen.
C synopsis	simInt simSerialClose(simInt portHandle)
C parameters	**portHandle**: the handle returned by the simSerialOpen function
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.serialClose(number portHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSerialOpen / sim.serialOpen

Description	Opens a serial port (RS-232) for communication. When called from a script, the function can only be called when the simulation is running (and in that case the port is automatically closed at simulation stop). See also sim.serialClose, sim.serialSend, sim.serialCheck and sim.serialRead.
C synopsis	simInt simSerialOpen(simChar* portString,simInt baudRate,simVoid* reserved1,simVoid* reserved2)
C parameters	**portString**: a string specifying the port to open. Under Windows, use something similar to "\\.\COM1". Under MacOS and Linux, use something similar to "/dev/*" (check the "/dev" folder to know what file to specify). Under Linux, you might have to launch V-REP with super user priviledges in order to access

	the serial port. **baudRate**: the baudrate **reserved1**: reserved for future extension. Keep at NULL. **reserved2**: reserved for future extension. Keep at NULL.
C return value	-1 if operation was not successful, otherwise a port handle
Lua synopsis	number result=sim.serialOpen(string portString,number baudRate)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSerialRead / sim.serialRead

Description	Reads from a previously opened serial port (RS-232). The C version of the function cannot be blocking. See also sim.serialCheck and sim.serialSend.
C synopsis	simInt simSerialRead(simInt portHandle,simChar* buffer,simInt dataLengthToRead)
C parameters	**portHandle**: the handle returned by the simSerialOpen function **buffer**: a buffer that will be filled with read data **dataLengthToRead**: the maximum data length that should be read
C return value	-1 if operation was not successful, otherwise the effective data length that was read
Lua synopsis	string data=sim.serialRead(number portHandle,number dataLengthToRead,Boolean blockingOperation,string closingString='',number timeout=0)
Lua parameters	**portHandle**: the handle returned by the sim.serialOpen function **dataLengthToRead**: the maximum data length that should be read **blockingOperation**: if true and the calling script is running in a thread, then the function only returns when the desired data length was read (or if the closingString was met, or if there was a timeout (see next arguments) **closingString**: a string (containing any byte value) can be specified, that will break from the blocking operation if a match was found in the incoming data. Useful when you know that a data packet is always ended with a given signature. Can be an empty string for default operation. **timeout**: duration after which the blocking operation will be aborted, or 0 if the timeout is infinite.
Lua return values	**data**: a string containing read data (excluding the closingString if it was specified and found)

simSerialSend / sim.serialSend

Description	Writes data to a previously opened serial port (RS-232). See also sim.serialRead.
C synopsis	simInt simSerialSend(simInt portHandle,const simChar* data,simInt dataLength)
C parameters	**portHandle**: the handle returned by the simSerialOpen function **data**: a pointer to the data that should be sent **dataLength**: length of the data to be sent
C return value	-1 if operation was not successful, otherwise the effective data length that was written
Lua synopsis	number charsSent=sim.serialSend(number portHandle,string data)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetArrayParameter / sim.setArrayParameter (remote API equivalent: simxSetArrayParameter)

Description	Sets 3 values of an array parameter. See also sim.getArrayParameter, sim.setBoolParameter, sim.setInt32Parameter and sim.setFloatParameter.
C synopsis	simInt simSetArrayParameter(simInt parameter,const simVoid* parameterValues)
C parameters	**parameter**: array parameter identifier **parameterValues**: array of 3 simFloat values related to the parameter (simVoid is kept for backward compatibility).
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setArrayParameter(number parameter,table parameterValues)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetBoolParameter / sim.setBoolParameter (remote API equivalent: simxSetBooleanParameter)

Description	Sets a boolean parameter. See also sim.getBoolParameter, sim.setInt32Parameter, sim.setArrayParameter and sim.setFloatParameter.
C synopsis	simInt simSetBoolParameter(simInt parameter,simBool boolState)
C parameters	**parameter**: Boolean parameter identifier **boolState**: new boolean state for the parameter
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setBoolParameter(number parameter,boolean boolState)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetBooleanParameter / sim.setBoolParameter (DEPRECATED)

Description	DEPRECATED. See sim.setBoolParameter instead.

simSetCollectionName / sim.setCollectionName

Description	Sets the name of a collection based on its handle. See also sim.getCollectionName.
C synopsis	simInt simSetCollectionName(simInt collectionHandle,const simChar* collectionName)
C parameters	**collectionHandle**: handle of the collection **collectionName**: new name of the collection
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setCollectionName(number collectionHandle,string collectionName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetConfigurationTree / sim.setConfigurationTree

Description	Restores configuration information previously retrieved with sim.getConfigurationTree (object relative positions/orientations, joint/path values). Dynamically simulated objects will implicitely be reset before the command is applied (i.e. similar to calling sim.resetDynamicObject just before)
C synopsis	simInt simSetConfigurationTree(const simChar* data)
C parameters	**data**: data returned by a previous call to simGetConfigurationTree
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setConfigurationTree(number rawBufferHandle)
Lua parameters	**rawBufferHandle**: handle to a block of memory previously returned by sim.getConfigurationTree. If not needed anymore, you can release the raw buffer with the simReleaseScriptRawBuffer (all raw buffers are however automatically released at the end of a simulation)
Lua return values	Same as C-function

simSetEngineBoolParameter / sim.setEngineBoolParameter

Description	Sets a bool-type physics engine property. You might have to call sim.resetDynamicObject for changes to take effect. See also the other engine properties setter and getter API functions.

C synopsis	simInt simSetEngineBoolParameter(simInt paramId,simInt objectHandle,const simVoid* object,simBool val)
C parameters	**paramId**: the engine parameter identifier. **objectHandle**: the handle of the shape or joint, or -1 to set a global engine parameter. If -1, then the *object* argument will be evaluated. **object**: a pointer to a shape or joint objects, or NULL to set a global engine parameter. If NULL, then the *objectHandle* argument will be evaluated. **val**: the new property values.
C return value	1 in case of success. This function call doesn't generate any error message.
Lua synopsis	number result=sim.setEngineBoolParameter(number paramId,number objectHandle,boolean boolParam)
Lua parameters	Similar as C-function
Lua return values	**result**: 1 in case of success. This function call doesn't generate any error message.

simSetEngineFloatParameter / sim.setEngineFloatParameter

Description	Sets a float-type physics engine property. You might have to call sim.resetDynamicObject for changes to take effect. See also the other engine properties setter and getter API functions.
C synopsis	simInt simSetEngineFloatParameter(simInt paramId,simInt objectHandle,const simVoid* object,simFloat val)
C parameters	**paramId**: the engine parameter identifier. **objectHandle**: the handle of the shape or joint, or -1 to set a global engine parameter. If -1, then the *object* argument will be evaluated. **object**: a pointer to a shape or joint objects, or NULL to set a global engine parameter. If NULL, then the *objectHandle* argument will be evaluated. **val**: the new property values.
C return value	1 in case of success. This function call doesn't generate any error message.
Lua synopsis	number result=sim.setEngineFloatParameter(number paramId,number objectHandle,number floatParam)
Lua parameters	Similar as C-function
Lua return values	**result**: 1 in case of success. This function call doesn't generate any error message.

simSetEngineInt32Parameter / sim.setEngineInt32Parameter

Description	Sets an int32-type physics engine property. You might have to call sim.resetDynamicObject for changes to take effect. See also the other engine properties setter and getter API functions.
C synopsis	simInt simSetEngineInt32Parameter(simInt paramId,simInt objectHandle,const simVoid* object,simInt val)
C parameters	**paramId**: the engine parameter identifier. **objectHandle**: the handle of the shape or joint, or -1 to set a global engine parameter. If -1, then the *object* argument will be evaluated. **object**: a pointer to a shape or joint objects, or NULL to set a global engine parameter. If NULL, then the *objectHandle* argument will be evaluated. **val**: the new property values.
C return value	1 in case of success. This function call doesn't generate any error message.
Lua synopsis	number result=sim.setEngineInt32Parameter(number paramId,number objectHandle,number int32Param)
Lua parameters	Similar as C-function
Lua return values	**result**: 1 in case of success. This function call doesn't generate any error message.

simSetExplicitHandling / sim.setExplicitHandling

Description	Sets the explicit handling flags for a general object. See also sim.getExplicitHandling.
C synopsis	simInt simSetExplicitHandling(simInt generalObjectHandle,int explicitFlags)
C parameters	**generalObjectHandle**: handle of a general object (can be a scene object, a collision object, a distance object, etc.) **explicitFlags**: the explicit handling flags. For now only bit 0 is used

C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setExplicitHandling(number generalObjectHandle,number explicitFlags)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetFloatParameter / sim.setFloatParameter (remote API equivalent: simxSetFloatingParameter)

Description	Sets a floating point parameter. See also sim.getFloatParameter, sim.setBoolParameter, sim.setArrayParameter and sim.setInt32Parameter.
C synopsis	simInt simSetFloatParameter(simInt parameter,simFloat floatState)
C parameters	**parameter**: floating parameter identifier **floatState**: new state for the parameter
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setFloatParameter(number parameter,number floatState)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetFloatSignal / sim.setFloatSignal (remote API equivalent: simxSetFloatSignal)

Description	Sets the value of a float signal. If that signal is not yet present, it is added. Signals created in the main script or in a child script are automatically cleared at simulation end. See also sim.getFloatSignal, the other signal functions, and sim.persistentDataWrite.
C synopsis	simInt simSetFloatSignal(const simChar* signalName,simFloat signalValue)
C parameters	**signalName**: name of the signal **signalValue**: value of the signal
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setFloatSignal(string signalName,number signalValue)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetFloatingParameter / sim.setFloatParameter (DEPRECATED)

Description	DEPRECATED. See sim.setFloatParameter instead.

simSetGraphUserData / sim.setGraphUserData

Description	Sets one value in a user-defined graph data stream. See also sim.resetGraph and sim.handleGraph.
C synopsis	simInt simSetGraphUserData(simInt graphHandle,const simChar* dataStreamName,simFloat data)
C parameters	**graphHandle**: handle of the graph object **dataStreamName**: the name of the data stream. The data stream must be of type "user-defined" **data**: the value to set. If, for a given simulation step this function is not called for a user-defined data stream, then the data will be missing for that simulation step.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setGraphUserData(number graphHandle,string dataStreamName,number data)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetIkElementProperties / sim.setIkElementProperties

Description	Sets properties of a specific inverse kinematics element (IK element). See also sim.setIkGroupProperties and sim.getIkGroupHandle.
C synopsis	simInt simSetIkElementProperties(simInt ikGroupHandle,simInt tipDummyHandle,simInt constraints,const simFloat* precision,const simFloat* weight,void* reserved)
C parameters	**ikGroupHandle**: handle of the IK group that contains the IK element to modify **tipDummyHandle**: handle of the tip dummy object of the IK element **constraints**: the constraints of the ik element. sim_ik_avoidance_constraint is not allowed **precision**: an array of two values where the first represents the linear precision, and the second the angular precision. Can be NULL to keep current settings. **weight**: an array of two values that represent the linear and angular resolution weights. Can be NULL to keep current settings. **reserved**: reserved for future extensions. Keep at NULL
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setIkElementProperties(number ikGroupHandle,number tipDummyHandle,number constraints,table_2 precision=nil,table_2 weight=nil
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetIkGroupProperties / sim.setIkGroupProperties

Description	Sets properties of an inverse kinematics group (IK group). See also sim.setIkElementProperties and sim.getIkGroupHandle.
C synopsis	simInt simSetIkGroupProperties(simInt ikGroupHandle,simInt resolutionMethod,simInt maxIterations,simFloat damping,void* reserved)
C parameters	**ikGroupHandle**: handle of the IK group **resolutionMethod**: the IK resolution method **maxIterations**: the maximum number of iteractions for the calculations **damping**: the DLS damping factor. **reserved**: reserved for future extensions. Keep at NULL
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setIkGroupProperties(number ikGroupHandle,number resolutionMethod,number maxIterations,number damping)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetInt32Parameter / sim.setInt32Parameter (remote API equivalent: simxSetIntegerParameter)

Description	Sets an integer parameter. See also sim.getInt32Parameter, sim.setBoolParameter, sim.setArrayParameter and sim.setFloatParameter.
C synopsis	simInt simSetInt32Parameter(simInt parameter,simInt intState)
C parameters	**parameter**: integer parameter identifier **intState**: new state for the parameter
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setInt32Parameter(number parameter,number intState)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetIntegerParameter / sim.setInt32Parameter (DEPRECATED)

Description	DEPRECATED. See sim.setInt32Parameter instead.

simSetIntegerSignal / sim.setIntegerSignal (remote API equivalent: simxSetIntegerSignal)

Description	Sets the value of an integer signal. If that signal is not yet present, it is added. Signals created in the main script or in a child script are automatically cleared at simulation end. See also sim.getIntegerSignal, the other signal functions, and sim.persistentDataWrite.
C synopsis	simInt simSetIntegerSignal(const simChar* signalName,simInt signalValue)
C parameters	**signalName**: name of the signal **signalValue**: value of the signal
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setIntegerSignal(string signalName,number signalValue)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetJointForce / sim.setJointForce (remote API equivalent: simxSetJointForce)

Description	Sets the maximum force or torque that a joint can exert. This function has no effect when the joint is not dynamically enabled, or when it is a spherical joint. See also sim.getJointForce.
C synopsis	simInt simSetJointForce(simInt objectHandle,simFloat forceOrTorque)
C parameters	**objectHandle**: handle of the joint object **forceOrTorque**: the maximum force or torque that the joint can exert
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setJointForce(number objectHandle,number forceOrTorque)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetJointInterval / sim.setJointInterval

Description	Sets the interval parameters of a joint (i.e. range values). The attributes or interval parameters might have no effect, depending on the joint-type. See also sim.getJointInterval.
C synopsis	simInt simSetJointInterval(simInt objectHandle,simBool cyclic,const simFloat* interval)
C parameters	**objectHandle**: handle of the joint object **cyclic**: indicates whether the joint is cyclic. Only revolute joints with a pitch of 0 can be cyclic **interval**: interval of the joint. interval[0] is the joint minimum allowed value, interval[1] is the joint range (i.e. the maximum allowed value is interval[0]+interval[1])
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setJointInterval(number objectHandle,boolean cyclic,table_2 interval)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetJointMode / sim.setJointMode

Description	Sets the operation mode of a joint. Might have as side-effect the change of additional properties of the joint. See also sim.getJointMode.
C synopsis	simInt simSetJointMode(simInt jointHandle,simInt jointMode,simInt options)
C parameters	**jointHandle**: handle of the joint object **jointMode**: a joint mode value **options**: bit-coded. For now only bit 0 is used (if set (1), the joint operates in hybrid mode)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be

	available
Lua synopsis	number result=sim.setJointMode(number jointHandle,number jointMode,number options)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetJointPosition / sim.setJointPosition (remote API equivalent: simxSetJointPosition)

Description	Sets the intrinsic position of a joint. May have no effect depending on the joint mode. This function cannot be used with spherical joints (use sim.setSphericalJointMatrix instead). See also sim.getJointPosition and sim.setJointTargetPosition.
C synopsis	simInt simSetJointPosition(simInt objectHandle,simFloat position)
C parameters	**objectHandle**: handle of the joint object **position**: position of the joint (angular or linear value depending on the joint type)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setJointPosition(number objectHandle,number position)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetJointTargetPosition / sim.setJointTargetPosition (remote API equivalent: simxSetJointTargetPosition)

Description	Sets the target position of a joint if the joint is in torque/force mode (also make sure that the joint's motor and position control are enabled). See also sim.getJointTargetPosition and sim.setJointPosition.
C synopsis	simInt simSetJointTargetPosition(simInt objectHandle,simFloat targetPosition)
C parameters	**objectHandle**: handle of the joint object **targetPosition**: target position of the joint (angular or linear value depending on the joint type)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setJointTargetPosition(number objectHandle,number targetPosition)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetJointTargetVelocity / sim.setJointTargetVelocity (remote API equivalent: simxSetJointTargetVelocity)

Description	Sets the intrinsic target velocity of a non-spherical joint. This command makes only sense when the joint mode is torque/force mode: the dynamics functionality and the joint motor have to be enabled (position control should however be disabled). See also sim.getJointTargetVelocity.
C synopsis	simInt simSetJointTargetVelocity(simInt objectHandle,simFloat targetVelocity)
C parameters	**objectHandle**: handle of the joint object **targetVelocity**: target velocity of the joint (linear or angular velocity depending on the joint-type).
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setJointTargetVelocity(number objectHandle,number targetVelocity)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetLastError

Description	Sets a custom error or warning message. This function is useful for plugins which wish to generate custom error/warning messages that will be displayed in the calling script. Errors/warnings are set and memorized on a thread-basis (e.g. threads originating from threaded scripts have each an individual error handler). See also sim.getLastError, the sim.intparam_error_report_mode and the error report

	modes.
C synopsis	simInt simSetLastError(const simChar* funcName,const simChar* errorMessage)
C parameters	**funcName**: name of the function where the error originated. If funcName starts with "warning@", then a warning messages will be registered instead. **errorMessage**: error or warning message
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	-
Lua parameters	-
Lua return values	-

simSetLightParameters / sim.setLightParameters

Description	Sets various parameters of a light object. See also sim.getLightParameters.
C synopsis	simInt simSetLightParameters(simInt objectHandle,simInt state,const simFloat* setToNULL,const simFloat* diffusePart,const simFloat* specularPart)
C parameters	**objectHandle**: handle of the light **state**: bit-coded. for now, only bit 0 is used: 1=light on **setToNULL**: not used, set to NULL **diffusePart**: red, green and blue component of the light's diffuse part. Can be NULL **specularPart**: red, green and blue component of the light's specular part. Can be NULL
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setLightParameters(number objectHandle,number state,nil,table_3 diffusePart,table_3 specularPart)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetLinkDummy / sim.setLinkDummy

Description	Defines (or breaks) a dummy-dummy link pair. Useful to create dynamic loop closure constraints on the fly (among others). See also sim.getLinkDummy.
C synopsis	simInt simSetLinkDummy(simInt dummyHandle,simInt linkedDummyHandle)
C parameters	**dummyHandle**: handle of the first dummy in the dummy-dummy link pair. **linkedDummyHandle**: handle of the second dummy in the dummy-dummy link pair. Set to -1 to unlink the first dummy.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.getLinkDummy(number dummyHandle,number linkedDummyHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetModelProperty / sim.setModelProperty (remote API equivalent: simxSetModelProperty)

Description	Sets the properties of a model. See also sim.getModelProperty, sim.setObjectProperty and sim.setObjectSpecialProperty.
C synopsis	simInt simSetModelProperty(simInt objectHandle,simInt prop)
C parameters	**objectHandle**: handle of the object that serves as the model base **prop**: model property. See the model property values. Combine them with the "or"-operator
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setModelProperty(number objectHandle,number prop)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetModuleInfo

Description	Attaches additional information to a loaded plugin. See also sim.getModuleInfo.
C synopsis	simInt simSetModuleInfo(const simChar* moduleName,simInt infoType,const simChar* stringInfo,simInt intInfo)
C parameters	**moduleName**: the name of the plugin. **infoType**: the type of information to set: 0: *extended version string* 1: *build date string* 2: *extended version integer* **stringInfo**: a string information value, in case the information type is for a string. **intInfo**: an integer information value, in case the information type is for an integer.
C return value	-1 in case of an error
Lua synopsis	-
Lua parameters	-
Lua return values	-

simSetModuleMenuItemState

Description	Updates the state of a menu item added with the simAddModuleMenuEntry command.
C synopsis	simInt simSetModuleMenuItemState(simInt itemHandle,simInt state,const simChar* label)
C parameters	**itemHandle**: handle of the item as returned by the simAddModuleMenuEntry function **state**: state of the item. Bit 0 (1)indicates the enabled state, bit 1 (2)indicates the checked state **label**: label of the item. Can be NULL in which case the label is kept unchanged. If label is "", the item becomes a separator
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	-
Lua parameters	-
Lua return values	-

simSetNameSuffix / sim.setNameSuffix

Description	Sets the name suffix adjustment number (for detailed information on this, read also the section on accessing general-type objects). In V-REP, all objects are identified by a name and a handle. When an object (scene object or general-type object) is copied at the same time as a child script, the newly created object's name will become "oldName#0", should the same object be pasted another time, the next name will be "oldName#1", etc. From within a child script, retrieving object handles is performed by automatically appending a name suffix to the object name (each script gets initialized with the name suffix number of the object it is attached to). This allows to copy-paste objects and scripts without having to manually adjust the scripts (the scripts will automatically know which object they have to access based on the set name suffix). From within a script, most of the time you won't need to set the name suffix, but in some special cases you might want to temporarily disable it (e.g. "myChildScript#42" (which has its name suffix automatically set to 42) copied itself together with its attached robot ("myRobot#42") and now from within "myChildScript#42" you want to shift "myRobot#43" to avoid collision. In that case set the name suffix to 43, shift "myRobot" (retrieve its handle with sim.getObjectHandle("myRobot") then set the suffix back to 42). From within a script, the sim.setNameSuffix command is influencing only current script. When accessing the API from outside of a script however, the name adjustment mechanism needs to be adjusted manually (make sure you reset the name suffix to its initial state after you are done retrieving handles). Imagine you have one robot in your scene that is named "robot". You can access the robot from a C/C++ application with sim.getObjectHandle("robot"). If the robot is duplicated, the second robot's name will be "robot#0", the third will be "robot#1", etc. From within you C/C++ application you can now access all robots with the same code, you just need to adjust the name suffix number. E.g. sim.setNameSuffix(42), then sim.getObjectHandle("robot") will retrieve the handle of

	"robot#42". Once you are done accessing objects, reset the name suffix number to -1 (sim.setNameSuffix(-1)). Setting the name suffix to -1 disables the name adjustment mechanism (default when accessing the API from outside of a script) See also the sim.getNameSuffix function.
C synopsis	simInt simSetNameSuffix(simInt nameSuffixNumber)
C parameters	**nameSuffixNumber**: a number starting from -1 (-1 is for no suffix, 0 is for the 0 suffix, etc.)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result sim.setNameSuffix(number nameSuffixNumber)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetNavigationMode / sim.setNavigationMode

Description	Sets the navigation and selection mode for the mouse. See also sim.getNavigationMode.
C synopsis	simInt simSetNavigationMode(simInt navigationMode)
C parameters	**navigationMode**: mouse navigation mode
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setNavigationMode(number navigationMode)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetObjectConfiguration / sim.setObjectConfiguration

Description	Sets configuration information for an object (object relative position/orientation, joint/path value). Dynamically simulated objects will implicitely be reset before the command is applied (i.e. similar to calling sim.resetDynamicObject just before). See also sim.getObjectConfiguration and sim.setConfigurationTree.
C synopsis	simInt simSetObjectConfiguration(const simChar* data)
C parameters	**data**: data returned by a previous call to simGetObjectConfiguration
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setObjectConfiguration(number rawBufferHandle)
Lua parameters	**rawBufferHandle**: handle to a raw data buffer (value returned by a previous call to sim.getObjectConfiguration). If not needed anymore, you can release the raw buffer with the simReleaseScriptRawBuffer (all raw buffers are however automatically released at the end of a simulation)
Lua return values	Same as C-function

simSetObjectFloatParameter / sim.setObjectFloatParameter (remote API equivalent: simxSetObjectFloatParameter)

Description	Sets a floating-point parameter of a scene object or calculation object. See also sim.getObjectFloatParameter, sim.setObjectInt32Parameter and sim.setObjectStringParameter
C synopsis	simInt simSetObjectFloatParameter(simInt objectHandle,simInt parameterID,simFloat parameter)
C parameters	**objectHandle**: handle of the object **parameterID**: identifier of the parameter to retrieve. See the list of all possible object parameter identifiers **parameter**: parameter value
C return value	-1 in case of an error, 0 if the parameter could not be set (e.g. because the parameterID doesn't exist, or because the specified object doesn't correspond to the correct type), or 1 if the operation was successful

Lua synopsis	number result=sim.setObjectFloatParameter(number objectHandle,number parameterID,number parameter)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetObjectInt32Parameter / sim.setObjectInt32Parameter (remote API equivalent: simxSetObjectIntParameter)

Description	Sets an int32 parameter of a scene object or calculation object. See also sim.getObjectInt32Parameter, sim.setObjectFloatParameter and sim.setObjectStringParameter
C synopsis	simInt simSetObjectInt32Parameter(simInt objectHandle,simInt parameterID,simInt parameter)
C parameters	**objectHandle**: handle of the object **parameterID**: identifier of the parameter to retrieve. See the list of all possible object parameter identifiers **parameter**: parameter value
C return value	-1 in case of an error, 0 if the parameter could not be set (e.g. because the parameterID doesn't exist, or because the specified object doesn't correspond to the correct type), or 1 if the operation was successful
Lua synopsis	number result=sim.setObjectInt32Parameter(number objectHandle,number parameterID,number parameter)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetObjectIntParameter / sim.setObjectInt32Parameter (DEPRECATED)

Description	DEPRECATED. See sim.setObjectInt32Parameter instead.

simSetObjectMatrix / sim.setObjectMatrix

Description	Sets the transformation matrix of an object. Dynamically simulated objects will implicitely be reset before the command is applied (i.e. similar to calling sim.resetDynamicObject just before). See also sim.getObjectMatrix, sim.setObjectPosition, sim.setObjectOrientation and the other matrix/transformation functions.
C synopsis	simInt simSetObjectMatrix(simInt objectHandle,simInt relativeToObjectHandle,const simFloat* matrix)
C parameters	**objectHandle**: handle of the object **relativeToObjectHandle**: indicates relative to which reference frame the matrix is specified. Specify -1 to set the absolute transformation matrix, sim_handle_parent to set the transformation matrix relative to the object's parent, or an object handle relative to whose reference frame the transformation matrix is specified. **matrix**: pointer to 12 simFloat values (the last row of the 4x4 matrix (0,0,0,1) is not needed) The x-axis of the orientation component is (matrix[0],matrix[4],matrix[8]) The y-axis of the orientation component is (matrix[1],matrix[5],matrix[9]) The z-axis of the orientation component is (matrix[2],matrix[6],matrix[10]) The translation component is (matrix[3],matrix[7],matrix[11])
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setObjectMatrix(number objectHandle,number relativeToObjectHandle,table_12 matrix)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetObjectName / sim.setObjectName

Description	Sets the name of an object based on its handle. See also sim.getObjectName.

C synopsis	simInt simSetObjectName(simInt objectHandle,const simChar* objectName)
C parameters	**objectHandle**: handle of the object. By adding *sim.handleflag_altname* to the object handle, the object alternative name can be set. By adding *sim.handleflag_silenterror* to the object handle, errors linked to the naming can be suppressed from output. **objectName**: name (or alternative name) of the object
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setObjectName(number objectHandle,string objectName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetObjectOrientation / sim.setObjectOrientation (remote API equivalent: simxSetObjectOrientation)

Description	Sets the orientation (Euler angles) of an object. Dynamically simulated objects will implicitely be reset before the command is applied (i.e. similar to calling sim.resetDynamicObject just before). See also sim.setObjectQuaternion, sim.getObjectOrientation, sim.setObjectPosition, sim.setObjectMatrix and the other matrix/transformation functions.
C synopsis	simInt simSetObjectOrientation(simInt objectHandle,simInt relativeToObjectHandle,const simFloat* eulerAngles)
C parameters	**objectHandle**: handle of the object **relativeToObjectHandle**: indicates relative to which reference frame the orientation is specified. Specify -1 to set the absolute orientation, sim_handle_parent to set the orientation relative to the object's parent, or an object handle relative to whose reference frame the orientation is specified. **eulerAngles**: Euler angles (alpha, beta and gamma)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setObjectOrientation(number objectHandle,number relativeToObjectHandle,table_3 eulerAngles)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetObjectParent / sim.setObjectParent (remote API equivalent: simxSetObjectParent)

Description	Sets an object's parent object. See also sim.getObjectParent.
C synopsis	simInt simSetObjectParent(simInt objectHandle,simInt parentObjectHandle,simBool keepInPlace)
C parameters	**objectHandle**: handle of the object that will become child of the parent object. Can be combined with sim_handleflag_assembly (simply add sim_handleflag_assembly to **objectHandle**), if the two objects can be assembled via a predefined assembly transformation (refer to the **assembling** option in the object common properties). In that case, **parentObjectHandle** can't be -1, and **keepInPlace** should be set to false. **parentObjectHandle**: handle of the object that will become parent, or -1 if the object should become parentless. **keepInPlace**: indicates whether the object's absolute position and orientation should stay same
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setObjectParent(number objectHandle,number parentObjectHandle,boolean keepInPlace)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetObjectPosition / sim.setObjectPosition (remote API equivalent: simxSetObjectPosition)

Description	Sets the position (x, y and z-coordinates) of an object. Dynamically simulated objects will implicitely be reset before the command is applied (i.e. similar to calling sim.resetDynamicObject just before). See also sim.getObjectPosition, sim.setObjectOrientation, sim.setObjectMatrix and the other matrix/transformation functions.
C synopsis	simInt simSetObjectPosition(simInt objectHandle,simInt relativeToObjectHandle,const simFloat*

	position)
C parameters	**objectHandle**: handle of the object **relativeToObjectHandle**: indicates relative to which reference frame the position is specified. Specify -1 to set the absolute position, sim_handle_parent to set the position relative to the object's parent, or an object handle relative to whose reference frame the position is specified. **position**: coordinates of the object (x, y and z)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setObjectPosition(number objectHandle,number relativeToObjectHandle,table_3 position)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetObjectProperty / sim.setObjectProperty

Description	Sets the properties of a scene object. See also sim.getObjectProperty, sim.setObjectSpecialProperty and sim.setModelProperty.
C synopsis	simInt simSetObjectProperty(simInt objectHandle,simInt prop)
C parameters	**objectHandle**: object handle **prop**: object property. See the object property values. Combine them with the "or"-operator
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setObjectProperty(number objectHandle,number prop)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetObjectQuaternion / sim.setObjectQuaternion

Description	Sets the quaternion (x,y,z,w) of an object. Be very careful to set only valid value (i.e. normalized), otherwise you will experience strange effects. Dynamically simulated objects will implicitely be reset before the command is applied (i.e. similar to calling sim.resetDynamicObject just before). See also sim.setObjectOrientation and the other matrix/transformation functions.
C synopsis	simInt simSetObjectQuaternion(simInt objectHandle,simInt relativeToObjectHandle,const simFloat* quaternion)
C parameters	**objectHandle**: handle of the object **relativeToObjectHandle**: indicates relative to which reference frame the orientation is specified. Specify -1 to set the absolute orientation, sim_handle_parent to set the orientation relative to the object's parent, or an object handle relative to whose reference frame the orientation is specified. **quaternion**: the quaternion (x,y,z,w)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setObjectQuaternion(number objectHandle,number relativeToObjectHandle,table_4 quaternion)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetObjectSizeValues / sim.setObjectSizeValues

Description	Sets the x, y and z size values of a scene object. The size values are different from the real object sizes. Use this to be able to react to scaling operations. See also sim.getObjectSizeValues.
C synopsis	simInt simSetObjectSizeValues(simInt objectHandle,const simFloat* sizeValues)
C parameters	**objectHandle**: handle of the scene object **sizeValues (input)**: a pointer to 3 size values (x, y and z)
C return value	-1 in case of an error
Lua synopsis	number result=sim.setObjectSizeValues(number objectHandle,table_3 sizeValues)
Lua parameters	Same as C-function

Lua return values	Same as C-function

simSetObjectSpecialProperty / sim.setObjectSpecialProperty

Description	Sets the special properties of a scene object. See also sim.getObjectSpecialProperty, sim.setObjectProperty and sim.setModelProperty.
C synopsis	simInt simSetObjectSpecialProperty(simInt objectHandle,simInt prop)
C parameters	**objectHandle**: object handle **prop**: object special property. See the object special property values. Combine them with the "or"-operator
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setObjectSpecialProperty(number objectHandle,number prop)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetObjectStringParameter / sim.setObjectStringParameter

Description	Sets a string parameter of a scene object or calculation object. See also sim.getObjectStringParameter, sim.setObjectInt32Parameter and sim.setObjectFloatParameter
C synopsis	simInt simSetObjectStringParameter(simInt objectHandle,simInt parameterID,simChar* parameter,simInt parameterLength)
C parameters	**objectHandle**: handle of the object **parameterID**: identifier of the parameter to retrieve. See the list of all possible object parameter identifiers **parameter**: parameter value (can contain embedded zeros) **parameterLength**: the length of the parameter value
C return value	-1 in case of an error, 0 if the parameter could not be set (e.g. because the parameterID doesn't exist, or because the specified object doesn't correspond to the correct type), or 1 if the operation was successful
Lua synopsis	number result=sim.setObjectStringParameter(number objectHandle,number parameterID,string parameter)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetPage / sim.setPage

Description	Switches between pages (main scene views). See also sim.getPage.
C synopsis	simInt simSetPage(simInt index)
C parameters	**index**: index of the page. Valid values are 0-7
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setPage(number index)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetPathPosition / sim.setPathPosition

Description	Sets the intrinsic position of a path object (i.e. the position along the path). The position is given in meters, but the actual position is dependent on the selected path length calculation method for the given path object.This function is the equivalent of sim.setJointPosition, but for a path object. See also sim.getPathPosition.
C synopsis	simInt simSetPathPosition(simInt objectHandle,simFloat position)

C parameters	**objectHandle**: handle of the path object **position**: linear position on the path given in meters (but dependent on the selected path length calculation method)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setPathPosition(number objectHandle,number position)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetPathTargetNominalVelocity / sim.setPathTargetNominalVelocity (DEPRECATED)

Description	DEPRECATED. Instead of using this function, handle the path intrinsic position update inside of a child script.

simSetPointCloudOptions / sim.setPointCloudOptions

Description	Sets various properties of a point cloud. See also sim.getPointCloudOptions and the other point cloud related functions.
C synopsis	simInt simSetPointCloudOptions(simInt pointCloudHandle,simFloat maxVoxelSize,simInt maxPtCntPerVoxel,simInt options,simFloat pointSize,simVoid* reserved)
C parameters	**pointCloudHandle**: the handle of the point cloud. See also simGetObjectHandle **maxVoxelSize**: the maximum size of the octree voxels containing points **maxPtCntPerVoxel**: the maximum number of points allowed in a same octree voxel **options**: bit-coded: bit0 set (1): points have random colors bit1 set (2): show octree structure bit2 set (4): reserved. keep unset bit3 set (8): do not use an octree structure. When enabled, point cloud operations are limited, and point clouds will not be collidable, measurable or detectable anymore, but adding points will be much faster bit4 set (16): color is emissive **pointSize**: the size of the points, in pixels **reserved**: reserved for future extensions. Set to NULL
C return value	1 if operation was successful
Lua synopsis	number result=sim.setPointCloudOptions(number pointCloudHandle,number maxVoxelSize,number maxPtCntPerVoxel,number options,number pointSize)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetReferencedHandles / sim.setReferencedHandles

Description	Attaches a list of custom handles to a given scene object. Those custom handles are handles of other objects, that are linked to the given scene object (for whatever purpose). The advantage of storing references to other objects with this function is that V-REP will take care of correctly adjusting the references if needed: For instance, imaging *objectA* storing the handle of *objectB* via this function. If *objectB* is deleted, then the stored handle will become -1. If *objectA* and *objectB* are duplicated at the same time, then the duplicate of *objectA* will store the handle of the duplicate of *objectB*. See also sim.getReferencedHandles.
C synopsis	simInt simSetReferencedHandles(simInt objectHandle,simInt count,const simInt* referencedHandles,const simInt* reserved1,const simInt* reserved2)
C parameters	**objectHandle**: handle of the scene object that will store the list of handles **count**: the number of handles to store **referencedHandles**: a list of handles. Handles of following object types are supported: scene objects, collision objects, distance objects, IK groups, collections, and geometric constraint solver objects. **reserved1**: reserved for future extensions **reserved2**: reserved for future extensions
C return value	-1 in case of an error.
Lua synopsis	number result=sim.setReferencedHandles(number objectHandle,table referencedHandles)

Lua parameters	Similar to the C-function
Lua return values	Similar to the C-function

simSetScriptAttribute / sim.setScriptAttribute

Description	Sets various script attributes or properties. See also sim.getScriptAttribute.
C synopsis	simInt simSetScriptAttribute(simInt scriptHandle,simInt attributeID,simFloat floatVal,simInt intOrBoolVal)
C parameters	**scriptHandle**: handle of a script **attributeID**: the script attributeID **floatVal**: the floating point attribute (if applicable) **intOrBoolVal**: the integer or Boolean attribute (if applicable)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setScriptAttribute(number scriptHandle,number attributeID,number/boolean attribute)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetScriptRawBuffer

Description	Attaches a raw buffer to a given script. All raw buffers are automatically released at the end of a simulation. See also simGetScriptRawBuffer.
C synopsis	simInt simSetScriptRawBuffer(simInt scriptHandle,const simChar* buffer,simInt bufferSize)
C parameters	**scriptHandle**: handle of the script **buffer**: pointer to the buffer **bufferSize**: size of the buffer (the function will copy the buffer which can then immediately be released by the calling application)
C return value	a handle to the buffer. Using that handle and the simGetScriptRawBuffer, the buffer can be retrieved. In case of an error, the return value is -1
Lua synopsis	-
Lua parameters	-
Lua return values	-

simSetScriptSimulationParameter / sim.setScriptSimulationParameter

Description	Sets a main script's or child script's parameter in its simulation parameter list. Useful for simple interaction with the user or for simple parameter exchange with other scripts. See also sim.getScriptSimulationParameter and the data packing/unpacking functions.
C synopsis	simInt simSetScriptSimulationParameter(simInt scriptHandle,const simChar* parameterName,const simChar* parameterValue,simInt parameterLength)
C parameters	**scriptHandle**: handle of the script, or sim_handle_main_script or sim_handle_all **parameterName**: name of the parameter to set **parameterValue**: value of the parameter (all parameters are treated as strings, but can be converted to number later on. Strings may contain any values (also embedded zeros)) **parameterLength**: number of bytes that parameterValue contains. If parameterValue is a regular string (without embedded zeros), then this is strlen(parameterValue).
C return value	number of parameters that were set (can be >1 if sim_handle_all was specified) or -1 if the parameterName could not be found or in case of an error
Lua synopsis	number setCount=sim.setScriptSimulationParameter(number scriptHandle,string parameterName,string/number parameterValue)
Lua parameters	**scriptHandle**: handle of the script, or sim.handle_main_script, sim.handle_all, sim.handle_tree, sim.handle_chain or sim.handle_self **parameterName**: Same as C-function **parameterValue**: Same as C-function
Lua return values	Same as C-function

simSetScriptText / sim.setScriptText

Description	Sets a new content for a script (i.e. attaches a new Lua code). During a simulation, the new script content might not be taken into consideration if a previous code was already executed at least once. Use with care when simulation is running. See also simGetScriptText, sim.addScript and sim.associateScriptWithObject.
C synopsis	simInt simSetScriptText(simInt scriptHandle,const simChar* scriptText)
C parameters	**scriptHandle**: handle of a script **scriptText**: pointer to a script buffer (0-terminated buffer). This function will copy the buffer content, so that it can immediately be released after this call
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setScriptText(number scriptHandle,string scriptText)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetScriptVariable / sim.setScriptVariable

Description	Sets or clears a script variable (i.e. a Lua variable). Call this only: a) from the main thread, or: b) from a thread that originated from a threaded child script. In that case, you cannot set a variable in a non-threaded child script. See also sim.callScriptFunction.
C synopsis	simInt simSetScriptVariable(simInt scriptHandleOrType,const simChar* variableNameAtScriptName,simInt stackHandle)
C parameters	**scriptHandleOrType**: the handle of the script, otherwise the type of the script: *sim_scripttype_mainscript* (0): the main script is the target. *sim_scripttype_childscript* (1): a child script is the target. In that case, *arrayNameAtScriptName* should also contain the name of the object associated with the script. *sim_scripttype_customizationscript* (6): a customization script is the target. In that case, *arrayNameAtScriptName* should also contain the name of the object associated with the script. **variableNameAtScriptName**: the name of the variable. If *scriptHandleOrType* is *sim_scripttype_childscript*, or *sim_scripttype_customizationscript*, then *variableNameAtScriptName* should also contain the name of the object associated with the script: "variableName@scriptName". **stackHandle**: the handle of a stack object. The top stack item represents the variable value. If the handle is 0, then the variable will be assigned the value *nil*. See also the available stack functions.
C return value	-1 in case of an error
Lua synopsis	number result=sim.setScriptVariable(string variableNameAtScriptName,number scriptHandleOrType,variable)
Lua parameters	Similar to the C-function, with the difference that a stack object is not required, and the desired variable value can directly be appended to the first two arguments.
Lua return values	Same as C-function

simSetShapeColor / sim.setShapeColor

Description	Sets the color (or transforms it) of one or several shapes. See also sim.getShapeColor.
C synopsis	simInt simSetShapeColor(simInt shapeHandle,simChar* colorName,simInt colorComponent,simFloat* rgbData)
C parameters	**shapeHandle**: handle of the shape, or sim_handle_all if the command should be directed at all shapes **colorName**: name of a color. Can be NULL, but if a name is provided, only shapes (or sub-entities of them) with a same color name will be modified. By specifying special names, the color can directly be transformed in the Hue-Saturation-Lightness (HSL) space: "@0": all specified shape outside colors will be transformed such as: HSL= {H+rgbData[0],S+rgbData[1],L+rgbData[2]}, where H operates in a cyclic manner. "@1": all specified shape inside colors will be transformed such as: HSL=

	{H+rgbData[0],S+rgbData[1],L+rgbData[2]}, where H operates in a cyclic manner. "@2": all specified shape edge colors will be transformed such as: HSL= {H+rgbData[0],S+rgbData[1],L+rgbData[2]}, where H operates in a cyclic manner. **colorComponent**: a color component **rgbData**: red, green and blue components of the color (3 values), or the transparency value (1 value)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setShapeColor(number shapeHandle,string colorName,number colorComponent,table_3 rgbData)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetShapeMassAndInertia / sim.setShapeMassAndInertia

Description	Applies mass and inertia information to a shape. If simulation is running, the shape will be dynamically reset (similar to calling sim.resetDynamicObject right after). See also sim.getShapeMassAndInertia, sim.getObjectMatrix and sim.buildMatrix.
C synopsis	simInt simSetShapeMassAndInertia(simInt shapeHandle,simFloat mass,const simFloat* inertiaMatrix,const simFloat* centerOfMass,const simFloat* transformation)
C parameters	**shapeHandle**: handle of the shape object **mass**: the new mass of the object **inertia matrix**: the new inertia matrix or tensor (9 values), expressed relative to the center of mass. The matrix should be relative to the orientational frame of **transformation** (see further below). **centerOfMass**: the new position of the center of mass, relative to the specified **transformation** (see next item). **transformation**: the transformation matrix (12 values) relative to which we specify the data. Can be NULL, in which case the data is relative to the absolute reference frame. See here to see how matrix transformations are specified in V-REP.
C return value	-1 in case of an error
Lua synopsis	number result=sim.setShapeMassAndInertia(number shapeHandle,number mass,table_9 inertiaMatrix,table_3 centerOfMass,table_12 transformation=nil)
Lua parameters	See the C-function for details
Lua return values	See the C-function for details

simSetShapeMaterial / sim.setShapeMaterial

Description	Sets the material (used by the physics engines) of a specific shape. You might have to also call sim.resetDynamicObject for changes to take effect.
C synopsis	simInt simSetShapeMaterial(simInt shapeHandle,simInt materialIdOrShapeHandle)
C parameters	**shapeHandle**: the handle of the shape. See also simGetObjectHandle. **materialIdOrShapeHandle**: a predefined dynamic material ID, or the handle of another shape you wish to copy the material properties from.
C return value	-1 in case of an error.
Lua synopsis	number result=sim.setShapeMaterial(number shapeHandle,number materialIdOrShapeHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetShapeTexture / sim.setShapeTexture

Description	Applies (or removes) a texture to a shape. See also sim.getTextureId, sim.getShapeTextureId and sim.createTexture.
C synopsis	simInt simSetShapeTexture(simInt shapeHandle,simInt textureId,simInt mappingMode,simInt options,const simFloat* uvScaling,const simFloat* position,const simFloat* orientation
C parameters	**shapeHandle**: the handle of the shape. **textureId**: the ID of the texture or -1 to remove any existing texture. See also simGetTextureId, simGetShapeTextureId and simCreateTexture. **mappingMode**: the texture mapping mode.

	options: bit-coded: bit0: if set (1), then adjacent texture pixels are not interpolated. bit1: if set (2), then the texture is applied as a decal (its appearance won't be influenced by light conditions). bit2: if set (4), then the texture will be repeated along the U direction. bit3: if set (8), then the texture will be repeated along the V direction. **uvScaling**: a pointer to 2 values that indicate the texture scaling factors along the U and V directions. **position**: a pointer to 3 values (x,y,z) that indicate the texture position on the shape. Can be NULL for default values. **orientation**: a pointer to 3 values (Euler angles) that indicate the texture orientation on the shape. Can be NULL for default values.
C return value	-1 in case of an error.
Lua synopsis	number result=sim.setShapeTexture(number shapeHandle,number textureId,number mappingMode,number options,table_2 uvScaling,table_3 position=nil,table_3 orientation=nil)
Lua parameters	Similar as C-function
Lua return values	Similar as C-function

simSetSimulationPassesPerRenderingPass

Description	Sets the number of simulation passes (calculation passes) per frame (display). This function will have an effect only if using a **custom dt** (which can be set in the simulation settings dialog). See also simGetSimulationPassesPerRenderingPass.
C synopsis	simInt simSetSimulationPassesPerRenderingPass(simInt p)
C parameters	**p**: the number of simulation passes for one rendering pass
C return value	-1 in case of error, otherwise the new number of simulation passes per frame.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simSetSphericalJointMatrix / sim.setSphericalJointMatrix (remote API equivalent: simxSetSphericalJointMatrix)

Description	Sets the intrinsic orientation matrix of a spherical joint object. This function cannot be used with non-spherical joints (use sim.setJointPosition instead). See also sim.getJointMatrix.
C synopsis	simInt simSetSphericalJointMatrix(simInt objectHandle,const simFloat* matrix)
C parameters	**objectHandle**: handle of the joint object **matrix**: pointer to 12 simFloat values (the last row of the 4x4 matrix (0,0,0,1) is not needed) The x-axis of the orientation component is (matrix[0],matrix[4],matrix[8]) The y-axis of the orientation component is (matrix[1],matrix[5],matrix[9]) The z-axis of the orientation component is (matrix[2],matrix[6],matrix[10]) The translation component is (matrix[3],matrix[7],matrix[11]) (the translational components will be ignored)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setSphericalJointMatrix(number objectHandle,table_12 matrix)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetStringParameter / sim.setStringParameter

Description	Sets a string parameter. See also sim.getStringParameter, sim.setBoolParameter, sim.setArrayParameter, sim.setFloatParameter and sim.setInt32Parameter.
C synopsis	simInt simSetStringParameter(simInt parameter,const simChar* stringState)
C parameters	**parameter**: string parameter identifier **stringState**: new state for the parameter
C return value	1 if operation was successful. -1 if parameter is not known
Lua synopsis	number result=sim.setStringParameter(number parameter,string stringState)

Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetStringSignal / sim.setStringSignal (remote API equivalent: simxSetStringSignal)

Description	Sets the value of a string signal. If that signal is not yet present, it is added. Signals created in the main script or in a child script are automatically cleared at simulation end. See also sim.getStringSignal, the other signal functions, the data packing/unpacking functions and sim.persistentDataWrite.
C synopsis	simInt simSetStringSignal(const simChar* signalName,const simChar* signalValue,simInt stringLength)
C parameters	**signalName**: name of the signal **signalValue**: value of the signal (which may contain any value, including embedded zeros) **stringLength**: the size of the string value.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setStringSignal(string signalName,string signalValue)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetThreadAutomaticSwitch / sim.setThreadAutomaticSwitch

Description	Allows to temporarily forbid thread switches. If the current script doesn't run in a thread (i.e. if it runs in the application main thread), this function has no effect. By default, V-REP doesn't use "regular" threads, but something similar to hybrid threads (which behave like coroutines, but can also behave like regular threads). This allows much more flexibility and execution control of the threads. For complete control over the switching moment, see also sim.getThreadAutomaticSwitch, sim.setThreadSwitchTiming, sim.switchThread, sim.setThreadIsFree and sim.setThreadResumeLocation.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result=sim.setThreadAutomaticSwitch(Boolean switchIsAutomatic)
Lua parameters	**switchIsAutomatic**: if true, the thread will be able to automatically switch to another thread, otherwise the switching is temporarily disabled.
Lua return values	**result**: 1 if the command was successful, 0 if it didn't have an effect (e.g. because the function was called from the main or application thread), or -1 in case of an error.

simSetThreadIsFree / sim.setThreadIsFree

Description	Threads created by V-REP are never running concurrently, they rather behave like coroutines. This allows achieving same results as with "pure threads", except when an external command is blocking (e.g. commands not directly supported by V-REP). Those can be lengthly image processing routines, or socket communication routines for example. When such external blocking commands are called, V-REP appears frozen until the external commands return. To avoid such a situation, you can declare a non-blocking section with the sim.setThreadIsFree command: sim.setThreadIsFree(true) starts a non-blocking section, and sim.setThreadIsFree(false) closes it. Try to avoid using V-REP commands when in a non-blocking section (bad synchronization), and never forget to close a non-blocking section, otherwise V-REP will hang indefinitely. Use sim.setThreadIsFree with extra care when calling it from C. A thread running in a non-blocking section cannot be paused nor stopped. This command has no effect when called from the main thread or a non-threaded script.
C synopsis	simInt simSetThreadIsFree(simBool freeMode)
C parameters	**freeMode**: specify 1 to start a non-blocking section. Specify 0 to end a non-blocking section
C return value	1 if operation was successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.setThreadIsFree(Boolean freeMode)
Lua parameters	**freeMode**: specify true to start a non-blocking section. Specify false to end a non-blocking section
Lua return values	Same as C-function

simSetThreadResumeLocation / sim.setThreadResumeLocation

Description	Allows specifying when and in which order child script threads are resumed. If the current script doesn't run in a thread (i.e. if it runs in the application main thread), this function has no effect. By default, V-REP doesn't use "regular" threads, but something similar to hybrid threads (which behave like coroutines, but can also behave like regular threads). This allows much more flexibility and execution control of the threads. Once a thread switched to another thread, it will resume execution when the main script calls sim.resumeThreads with the corresponding argument, which represents a child script thread resume location. In order to also have full synchronization control between threads, you can assign a resume location and order to each thread with this function. See also sim.setThreadSwitchTiming, sim.setThreadAutomaticSwitch, sim.switchThread and sim.setThreadIsFree.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result=sim.setThreadResumeLocation(number location,number order)
Lua parameters	**location**: a child script thread resume location. **order**: a script resume or execution order.
Lua return values	**result**: 1 if the command was applied, 0 if it was not (e.g. because the function was called from the main or application thread), or -1 in case of an error.

simSetThreadSwitchTiming / sim.setThreadSwitchTiming

Description	Allows specifying a switching time for the thread in which the current script runs. If the current script doesn't run in a thread (i.e. if it runs in the application main thread), this function has no effect. By default, V-REP doesn't use "regular" threads, but something similar to hybrid threads (which behave like coroutines, but can also behave like regular threads). This allows much more flexibility and execution control of the threads: each thread (except for the main or application thread) has a switch timing associated, which specifies how long the thread will run before switching to other threads (the execution duration per calculation pass). By default this value is 2 millisecond, but this function allows changing that value (on a thread-basis). Acceptable values are between 0 and 200. For complete control over the switching moment, see also sim.setThreadAutomaticSwitch, sim.switchThread, sim.setThreadIsFree and sim.setThreadResumeLocation.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result=sim.setThreadSwitchTiming(number deltaTimeInMilliseconds)
Lua parameters	**deltaTimeInMilliseconds**: desired non-stop execution time before a switching occurs. A value of x will let the thread execute for x-1 to x milliseconds before switching to another thread.
Lua return values	**result**: 1 if the timing was set, 0 if it was not set (e.g. because the function was called from the main or application thread), or -1 in case of an error.

simSetUIButtonArrayColor (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simSetUIButtonColor (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simSetUIButtonLabel (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simSetUIButtonProperty (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simSetUIButtonTexture (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simSetUIPosition (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simSetUIProperty (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simSetUISlider (DEPRECATED)

Description	DEPRECATED. Use the Qt-based custom user interfaces instead.

simSetVisionSensorCharImage / sim.setVisionSensorCharImage (remote API equivalent: simxSetVisionSensorImage)

Description	Sets the rgb-image of a vision sensor (and applies any image processing filter if specified in the vision sensor dialog). Make sure the vision sensor is flagged as **external input**. Use sim.getVisionSensorResolution to know the size of the image buffer that you need to provide (buffer size=resolutionX *resolutionY*3). The "regular" use of this function is to first read the data from a vision sensor with sim.getVisionSensorCharImage, do some custom filtering, then write the modified image back. The alternate use of this function is to display textures, video images, etc. by using a vision sensor object (without however making use of the vision sensor functionality), since a vision sensor can be "looked through" like camera objects. See also sim.setVisionSensorImage.
C synopsis	simInt simSetVisionSensorCharImage(simInt sensorHandle,const simUChar* image)
C parameters	**sensorHandle**: handle of the vision sensor object. Can be combined with sim_handleflag_greyscale (simply add sim_handleflag_greyscale to sensorHandle), if you wish to provide grey scale values instead of rgb values.. **image**: rgb buffer containing the image (buffer size must be resolutionX*resolutionY*3). Values in the buffer should vary between 0 and 255. In case a grey scale image is provided, the buffer size must be resolutionX*resolutionY.
C return value	-1 if operation was not successful. 0 if the applied filter didn't trigger anything, 1 if the appIied filter triggered a detection
Lua synopsis	number result=sim.setVisionSensorCharImage(number sensorHandle,string image)
Lua parameters	Similar as C-function
Lua return values	Same as C-function

simSetVisionSensorFilter / sim.setVisionSensorFilter

Description	Sets the parameters and settings of a specific filter component of a vision sensor. See also sim.getVisionSensorFilter and the other vision sensor related API functions.
C synopsis	simInt simSetVisionSensorFilter(simInt visionSensorHandle,simInt filterIndex,simInt options,const simInt* pSizes,const simUChar* bytes,const simInt* ints,const simFloat* floats,const simUChar* custom)
C parameters	**visionSensorHandle**: handle of a vision sensor. See also simGetObjectHandle. **filterIndex**: the zero-based index of the filter position. **options**: bit-coded value: bit 0 set (1): the component is enabled **pSizes**: a pointer to 4 integer values indicating the sizes of the provided buffers (see next 4 arguments). **bytes**: a buffer of bytes values representing the byte parameters of the filter component. **ints**: a buffer of ints values representing the int parameters of the filter component. **floats**: a buffer of floats values representing the float parameters of the filter component. **custom**: a buffer of bytes values representing the custom parameters of the filter component.
C return value	-1 in case of an error, 0 if the *filterIndex* is not valid, otherwise the type of filter component pointed by the *filterIndex*.
Lua synopsis	number filterType=sim.setVisionSensorFilter(number sensorHandle,number filterIndex,number options,table byteVals,table intVals,table floatVals,string customBuffer)
Lua parameters	Similar as C-function
Lua return values	Same as C-function

simSetVisionSensorImage / sim.setVisionSensorImage (remote API equivalent: simxSetVisionSensorImage)

Description	Sets the rgb-image of a vision sensor (and applies any image processing filter if specified in the vision sensor dialog). Make sure the vision sensor is flagged as **external input**. Use sim.getVisionSensorResolution to know the size of the image buffer that you need to provide (buffer size=resolutionX *resolutionY*3). The "regular" use of this function is to first read the data from a vision sensor with sim.getVisionSensorImage, do some custom filtering, then write the modified image back. The alternate use of this function is to display textures, video images, etc. by using a vision sensor object (without however making use of the vision sensor functionality), since a vision sensor can be "looked through" like camera objects. See also sim.setVisionSensorCharImage.
C synopsis	simInt simSetVisionSensorImage(simInt sensorHandle,const simFloat* image)
C parameters	**sensorHandle**: handle of the vision sensor object. Can be combined with sim_handleflag_greyscale (simply add sim_handleflag_greyscale to sensorHandle), if you wish to provide grey scale values instead of rgb values. **image**: rgb buffer containing the image (buffer size must be resolutionX*resolutionY*3). Values in the buffer should vary between 0 and 1. In case a grey scale image is provided, the buffer size must be resolutionX*resolutionY.
C return value	-1 if operation was not successful. 0 if the applied filter didn't trigger anything, 1 if the appIied filter triggered a detection
Lua synopsis	(1) number result=sim.setVisionSensorImage(number sensorHandle,table image) (2) number result=sim.setVisionSensorImage(number sensorHandle,string image)
Lua parameters	**sensorHandle**: handle of the vision sensor object. Can be combined with sim.handleflag_greyscale (simply add sim.handleflag_greyscale to sensorHandle), if you wish to provide grey scale values instead of rgb values. (1) a table containing individual values [0-1] for the red, green and blue components, or for the grey component. (2) a string containing individual chars [0-255] for the red, green and blue components, or for the grey component.
Lua return values	Same as C-function

simSimplifyMpPath (DEPRECATED)

Description	DEPRECATED. See the OMPL library based path/motion planning functionality instead. Simplifies a path retrieved via the motion planning function simFindMpPath. The function uses V-REP's

	motion planning functionality.
C synopsis	simFloat* simSimplifyMpPath(simInt motionPlanningObjectHandle,const simFloat* pathBuffer,simInt configCnt,simInt options,simFloat stepSize,simInt increment,simInt* outputConfigsCnt,simInt maxTimeInMs,simFloat* reserved,const simInt* auxIntParams,const simFloat* auxFloatParams)
C parameters	**motionPlanningObjectHandle**: the handle of a motion planning object. Refer to simGetMotionPlanningHandle. **pathBuffer**: the buffer returned by the simFindMpPath function.. **configCnt**: the number of configuration nodes the pathBuffer contains. **options**: bit-coded: bit0: not used. Keep unset. bit1: not used. Keep unset. bit2: if set (4), then the found path will be visualized in yellow. bit3: if set (8), then some information will be output to the console. bit4: if set (16), then robot self-interferences will be ignored and calculations can drastically be sped-up. bit5: if set (32), then robot-environment interferences will be ignored and calculations can drastically be sped-up. bit6: not used, keep unset. bit7: not used, keep unset. bit8: if set (256), then the returned Cartesian space distances will ignore the orientational distance component. **stepSize**: the maximum configuration space distance between individual collision-free phase2 nodes. the distance calculation will use the weight specified for each joint in the motion planning properties. **increment**: a value indicating how fine-grainer the simplification should be. 1 for best quality. **outputConfigsCnt**: a pointer to an integer receiving the number of returned configurations. **maxTimeInMs**: the maximum time in milliseconds after which the simplification operation is aborted. Specify 0 for an infinite time. **reserved**: reserved. Keep NULL. **auxIntParams**: reserved. Keep NULL. **auxFloatParams**: reserved. Keep NULL.
C return value	NULL in case of an error, or when the search failed. Otherwise a buffer of float values that the user is in charge of releasing with simReleaseBuffer. The returned buffer contains: **the found path** (x*n values): n configurations with each x values (x is the number of DoFs of the specified motion planning task). The configurations will include the start and the goal configuration, except when start and goal are coincident, in which case a single configuration is returned. **the configuration space distances** (n values): for each returned configuration, a distance to the start configuration (following the path). The last of the n values represents the length of the found path in the configuration space. The distance is calculated using the weight specified for each joint in the motion planning properties. **the end-effector positions** (3*n values): for each returned configuration, the position of the corresponding end-effector (x, y, z). **the end-effector quaternions** (4*n values): for each returned configuration, the quaternion of the corresponding end-effector (x, y, z, w). **the Cartesian space distances** (n values): for each returned configuration, a distance to the start pose (following the path). The last of the n values represents the length of the found path in the Cartesian space. The distance is calculated using the Cartesian space metric specified in the motion planning properties.
Lua synopsis	table path,table confSpaceDist,table tipPositions,table tipQuaternions,table cartesianSpaceDist=simSimplifyMpPath(number motionPlanningObjectHandle,table path,number configCnt,number options,number stepSize,number increment,number maxTimeInMs=0,table auxIntParams=nil,table auxFloatParams=nil)
Lua parameters	Similar as C-function
Lua return values	Similar as C-function

simStartSimulation / sim.startSimulation (remote API equivalent: simxStartSimulation)

Description	Requests a start of a simulation (or a resume of a paused simulation). See also sim.pauseSimulation, sim.stopSimulation and sim.getSimulationState. See also the simulation state diagram.
C synopsis	simInt simStartSimulation()
C parameters	None
C return value	-1 in case of an error, 0 if the operation could not be performed. >0 in case of success.
Lua synopsis	number result=sim.startSimulation()
Lua parameters	Same as C-function
Lua return values	Same as C-function

simStopSimulation / sim.stopSimulation (remote API equivalent: simxStopSimulation)

Description	Requests a stop of the running simulation. See also sim.startSimulation, sim.pauseSimulation and sim.getSimulationState. See also the simulation state diagram.
C synopsis	simInt simStopSimulation()
C parameters	None
C return value	-1 in case of an error, 0 if the operation could not be performed. >0 in case of success.
Lua synopsis	number result=sim.stopSimulation()
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSubtractObjectFromOctree / sim.subtractObjectFromOctree

Description	Removes an object from an octree, as voxel subtractions. See also sim.insertObjectIntoOctree, sim.removeVoxelsFromOctree and the other octree related functions.
C synopsis	simInt simSubtractObjectFromOctree(simInt octreeHandle,simInt objectHandle,simInt options,simVoid* reserved)
C parameters	**octreeHandle**: the handle of the octree. See also simGetObjectHandle **objectHandle**: the handle of the object to subtract. Only potentially collidable objects are supported **options**: reserved. Set to 0 **reserved**: reserved for future extensions. Set to NULL
C return value	-1 if operation was not successful, otherwise the total number of voxels in the octree
Lua synopsis	number totalVoxelCnt=sim.subtractObjectFromOctree(number octreeHandle,number objectHandle,number options)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSubtractObjectFromPointCloud / sim.subtractObjectFromPointCloud

Description	Removes an object from a point cloud, as a subtraction. See also sim.insertPointsIntoPointCloud, sim.insertObjectIntoPointCloud, sim.removePointsFromPointCloud and the other point cloud related functions.
C synopsis	simInt simSubtractObjectFromPointCloud(simInt pointCloudHandle,simInt objectHandle,simInt options,simFloat tolerance,simVoid* reserved)
C parameters	**pointCloudHandle**: the handle of the point cloud. See also simGetObjectHandle **objectHandle**: the handle of the object to subtract. Only potentially measurable objects are supported. **options**: reserved. Set to 0 **tolerance**: a distance used as a tolerance value **reserved**: reserved for future extensions. Set to NULL
C return value	-1 if operation was not successful, otherwise the total number of points in the point cloud
Lua synopsis	number totalPointCnt=sim.subtractObjectFromPointCloud(number pointCloudHandle,number objectHandle,number options,number tolerance)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSwitchThread / sim.switchThread

Description	Allows specifying the exact moment at which the current thread should switch to another thread. If the current script doesn't run in a thread (i.e. if it runs in the application main thread), this function has no effect. By default, V-REP doesn't use "regular" threads, but something similar to hybrid threads (which behave like coroutines, but can also behave like regular threads). This allows much more flexibility and

	execution control of the threads: each thread (except for the main or application thread) has a switch timing associated, which specifies how long the thread will run before switching to other threads. By default this value is 2 millisecond, but can be modified with sim.setThreadSwitchTiming. That timing can be shortened with sim.switchThread. Use with care when calling this function from a plugin. See also the sim.setThreadAutomaticSwitch, sim.setThreadResumeLocation and sim.setThreadIsFree functions.
C synopsis	simInt simSwitchThread()
C parameters	None
C return value	1 if the thread was switched (the current thread gave control to other threads until the next calculation pass), 0 if it was not switched (e.g. because the function was called from the main or application thread, or from a thread started by the user), or -1 in case of an error.
Lua synopsis	number result=sim.switchThread()
Lua parameters	None
Lua return values	**result**: 1 if the thread was switched (the current thread gave control to other threads until the next calculation pass), 0 if it was not switched (e.g. because the function was called from the main or application thread), or -1 in case of an error.

simTransformBuffer / sim.transformBuffer

Description	Modified a buffer than contains packed data. See also the data packing/unpacking functions.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	1) string outBuffer=sim.transformBuffer(string inBuffer,number inFormat,number multiplier,number offset,number outFormat) 2) table outBuffer=sim.transformBuffer(string inBuffer,sim.buffer_uint8,0,number splitSize,sim.buffer_split)
Lua parameters	**inBuffer**: the input buffer that contains packed data. **inFormat**: a buffer type. **multiplier**: a multiplier value. We have out=offset+multiplier*in **offset/splitSize**: an offset or split size value. We have out=offset+multiplier*in **outFormat**: the desired buffer type for the returned buffer.
Lua return values	1) **outBuffer**: the modified buffer, or nil in case of an error 2) **table**: the various split buffers, or nil in case of an error

simTransformImage / sim.transformImage

Description	Transforms an image in various ways. See also sim.loadImage, sim.getScaledImage, sim.transformBuffer and sim.combineRgbImages.
C synopsis	simInt simTransformImage(simUChar* image,const simInt* resolution,simInt options,const simFloat* floatParams,const simInt* intParams,simVoid* reserved)
C parameters	**image**: a pointer to rgb or rgba values of the image **resolution**: the resolution of the image **options**: bit-coded: bit0 set (1): the provided image is rgba, otherwise it is rgb bit1 set (2): the image will be flipped on its x-axis bit2 set (4): the image will be flipped on its y-axis **floatParams**: Reserved for future extension. Set to NULL. **intParams**: Reserved for future extension. Set to NULL. **reserved**: Reserved for future extension. Set to NULL.
C return value	-1 if operation was not successful.
Lua synopsis	number result=sim.transformImage(string image,table_2 resolution,number options)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simTransformVector

Description	Multiplies a vector with a transformation matrix (v=m*v). See also the other matrix/transformation functions.
C synopsis	simInt simTransformVector(const simFloat* matrix,simFloat* vect)
C parameters	**matrix**: the transformation matrix The x-axis of the orientation component is (matrix[0],matrix[4],matrix[8]) The y-axis of the orientation component is (matrix[1],matrix[5],matrix[9]) The z-axis of the orientation component is (matrix[2],matrix[6],matrix[10]) The position component is (matrix[3],matrix[7],matrix[11]) **vect**: the vector to be transformed (a pointer to 3 values (the last element of the homogeneous coordinates is not required (1)))
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	See sim.multiplyVector
Lua parameters	-
Lua return values	-

simTubeClose / sim.tubeClose

Description	Closes a communication tube previously opened with sim.tubeOpen. Data written with sim.tubeWrite and that hasn't been read yet on the other side of the tube will persist.
C synopsis	simInt simTubeClose(simInt tubeHandle)
C parameters	**tubeHandle**: the handle of the tube that was returned by the simTubeOpen function. Once a tube was closed on one side, it is again ready for a new connection.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.tubeClose(number tubeHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simTubeOpen / sim.tubeOpen

Description	Opens a tube for communication within V-REP. A tube is similar to a bidirectional communication pipe. Messages written on one side can be read on the other side in the same order as they were written. Tubes opened via a script will automatically close upon simulation end. A scene switch will close all communication tubes and tube handles will not be valid anymore. See also sim.tubeClose, sim.tubeWrite, sim.tubeRead, sim.tubeStatus, sim.sendData and sim.receiveData.
C synopsis	simInt simTubeOpen(simInt dataHeader,const simChar* dataName,simInt readBufferSize,simBool notUsedButKeepFalse)
C parameters	**dataHeader**: number indicating who "designed" the communication message. Always use the same header (because only you will know the meaning of the message) and stick to it. The best is to use the serial number of your V-REP copy (check the "Help" menu, in the "About" item for the serial number). Otherwise, you risk collision with other developer's messages which might use the same header as yours. **dataName**: name indicating the type of message. A tube will only be able to connect if its two sides have the same dataHeader and dataName **readBufferSize**: the number of data packets that can be stored in the input buffer of this side of the tube **notUsedButKeepFalse**: not used. Keep to false.
C return value	-1 if operation was not successful, otherwise the tube handle.
Lua synopsis	number tubeHandle=sim.tubeOpen(number dataHeader,string dataName,number readBufferSize)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simTubeRead / sim.tubeRead

Description	Receives a data packet from a communication tube previously opened with sim.tubeOpen. The tube needs to be connected (see sim.tubeStatus).

C synopsis	simChar* simTubeRead(simInt tubeHandle,simInt* dataLength)
C parameters	**tubeHandle**: the handle of the tube that was returned by the simTubeOpen function. **dataLength**: size of the returned data packet
C return value	Pointer to a data packet, or NULL if nothing could be read. The user is in charge of releasing the returned buffer with simReleaseBuffer.
Lua synopsis	string data=sim.tubeRead(number tubeHandle,boolean blockingOperation=false)
Lua parameters	**tubeHandle**: Same as C-function **blockingOperation**: if true, then the call will block until something can be read or until an error occurred. The script should be threaded in that case. Default value is false.
Lua return values	**data**: string containing the received data, or nil in case of an error or if no data is present. If received data is packed, see also the data packing/unpacking functions.

simTubeStatus / sim.tubeStatus

Description	Checks the status of a communication tube previously opened with sim.tubeOpen.
C synopsis	simInt simTubeStatus(simInt tubeHandle,simInt* readPacketsCount,simInt* writePacketsCount)
C parameters	**tubeHandle**: the handle of the tube that was returned by the simTubeOpen function. **readPacketsCount**: pointer to an integer that will indicate the number of packets waiting to be read. Can be NULL **writePacketsCount**: pointer to an integer that will indicate the number of packets waiting to be read on the other side of the tube. Can be NULL
C return value	-1 if operation was not successful, 0 if the tube is not connected, 1 if the tube is connected
Lua synopsis	number status,number readPacketsCount,number writePacketsCount=sim.tubeStatus(number tubeHandle)
Lua parameters	Same as C-function
Lua return values	**status**: nil if operation was not successful, otherwise 0 if the tube is not connected, 1 if the tube is connected **readPacketsCount**: nil if operation was not successful, otherwise the number of packets waiting to be read **writePacketsCount**: nil if operation was not successful, otherwise the number of packets waiting to be read on the other side of the tube

simTubeWrite / sim.tubeWrite

Description	Sends a data packet into a communication tube previously opened with sim.tubeOpen. The tube needs to be connected (see sim.tubeStatus).
C synopsis	simInt simTubeWrite(simInt tubeHandle,const simChar* data,simInt dataLength)
C parameters	**tubeHandle**: the handle of the tube that was returned by the simTubeOpen function. **data**: pointer to a data packet. simTubeWrite will copy that packet so that it can directly be released **dataLength**: size of the data packet
C return value	1 if operation was successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.tubeWrite(number tubeHandle,string data)
Lua parameters	**tubeHandle**: Same as C-function **data**: string representing a data packet (each character can have values between 0 and 255). See also the data packing/unpacking functions.
Lua return values	Same as C-function

simUnfoldStackTable

Description	Extracts all key-value pairs from the table at the top of the stack, and removes the table. See also simGetStackSize and the other stack functions.
C synopsis	simInt simUnfoldStackTable(simInt stackHandle)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack.
C return value	-1 in case of an error
Lua synopsis	-
Lua parameters	-

Lua return values	-

simUngroupShape / sim.ungroupShape

Description	Ungroups a compound shape into several simple shapes. See also sim.groupShapes, sim.convexDecompose, sim.getQHull and sim.getDecimatedMesh.
C synopsis	simInt* simUngroupShape(simInt shapeHandle,simInt* shapeCount)
C parameters	**shapeHandle**: the handles of the shape you wish to ungroup. If you specify for this argument (-2-handleOfShape), then the shape will be divided instead of ungrouped. **shapeCount** (output): the size of the returned buffer.
C return value	a pointer to an array holding the handles of the resulting shapes, or NULL in case of an error. The user is in charge of releasing the array buffer with simReleaseBuffer.
Lua synopsis	table simpleShapeHandles=sim.ungroupShape(number shapeHandle)
Lua parameters	Similar to C-function
Lua return values	Similar to C-function

simUnloadModule / sim.unloadModule

Description	Unloads a V-REP plugin. This should usually be done just before ending the simulator. Alternatively, you can also unload a plugin that was dynamically loaded at any time. This can however lead to a crash if the plugin registered custom Lua functions via simRegisterCustomLuaFunction (deprecated) without specifying a function name as *functionName@pluginName*. See also simSendModuleMessage and sim.loadModule.
C synopsis	simInt simUnloadModule(simInt pluginhandle)
C parameters	**pluginhandle**: handle of the plugin
C return value	0 if plugin was not unloaded (e.g. because it was not loaded, or because it was loaded more than once). Different from 0 if the plugin was successfully unloaded
Lua synopsis	number result=sim.unloadModule(number pluginHandle)
Lua parameters	Similar to C-function
Lua return values	Similar to C-function

simUnlockResources

Description	Unlocks resources previously locked via simLockResources.
C synopsis	simInt simUnlockResources(simInt lockHandle)
C parameters	**lockHandle**: the lock handle returned by simLockResources.
C return value	-1 in case of an error.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simUnpackBytes / sim.unpackUInt8Table (DEPRECATED)

Description	DEPRECATED. Refer to sim.unpackUInt8Table instead.

simUnpackDoubleTable / sim.unpackDoubleTable

Description	Unpacks a string (or part of it) into a table of double floating-point numbers. See also sim.packDoubleTable and the other packing/unpacking functions.

C synopsis	-
C parameters	-
C return value	-
Lua synopsis	table doubleNumbers=sim.unpackDoubleTable(string data,number startDoubleIndex=0,number doubleCount=0,number additionalByteOffset=0)
Lua parameters	**data**: a string (values between 0 and 255) that contains packed floating-point numbers **startDoubleIndex**: the zero-based index from which on data should be unpacked (from data[8*startDoubleIndex+1+additionalByteOffset]). Can be omitted in which case 0 is used **doubleCount**: the amount of doubles that should be unpacked. Can be omitted in which case 0 is used (which indicates that the maximum of doubles should be unpacked from the indicated startIndex) **additionalByteOffset**: a byte offset that will be added before reading the doubles. Can be omitted, in which case 0 is used.
Lua return values	**doubleNumbers**: a table containing unpacked double floating-point numbers if operation was successful, nil otherwise

simUnpackDoubles / sim.unpackDoubleTable (DEPRECATED)

Description	DEPRECATED. Refer to sim.unpackDoubleTable instead.

simUnpackFloatTable / sim.unpackFloatTable

Description	Unpacks a string (or part of it) into a table of floating-point numbers. See also sim.packFloatTable and the other packing/unpacking functions.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	table floatingNumbers=sim.unpackFloatTable(string data,number startFloatIndex=0,number floatCount=0,number additionalByteOffset=0)
Lua parameters	**data**: a string (values between 0 and 255) that contains packed floating-point numbers **startFloatIndex**: the zero-based index from which on data should be unpacked (from data[4*startFloatIndex+1+additionalByteOffset]). Can be omitted in which case 0 is used **floatCount**: the amount of floats that should be unpacked. Can be omitted in which case 0 is used (which indicates that the maximum of floats should be unpacked from the indicated startFloatIndex) **additionalByteOffset**: a byte offset that will be added before reading the floats. Can be omitted, in which case 0 is used.
Lua return values	**floatingNumbers**: a table containing unpacked floating-point numbers if operation was successful, nil otherwise

simUnpackFloats / sim.unpackFloatTable (DEPRECATED)

Description	DEPRECATED. Refer to sim.unpackFloatTable instead.

simUnpackInt32Table / sim.unpackInt32Table

Description	Unpacks a string (or part of it) into a table of int32 numbers. See also sim.packInt32Table and the other packing/unpacking functions.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	table int32Numbers=sim.unpackInt32Table(string data,number startInt32Index=0,number int32Count=0,number additionalByteOffset=0)
Lua parameters	**data**: a string (values between 0 and 255) that contains packed int32 numbers **startInt32Index**: the zero-based index from which on data should be unpacked (from data[4*startInt32Index+1+additionalByteOffset]). Can be omitted in which case 0 is used

	int32Count: the amount of int32s that should be unpacked. Can be omitted in which case 0 is used (which indicates that the maximum of int32s should be unpacked from the indicated startInt32Index) **additionalByteOffset**: a byte offset that will be added before reading the int32s. Can be omitted, in which case 0 is used.
Lua return values	**int32Numbers**: a table containing unpacked int32 numbers if operation was successful, nil otherwise

simUnpackInts / sim.unpackInt32Table (DEPRECATED)

Description	DEPRECATED. Refer to sim.unpackInt32Table instead.

simUnpackTable / sim.unpackTable

Description	Unpacks a buffer into a table. See also sim.packTable, the other stack functions and the other packing/unpacking functions.
C synopsis	simInt simUnpackTable(simInt stackHandle,const simChar* buffer,simInt bufferSize)
C parameters	**stackHandle**: a stack handle obtained with simCreateStack. The unpacked table will be pushed onto the stack. **buffer**: the packed table (buffer). **bufferSize**: the size of the buffer.
C return value	-1 in case of an error.
Lua synopsis	table aTable=sim.unpackTable(string buffer)
Lua parameters	**buffer**: a string buffer.
Lua return values	**aTable**: a script table.

simUnpackUInt16Table / sim.unpackUInt16Table

Description	Unpacks a string (or part of it) into a table of uint16 numbers. See also sim.packUInt16Table and the other packing/unpacking functions.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	table uint16Numbers=sim.unpackUInt16Table(string data,number starUint16Index=0,number uint16Count=0,number additionalByteOffset)
Lua parameters	**data**: a string (values between 0 and 255) that contains packed uint16 numbers **starUint16Index**: the zero-based index from which on data should be unpacked (from data[2*starUint16Index+1+additionalByteOffset]). Can be omitted in which case 0 is used. **uint16Count**: the amount of uint16s that should be unpacked. Can be omitted in which case 0 is used (which indicates that the maximum of uint16s should be unpacked from the indicated starUint16Index). **additionalByteOffset**: a byte offset that will be added before reading the uint16s. Can be omitted, in which case 0 is used.
Lua return values	**uint16Numbers**: a table containing unpacked uint16 numbers if operation was successful, nil otherwise

simUnpackUInt32Table / sim.unpackUInt32Table

Description	Unpacks a string (or part of it) into a table of uint32 numbers. See also sim.packUInt32Table and the other packing/unpacking functions.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	table uint32Numbers=sim.unpackUInt32Table(string data,number startUint32Index=0,number uint32Count=0,number additionalByteOffset=0)
Lua parameters	**data**: a string (values between 0 and 255) that contains packed uint32 numbers

	startUint32Index: the zero-based index from which on data should be unpacked (from data[4*startUint32Index+1+additionalByteOffset]). Can be omitted in which case 0 is used **uint32Count**: the amount of uint32s that should be unpacked. Can be omitted in which case 0 is used (which indicates that the maximum of uint32s should be unpacked from the indicated startUint32Index) **additionalByteOffset**: a byte offset that will be added before reading the uint32s. Can be omitted, in which case 0 is used.
Lua return values	**integerNumbers**: a table containing unpacked uint32 numbers if operation was successful, nil otherwise

simUnpackUInt8Table / sim.unpackUInt8Table

Description	Unpacks a string (or part of it) into a table of uint8 numbers. See also sim.packUInt8Table and the other packing/unpacking functions.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	table uint8Numbers=sim.unpackUInt8Table(string data,number startUint8Index=0,number uint8Count=0)
Lua parameters	**data**: a string (values between 0 and 255) that contains uint8 numbers **startUint8Index**: the zero-based index from which on data should be unpacked (from data[startUint8Index]). Can be omitted in which case 0 is used. **uint8Count**: the amount of uint8s that should be unpacked. Can be omitted in which case 0 is used (which indicates that the maximum of uint8s should be unpacked from the indicated startUint8Index).
Lua return values	**uint8Numbers**: a table containing uint8 numbers if operation was successful, nil otherwise

simUnpackUInts / sim.unpackUInt32Table (DEPRECATED)

Description	DEPRECATED. Refer to sim.unpackUInt32Table instead.

simUnpackWords / sim.unpackUInt16Table (DEPRECATED)

Description	DEPRECATED. Refer to sim.unpackUInt16Table instead.

simWait / sim.wait

Description	Waits for a certain amount of time. This function can only be called from child scripts running in a thread (since this is a blocking operation) and is not available from the C-API. See also sim.waitForSignal.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number deltaTimeLeft=sim.wait(number deltaTime,Boolean simulationTime)
Lua parameters	**deltaTime**: the minimum time duration to wait. If the function was called at simulation time X (or real-time X), then the function will return when simulation time>=X+deltaTime (or when real-time>=X+deltaTime) **simulationTime**: indicates whether we want to wait in terms of simulation- or real-time. Can be omitted (is true by default)
Lua return values	**deltaTimeLeft**: the "wait resolution" of this function is the simulation time step, and the sim.wait command may overshoot the requested waiting time. deltaTimeLeft is the negative overshoot time. If the function was called at simulation time X, and the function returned at simulation time Y, then deltaTimeLeft is deltaTime-(Y-X). In case of an error, deltaTimeLeft is nil. deltaTimeLeft is also memorized internally on a thread-basis and used as compensation or correction factor in subsequent blocking commands. deltaTimeLeft is 0 if the simulationTime argument was false

simWaitForSignal / sim.waitForSignal

Description	Waits for a signal. Signals are cleared at simulation start. This function will first check whether an integer signal with that name is present, then if a float signal with that name is present and finally if a string signal with that name is present. The function only returns when the signal is present (defined). This function can only be called from child scripts running in a thread (since this is a blocking operation) and is not available from the C-API. See also the other signal functions and sim.wait.
C synopsis	-
C parameters	-
C return value	-

Lua synopsis	Number/string signalValue=sim.waitForSignal(string signalName)
	
	The function is equivalent to:
	
	`signalValue=nil`
	`while (signalValue==nil) do`
	`  signalValue=sim.getIntegerSignal(signalName) or sim.getFloatSignal(signalName) or sim.getStringSignal(signalName)`
	`  if (signalValue==nil) then`
	`  sim.switchThread()`
	`  end`
	`end`

Lua parameters	**signalName**: name of the signal
Lua return values	**signalValue**: value of the signal, or nil when the command is cancelled by stopping the simulation

simWaitForWorkThreads (DEPRECATED)

Description	DEPRECATED. Has no effect.

simWriteCustomDataBlock / sim.writeCustomDataBlock

Description	Adds or removes custom data to be stored and saved together with an object, a script or a scene (i.e. the data will be part of the object, the script or the scene). If the tag name starts with the string "@tmp", then the data will not be saved during a scene or model save operation. The data can also be saved globally for the application (for the current V-REP session). See also sim.readCustomDataBlock and the data packing/unpacking functions. If you wish to store a reference to another object, have a look at sim.setReferencedHandles.
C synopsis	simInt simWriteCustomDataBlock(simInt objectHandle,const simChar* tagName,const simChar* data,simInt dataSize)
C parameters	**objectHandle**: handle of the object or script where you want to store your data, or sim_handle_scene if you wish to store the data with the scene, or sim_handle_app if you wish to store the data with the application's current session. **tagName**: a string that identifies the data. An empty string will remove all custom data blocks. **data**: your custom data. If NULL, the current data under the specified dataName will be removed. **dataSize**: the size of your custom data The data will be copied to an internal buffer inside of the object or scene, and next time a scene or model is saved, will also be saved. The data buffer can be released after this call.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=sim.writeCustomDataBlock(number objectHandle,string tagName,string data)
Lua parameters	Same as C-function. In addition, you can specify sim.handle_self for the **objectHandle** argument, if your target is the current script.
Lua return values	Same as C-function.

simWriteTexture / sim.writeTexture

Description	Overwrites a specific texture (or a portion of it) with RGB data. See also sim.getTextureId, sim.readTexture and sim.createTexture.
C synopsis	simInt simWriteTexture(simInt textureId,simInt options,const simChar* textureData,simInt posX,simInt posY,simInt sizeX,simInt sizeY,simFloat interpolation)
C parameters	**textureId**: the ID of the texture. See also simGetTextureId. **options**: bit-coded: **bit0** reserved. Do not set. **bit1** reserved. Do not set. **bit2** set (4)=only an elliptical/circular portion of the texture data will be written. **textureData**: RGB data to write onto the texture. Each pixel is represented with 3 bytes (0-255). **posX / posY**: the x/y position where to copy the RGB data. Set to 0/0 to overwrite the full texture **sizeX / sizeY**: the x/y size of the RGB data. Set to 0/0 to overwrite the full texture **interpolation**: the fade or interpolation factor. 0 for no fade.
C return value	-1 in case of an error.
Lua synopsis	number result=sim.writeTexture(number textureId,number options,string textureData,number posX=0,number posY=0,number sizeX=0,number sizeY=0)
Lua parameters	Same as C-function
Lua return values	Same as C-function