Regular API Functions

The list of API functions below allows you to access many V-REP parameters. There are however too many parameters in V-REP to have a specific API function for each one of them. Auxiliary parameters can be accessed via a set of given functions that use object parameter IDs. Refer also to the global parameter IDs.

simAddBanner

Description	Adds a banner to the scene. Banners created in a simulation script will be automatically removed at simulation end. See also simRemoveBanner and simAddDrawingObject
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=simAddBanner(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

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 simAddDrawingObjectItem, simRemoveDrawingObject and simAddPointCloud.
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=simAddDrawingObject(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

Description	Adds an item (or clears all items) to a previously inserted drawing object. See also simAddDrawingObject and simRemoveDrawingObject
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. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simAddDrawingObjectItem(number drawingObjectHandle,table itemData)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddForce

Description	Adds a non-central force to a shape object that is dynamically enabled. Added forces are cumulative, and are reset to zero after simHandleDynamics was called. See also simAddForceAndTorque.
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=simAddForce(number shapeHandle,table_3 position,table_3 force)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddForceAndTorque

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 simHandleDynamics was called. See also simAddForce.
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=simAddForceAndTorque(number shapeHandle,table_3 force,table_3 torque)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddGhost

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 simModifyGhost. 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=simAddGhost(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

Description	Adds or removes custom data to be stored and saved together with an object. This function is useful for external applications or plugin which want to save their data together with an object. Use in conjunction with simGetObjectCustomData and simGetObjectCustomDataLength. See also simWriteCustomDataBlock and simAddSceneCustomData.
C synopsis	simInt simAddObjectCustomData(simInt objectHandle,simInt header,const simChar* data,simInt dataLength)
C parameters	**objectHandle**: handle of the object where you want to store your data **header**: identifier for the custom data. If you plan to add custom data (as a company or individual), always use the same header (because only you will know what data type is stored under that header) 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 data which might use the same header as yours. **data**: your custom data. If NULL, the current data under that header will be removed. If you have several items or data types to save, it is your responsibility to pack and code it in data (don't use various headers for each of your items that you want to save with an object (risk of collision with other developers data (see above))!) **dataLength**: the length of your custom data The data will be copied to an internal buffer inside of the object, 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=simAddObjectCustomData(number objectHandle,number header,string data)
Lua parameters	Same as C-function. (remember that a string in Lua might contain any character, also embedded zeros)
Lua return values	Same as C-function.

simAddObjectToCollection

Description	Adds an object (or a group of objects) to a collection. See also simRemoveCollection, simGetCollectionHandle, simGetObjectHandle and simCreateCollection.
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=simAddObjectToCollection(number collectionHandle,number objectHandle,number what,number options)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddObjectToSelection

Description	Adds an object to the selection. See also simRemoveObjectFromSelection and simGetObjectSelection.
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=simAddObjectToSelection(number what,number objectHandle) (2) number result=simAddObjectToSelection(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

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 simAddParticleObjectItem and simRemoveParticleObject
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 and Vortex linear drag parameter (default: 0.0). Adds a force opposite to the particle velocity (f=v*parameter) 6: Bullet, ODE and Vortex quadratic drag parameter (default: 0.0). Adds a force opposite to the particle velocity (f=v*v*parameter) 7: Bullet, ODE 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 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) 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=simAddParticleObject(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

Description	Adds an item (or clears all items) to a previously inserted particle object. See also simAddParticleObject and simRemoveParticleObject
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=simAddParticleObjectItem(number particleObjectHandle,table itemData)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddPointCloud

Description	Adds a point cloud to be efficiently displayed in the scene. Point clouds created a simulation script will be automatically removed at simulation end. See also simModifyPointCloud and simAddDrawingObject.
C synopsis	simInt simAddPointCloud(simInt pageMask,simInt layerMask,simInt objectHandle,simInt options,simFloat pointSize,simInt ptCnt,const simFloat* pointCoordinates,const simChar* defaultColors,const simChar* pointColors,const simFloat* pointNormals)
C parameters	**pageMask**: a mask indicating in which page the point cloud should be displayed. Set to zero to display the point cloud in all pages. **layerMask**: a mask indicating in which layers the point cloud should be displayed. Set to 255 for a default behaviour. **objectHandle**: the handle of a scene object to which the point cloud should be attached, or -1 if the point cloud is independent (i.e. non-attached, fixed to the world). **options**: bit coded: **bit0 (1)**=the point cloud will be persistent (i.e. not removed at simulation end) **bit1 (2)**=the point cloud will not be visible when seen from vision sensors **bit2 (4)**=if individual point colors are used, then they will be emissive **pointSize**: the size of the points, in pixels. **ptCnt**: the number of points **pointCoordinates**: a pointer to the point coordinates (a succession of x, y, z values). **defaultColors**: a pointer to the default RGB colors for the points (ambient_diffuse RGB, 3 reserved values (set to zero), specular RGB, emissive RGB), where values vary between 0 and 255. Can be NULL. **pointColors**: a pointer to individual ambient_diffuse RGB colors for each point (i.e. 3 values between 0 and 255 for each point). Can be NULL. **pointNormals**: a pointer to individual normal vectors for each point (i.e. 3 values for each point). Can be NULL.
C return value	handle of the point cloud if successful, -1 otherwise.
Lua synopsis	number pointCloudHandle=simAddPointCloud(number pageMask,number layerMask,number objectHandle,number options,number pointSize,table pointCoordinates,table_12 defaultColors=nil,table pointColors=nil,table pointNormals=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddSceneCustomData

Description	Adds or removes custom data to be saved together with a scene. This function is useful for external applications or plugins which want to save their data together with a scene. Use in conjunction with simGetSceneCustomData and simGetSceneCustomDataLength. See also simAddObjectCustomData, simWriteCustomDataBlock and simPersistentDataWrite.
C synopsis	simInt simAddSceneCustomData(simInt header,const simChar* data,simInt dataLength)
C parameters	**header**: identifier for the custom data. If you plan to add custom data (as a company or individual), always use the same header (because only you will know what data type is stored under that header) 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 data which might use the same header as yours. **data**: your custom data. If NULL, the current data under that header will be removed. If you have several items or data types to save, it is your responsibility to pack and code it in data (don't use various headers for each of your items that you want to save with a scene (risk of collision with other developers data (see above))!) **dataLength**: the length of your custom data The data will be copied to an internal buffer, and next time a scene 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=simAddSceneCustomData(number header,string data)
Lua parameters	Same as C-function. (remember that a string in Lua might contain any character, also embedded zeros)
Lua return values	Same as C-Function.

simAddScript

Description	Inserts a new script. Use with care when simulation is running. See also simAssociateScriptWithObject.
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=simAddScript(number scriptType)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAddStatusbarMessage

remote API equivalent: simxAddStatusbarMessage
ROS API equivalent: simRosAddStatusbarMessage

Description	Adds a message to the status bar
C synopsis	simInt simAddStatusbarMessage(const simChar* message)
C parameters	**message**: message
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=simAddStatusbarMessage(string message)
Lua parameters	Same as C-function
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

Description	Adjusts parameters of a view. See also the simFloatingViewAdd and simCameraFitToView 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=simAdjustView(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 simGetSimulationState) 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

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 simAddSceneCustomData or simAddObjectCustomData). 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=simAnnounceSceneContentChange()
Lua parameters	-
Lua return values	Same as C function

simApplyMilling

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 simResetMilling, simHandleMill and simResetMill.
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=simApplyMilling(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAssociateScriptWithObject

Description	Sets the associated object of a child script. Use with care when simulation is running. See also simGetObjectAssociatedWithScript, simAddScript and simSetScriptText.
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=simAssociateScriptWithObject(number scriptHandle,number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simAuxiliaryConsoleClose
remote API equivalent: simxAuxiliaryConsoleClose
ROS API equivalent: simRosAuxiliaryConsoleClose

Description	Closes an auxiliary console window. See also simAuxiliaryConsoleOpen.
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=simAuxiliaryConsoleClose(number consoleHandle)
Lua parameters	Same as C-function.
Lua return values	Same as C-function.

simAuxiliaryConsoleOpen
remote API equivalent: simxAuxiliaryConsoleOpen
ROS API equivalent: simRosAuxiliaryConsoleOpen

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 simAuxiliaryConsolePrint and simAuxiliaryConsoleClose.
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=simAuxiliaryConsoleOpen(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

remote API equivalent: simxAuxiliaryConsolePrint
ROS API equivalent: simRosAuxiliaryConsolePrint

Description	Prints to an auxiliary console window. See also simAuxiliaryConsoleOpen.
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=simAuxiliaryConsolePrint(number consoleHandle,string text)
Lua parameters	Same as C-function.
Lua return values	Same as C-function.

simAuxiliaryConsoleShow

remote API equivalent: simxAuxiliaryConsoleShow
ROS API equivalent: simRosAuxiliaryConsoleShow

Description	Shows or hides an auxiliary console window. See also simAuxiliaryConsoleOpen and simAuxiliaryConsoleClose.
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=simAuxiliaryConsoleShow(number consoleHandle,Boolean showState)
Lua parameters	Same as C-function.
Lua return values	Same as C-function.

simBoolAnd16 (DEPRECATED)

Description	DEPRECATED. See simBoolAnd32 instead.

simBoolAnd32

Description	Performs a 32-bit Boolean AND operation between two numbers. See also simBoolOr32 and simBoolXor32.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result=simBoolAnd32(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

simBoolOr16 (DEPRECATED)

Description	DEPRECATED. See simBoolOr32 instead.

simBoolOr32

Description	Performs a 32-bit Boolean OR operation between two numbers. See also simBoolAnd32 and simBoolXor32.
C synopsis	-

C parameters	-
C return value	-
Lua synopsis	number result=simBoolOr32(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

simBoolXor16 (DEPRECATED)

Description	DEPRECATED. See simBoolXor32 instead.

simBoolXor32

Description	Performs a 32-bit Boolean exclusive-OR operation between two numbers. See also simBoolAnd32 and simBoolOr32.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result=simBoolXor32(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
remote API equivalent: simxBreakForceSensor
ROS API equivalent: simRosBreakForceSensor

Description	Allows breaking a force sensor during simulation. A broken force sensor will lose its positional and orientational constraints. See also simReadForceSensor.
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=simBreakForceSensor(number objectHandle,number desiredBreakState)
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 the same identifier as the header you would use in the simAddSceneCustomData or simAddObjectCustomData function (i.e. 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

Description	Builds an identity transformation matrix
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=simBuildIdentityMatrix()
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

Description	Builds a transformation matrix based on a position vector and Euler angles. See also simBuildMatrixQ.
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=simBuildMatrix(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

Description	Builds a transformation matrix based on a position vector and a quaternion. See also simBuildMatrix.
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=simBuildMatrixQ(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!

simCameraFitToView

Description	Shifts and adjusts a camera associated with a view to fill the view entirely with the specified objects or models. See also the simAdjustView and simFloatingViewAdd 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. **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=simCameraFitToView(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

Description	Checks whether two entities are colliding. Detection is silent (no visual feedback) compared to simHandleCollision. Also, the collidable flags of the entities are overridden if the entities are objects. See also simReadCollision and simCheckCollisionEx.
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=simCheckCollision(number entity1Handle,number entity2Handle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCheckCollisionEx

Description	Checks whether two entities are colliding. This is the extended functionality version of simCheckCollision, and will return all intersections between the two entities. Detection is silent (no visual feedback) compared to simHandleCollision. Also, the collidable flags of the entities are overridden if the entities are objects. See also simReadCollision.
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=simCheckCollisionEx(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

Description	Checks the minimum distance between two entities. Detection is silent (no visual feedback) compared to simHandleDistance. Also, the measurable flags of the entities are overridden if the entities are objects. See also simReadDistance.
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=simCheckDistance(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

Description	Solves a registered IK group, but unlike the simHandleIkGroup function, simCheckIkGroup will not apply the calculated joint values, but instead return them in an array. See also simHandleIkGroup and simFindIkPath.
C synopsis	simInt simHandleIkGroup(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=simCheckIkGroup(number ikGroupHandle,table jointHandles,table jointOptions=nil)
Lua parameters	Similar as C-function
Lua return values	Similar as C-function

simCheckProximitySensor

Description	Checks whether the proximity sensor detects the indicated entity. Detection is silent (no visual feedback) compared to simHandleProximitySensor. Also, the detectable flags of the entity are overridden if the entity is an object. See also simReadProximitySensor and simCheckProximitySensorEx.
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=simCheckProximitySensor(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

Description	Checks whether the proximity sensor detects the indicated entity. This is the extended functionality version of simCheckProximitySensor. Detection is silent (no visual feedback) compared to simHandleProximitySensor. Also, the detectable flags of the entity are overridden if the entity is an object. see also simReadProximitySensor and simCheckProximitySensorEx2.
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=simCheckProximitySensorEx(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

Description	Checks whether the proximity sensor detects the indicated points, segments or triangles. Detection is silent (no visual feedback). See also simReadProximitySensor and simCheckProximitySensorEx.
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=simCheckProximitySensorEx2(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 simCheckProximitySensorEx. (simCheckProximitySensorEx2 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 simCheckProximitySensorEx.

simCheckVisionSensor

Description	Checks whether the vision sensor detects the indicated entity. Detection is silent (no visual feedback) compared to simHandleVisionSensor. Also, the renderable flag of the entity is overridden if the entity is an object. See also simReadVisionSensor and simCheckVisionSensorEx.
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.=simCheckVisionSensor(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

Description	Checks whether the vision sensor detects the indicated entity. This is the extended functionality version of simCheckVisionSensor. Detection is silent (no visual feedback) compared to simHandleVisionSensor. Also, the renderable flag of the entity is overridden if the entity is an object. See also simReadVisionSensor.
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=simCheckVisionSensorEx(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
remote API equivalent: simxClearFloatSignal
ROS API equivalent: simRosClearFloatSignal

Description	Clears a float signal (removes it). See also simSetFloatSignal, simClearIntegerSignal and simClearStringSignal.
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=simClearFloatSignal(string signalName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simClearIntegerSignal
remote API equivalent: simxClearIntegerSignal
ROS API equivalent: simRosClearIntegerSignal

Description	Clears an integer signal (removes it). See also simSetIntegerSignal, simClearFloatSignal and simClearStringSignal.
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=simClearIntegerSignal(string signalName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simClearStringSignal
remote API equivalent: simxClearStringSignal
ROS API equivalent: simRosClearStringSignal

Description	Clears a string signal (removes it). See also simSetStringSignal, simClearIntegerSignal and simClearFloatSignal.
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=simClearStringSignal(string signalName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCloseModule

Description	Releases resources reserved with the simOpenModule 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). simCloseModule is not available in the C-API. Look at the default main script to get an idea about how to use simOpenModule, simHandleModule and simCloseModule.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result=simCloseModule(number sim_handle_all) number result=simCloseModule(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
ROS API equivalent: simRosCloseScene

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 simLoadScene and simSaveScene.
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	-

simConvexDecompose

Description	Calculates the convex decomposition of a shape. See also simUngroupShape.
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): extra points will be added when computing the concavity bit4 set (16): faces points will be added when computing the concavity bit5 set (32): each individual mesh of a grouped shape will be handled on its own during decomposition, otherwise the grouped shape is considered as a single mesh bit6 set (64): convex elements have random colors for easier distinction **intParams**: 5 int values: intParams[0]: the minimum number of clusters to be generated (e.g. 1) intParams[1]: the targeted number of triangles of the decimated mesh (e.g. 500) intParams[2]: the maximum number of vertices for each generated convex hull (e.g. 100) intParams[3]: the maximum number of iterations. Use 0 for the default value (i.e. 4). intParams[4]: reserved. Set to 0. **floatParams**: 5 float values: intParams[0]: the maximum allowed concavity (e.g. 100.0) intParams[1]: the maximum allowed distance to get convex clusters connected (e.g. 30) intParams[2]: the threshold to detect small clusters. The threshold is expressed as a percentage of the total mesh surface (e.g. 0.25) intParams[3]: reserved. Set to 0.0 intParams[4]: reserved. Set to 0.0
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=simConvexDecompose(number shapeHandle,number options,table_4 intParams,table_3 floatParams)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCopyMatrix

Description	Copies a transformation matrix
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=simCopyMatrix(table_12 matrixIn)
Lua parameters	**matrixIn**: matrix to be copied
Lua return values	**matrixOut**: copied matrix, or nil in case of an error

simCopyPasteObjects
remote API equivalent: simxCopyPasteObjects
ROS API equivalent: simRosCopyPasteObjects

Description	Copies and pastes objects, together with all their associated calculation objects and associated scripts. See also simRemoveObject and simRemoveModel.
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=simCopyPasteObjects(table objectHandles,number options)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCopyPasteSelectedObjects (DEPRECATED)

Description	DEPRECATED. See simCopyPasteObjects instead.

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

Description	Creates a new collection. See also simRemoveCollection and simAddObjectToCollection.
C synopsis	simInt simCreateCollection(const simChar* collectionName,simInt options)
C parameters	**collectionName**: the name of the collection. **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=simCreateCollection(string collectionName,number options)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateDummy

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=simCreateDummy(number size,table_12 color=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateForceSensor

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=simCreateForceSensor(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

Description	Creates a heightfield shape. See also simCreatePureShape, simCreateMeshShape and simAddParticleObject.
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=simCreateHeightfieldShape(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

Description	Creates an IK element. See also simCreateIkGroup.
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=simCreateIkElement(number ikGroupHandle,number options,table intParams,table floatParams=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateIkGroup

Description	Creates an IK group. See also simRemoveIkGroup and simCreateIkElement.
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=simCreateIkGroup(number options,table intParams=nil,table floatParams=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateJoint

Description	Creates a joint. See also simSetJointInterval.
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=simCreateJoint(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

Description	Creates a mesh shape. See also simCreatePureShape, simCreateHeightfieldShape and simGetShapeMesh, and see simImportMesh 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=simCreateMeshShape(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

Description	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

simCreatePath

Description	Creates a path object. See also simInsertPathCtrlPoints and simCutPathCtrlPoints.
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=simCreatePath(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

simCreateProximitySensor

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=simCreateProximitySensor(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

Description	Creates a pure primitive shape. See also simCreateMeshShape, simCreateHeightfieldShape and simAddParticleObject.
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=simCreatePureShape(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

simCreateTexture

Description	Creates a planar shape, that will be textured with a new, or imported texture. See also simGetTextureId, simReadTexture, simWriteTexture and simSetShapeTexture.
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=simCreateTexture(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

Description	Creates an empty custom user interface. See also simRemoveUI.
C synopsis	simInt simCreateUI(const simChar* uiName,simInt menuAttributes,const simInt* clientSize,const simInt* cellSize,simInt* buttonHandles)
C parameters	**uiName**: name of the custom user interface. If custom user interface with such a name already exists, a new name is generated and no error produced **menuAttributes**: combination of custom user interface menu attributes. Set to 0 for a menuless custom user interface **clientSize**: sizes (in terms of cells (x and y)) of the client surface. (client surface=surface minus the menu bar space) **cellSize**: cell sizes (x and y). must be a multiple of 2 **buttonHandles**: handles of buttons as specified in the menuAttributes. Array size must be appropriate
C return value	handle of custom user interface if value >=0, -1 if operation was not successful
Lua synopsis	number uiHandle,table buttonHandles=simCreateUI(string uiName,number menuAttributes,table_2 clientSize,table_2 cellSize)
Lua parameters	**uiName**: name of the custom user interface. If a custom user interface with such a name already exists, a new name is generated and no error produced **menuAttributes**: combination of custom user interface menu attributes. Set to 0 for a menuless custom user interface **clientSize**: sizes (in terms of cells (x and y)) of the client surface. (client surface=surface minus the menu bar space) **cellSize**: cell sizes (x and y). must be a multiple of 2
Lua return values	**uiHandle**: handle of the newly created custom user interface if value >=0, error otherwise **buttonHandles**: handles of the created buttons. Is nil if uiHandle is -1.

simCreateUIButton

Description	Creates a button inside of a custom user interface
C synopsis	simInt simCreateUIButton(simInt uiHandle,const simInt* position,const simInt* size,simInt buttonProperty)
C parameters	**uiHandle**: handle to a valid custom user interface **position**: position of the button (in terms of cells (x and y)) **size**: size of the button (in terms of cells (x and y)) **buttonProperty**: button property. Combination of button property values.
C return value	handle of the button, or -1 if operation was not successful
Lua synopsis	number buttonHandle=simCreateUIButton(number uiHandle,table_2 position,table_2 size,number buttonProperty)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateUIButtonArray

Description	Creates an array inside a custom user interface button. Use this to display big arrays instead of creating an array with individual buttons (slower and not memory efficient). The size of the array is the number of cells inside of the button horizontally and vertically. See also simDeleteUIButtonArray and simSetUIButtonArrayColor.
C synopsis	simInt simCreateUIButtonArray(simInt uiHandle,simInt buttonHandle)
C parameters	**uiHandle**: handle of a valid custom user interface **buttonHandle**: handle of a valid button in the specified custom user interface
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=simCreateUIButtonArray(number uiHandle,number buttonHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simCreateVisionSensor

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 ((body passive, body active, volume passive, volume active) 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=simCreateVisionSensor(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

Description	Removes one or several control points from a path object. See also simInsertPathCtrlPoints and simCreatePath.
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=simCutPathCtrlPoints(number pathHandle,number startIndex,number ptCnt)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simDelegateChildScriptExecution (DEPRECATED)

Description	DEPRECATED. Has no effect.

simDeleteSelectedObjects (DEPRECATED)

Description	DEPRECATED. See simRemoveObject instead.

simDeleteUIButtonArray

Description	Deletes (removes) a custom user interface button array previously created with simCreateUIButtonArray
C synopsis	simInt simDeleteUIButtonArray(simInt uiHandle,simInt buttonHandle)
C parameters	**uiHandle**: handle of a valid custom user interface **buttonHandle**: handle of a valid button in the specified custom user interface
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=simDeleteUIButtonArray(number uiHandle,number buttonHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simDisplayDialog
remote API equivalent: simxDisplayDialog
ROS API equivalent: simRosDisplayDialog

Description	Displays a generic dialog box. Use in conjunction with simGetDialogResult ,simGetDialogInput and simEndDialog. 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 simMsgBox and simFileDialog.

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 custom user interface handle. Can be NULL
C return value	handle of generic dialog (different from custom user interface 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=simDisplayDialog(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 custom user interface handle!!), or nil if operation failed **uiHandle**: handle of corresponding custom user interface, 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	-

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

Description	Enables or disables work threads that can accelerate certain calculations by executing tasks in parallel. Currently, work threads support following calculations, and only if they are implicitely handled (i.e. non-explicit handling): collision detections, distance calculations and proximity senso