



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	<p>label: the label to display on the banner</p> <p>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</p> <p>options: a combination of banner options</p> <p>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</p> <p>parentObjectHandle: the handle of a scene object you wish to attach the banner to, or -1 if the banner should be independent.</p> <p>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</p> <p>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</p>
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	<p>objectType: a drawing object type combined with attributes</p> <p>size: size of the item (width of lines or size of points are in pixels, other sizes are in meters)</p> <p>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.</p> <p>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)</p> <p>maxItemCount: maximum number of items this object can hold.</p> <p>ambient_diffuse: default ambient/diffuse color (pointer to 3 rgb values). Can be NULL</p> <p>setToNULL: not used, set to NULL</p> <p>specular: default specular color (pointer to 3 rgb values). Can be NULL</p> <p>emission: default emissive color (pointer to 3 rgb values). Can be NULL</p>
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: <ul style="list-style-type: none"> 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.

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	<p>objectType: a particle object type combined with attributes</p> <p>size: diameter of the particles (spheres)</p> <p>density: density of the particles</p> <p>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:</p> <ul style="list-style-type: none"> 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 ($f=v*\text{parameter}$) 6: Bullet, ODE and Vortex quadratic drag parameter (default: 0.0). Adds a force opposite to the particle velocity ($f=v*v*\text{parameter}$) 7: Bullet, ODE and Vortex linear drag parameter in air ($z>0$) if <code>sim_particle_water</code> was specified (default: 0.0). Adds a force opposite to the particle velocity ($f=v*\text{parameter}$) 8: Bullet, ODE and Vortex quadratic drag parameter in air ($z>0$) if <code>sim_particle_water</code> was specified (default: 0.0). Adds a force opposite to the particle velocity ($f=v*v*\text{parameter}$) 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) <p>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</p> <p>lifeTime: simulation time after which the particles are destroyed. Set to 0.0 for an unlimited lifetime.</p> <p>maxItemCount: the maximum number of particles that this object can hold</p> <p>ambient_diffuse: default ambient/diffuse color (pointer to 3 rgb values). Can be NULL</p> <p>setToNULL: not used, set to NULL</p> <p>specular: default specular color (pointer to 3 rgb values). Can be NULL</p> <p>emission: default emissive color (pointer to 3 rgb values). Can be NULL</p>
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 partide object. See also simAddParticleObject and simRemovePartideObject
C synopsis	simInt simAddParticleObjectItem(simInt objectHandle,const simFloat* itemData)
C parameters	<p>objectHandle: handle of a previously added particle object</p> <p>itemData: data relative to an item. All items (particles) require at least 6 values: \hat{A} 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</p>
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	<p>pageMask: a mask indicating in which page the point cloud should be displayed. Set to zero to display the point cloud in all pages.</p> <p>layerMask: a mask indicating in which layers the point cloud should be displayed. Set to 255 for a default behaviour.</p> <p>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).</p> <p>options: bit coded:</p> <ul style="list-style-type: none"> 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 <p>pointSize: the size of the points, in pixels.</p>

	<p>ptCnt: the number of points</p> <p>pointCoordinates: a pointer to the point coordinates (a succession of x, y, z values).</p> <p>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.</p> <p>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.</p> <p>pointNormals: a pointer to individual normal vectors for each point (i.e. 3 values for each point). Can be NULL.</p>
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	<p>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.</p> <p>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)!))</p> <p>dataLength: the length of your custom data</p> <p>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.</p>
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 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 with plugins. 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	-
Lua parameters	-
Lua return values	-

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 console 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

simBreakForceSensorremote 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 simSendMessage .
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

	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 items 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: 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 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 explicitly 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	<p>sensorHandle: handle of the proximity sensor object</p> <p>vertices: a table containing vertices</p> <p>itemType: 0 for points, 1 for segments and 2 for triangles</p> <p>itemCount: the number of items that the 'vertices' table contains</p> <p>For the other parameters, see the description in simCheckProximitySensorEx. (simCheckProximitySensorEx2 doesn't support occlusion checking)</p>
Lua return values	<p>result: -1 (error), 0 (not detected) or 1 (detected)</p> <p>For the other return values, see the description in simCheckProximitySensorEx.</p>

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	<p>sensorHandle: handle of the vision sensor object</p> <p>entityHandle: handle of entity to detect (object or collection), or sim_handle_all to detect all detectable objects</p> <p>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).</p> <p>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).</p> <p>Usage example:</p> <pre>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); }</pre>
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	<p>sensorHandle: handle of the vision sensor object</p> <p>entityHandle: handle of entity to detect (object or collection), or sim_handle_all to detect all detectable objects</p>
Lua return values	<p>result: -1 if operation was not successful, otherwise 0 (no detection) or 1 (detection)</p> <p>auxiliaryValuePacket1: default auxiliary value packet (same as for the C-function) (table values in Lua are indexed from 1, not 0!)</p> <p>auxiliaryValuePacket2: additional auxiliary value packet (e.g. from a filter component)</p> <p>auxiliaryValuePacket3: etc. (the function returns as many tables as there are auxiliary value packets)</p>

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	<p>sensorHandle: handle of the vision sensor object</p> <p>entityHandle: handle of entity to detect (object or collection), or sim_handle_all to detect all detectable objects</p> <p>returnImage: specifies what should be returned. If true, the sensor's image buffer is returned, otherwise its depth buffer is returned</p>
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)

simClearFloatSignalremote 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

simClearIntegerSignalremote 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

simClearStringSignalremote 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 <code>sim_handle_all</code> argument). <code>simCloseModule</code> is not available in the C-API. Look at the default main script to get an idea about how to use simOpenModule , simHandleModule and <code>simCloseModule</code> .
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.

simCloseSceneremote 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	<p>shapeHandle: handle of the shape to operate on</p> <p>options: bit-coded:</p> <ul style="list-style-type: none"> bit0 set (1): the specified shape will be morphed into its convex decomposition. Otherwise, the convex decomposition will simply 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 <p>intParams: 5 int values:</p> <ul style="list-style-type: none"> 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. <p>floatParams: 5 float values:</p> <ul style="list-style-type: none"> 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	<p>matrixIn: matrix to be copied</p> <p>matrixOut: copy of matrixIn (after the call)</p> <p>matrixIn and matrixOut are pointers to 12 simFloat values (the last row of the 4x4 matrix (0,0,0,1) is not needed)</p> <p>The x-axis of the orientation component is (matrix[0],matrix[4],matrix[8])</p> <p>The y-axis of the orientation component is (matrix[1],matrix[5],matrix[9])</p> <p>The z-axis of the orientation component is (matrix[2],matrix[6],matrix[10])</p> <p>The position component is (matrix[3],matrix[7],matrix[11])</p>
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	<p>objectHandles: array containing the handles of the objects to copy and paste. The same array will receive the copied object handles.</p> <p>objectCount: the number of handles contained in the objectHandles array.</p> <p>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..</p>
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	-

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 responsible 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

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

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	<p>sensorType: the desired proximity sensor type (e.g. sim_proximitysensor_cone_subtype)</p> <p>subType: the desired proximity sensor sub-type (e.g. sim_objectspecialproperty_detectable_ultrasonic)</p> <p>options: bit-coded options: bit 0 set (1): the sensor will be explicitly 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)</p> <p>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</p> <p>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</p> <p>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 zero), specular rgb and emission rgb)). Can be NULL for default values</p>
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	<p>primitiveType: 0 for a cuboid, 1 for a sphere, 2 for a cylinder and 3 for a cone</p> <p>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</p> <p>sizes: 3 values indicating the size of the shape</p> <p>mass: the mass of the shape</p> <p>precision: 2 values that allow specifying the number of sides and faces of a cylinder or sphere. Can be NULL for default values</p>
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	<p>fileName: the filename of the texture to import, or an empty string if you wish to create a new texture.</p> <p>options: bit-coded:</p> <ul style="list-style-type: none"> 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. <p>planeSizes: a pointer to 2 values: the dimensions of the planar shape that will be generated. Can be NULL for default dimensions.</p> <p>scalingUV: a pointer to 2 values: the desired scaling of the texture, along the U and V directions. Can be NULL for default scalings.</p> <p>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.</p> <p>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).</p> <p>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.</p> <p>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.</p> <p>reserved: reserved. Set to NULL.</p>
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	<p>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</p> <p>menuAttributes: combination of custom user interface menu attributes. Set to 0 for a menuless custom user interface</p> <p>clientSize: sizes (in terms of cells (x and y)) of the client surface. (client surface=surface minus the menu bar space)</p> <p>cellSize: cell sizes (x and y). must be a multiple of 2</p> <p>buttonHandles: handles of buttons as specified in the menuAttributes. Array size must be appropriate</p>
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	<p>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</p> <p>menuAttributes: combination of custom user interface menu attributes. Set to 0 for a menuless custom user interface</p> <p>clientSize: sizes (in terms of cells (x and y)) of the client surface. (client surface=surface minus the menu bar space)</p> <p>cellSize: cell sizes (x and y). must be a multiple of 2</p>
Lua return values	<p>uiHandle: handle of the newly created custom user interface if value >=0, error otherwise</p> <p>buttonHandles: handles of the created buttons. Is nil if uiHandle is -1.</p>

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	<p>uiHandle: handle to a valid custom user interface</p> <p>position: position of the button (in terms of cells (x and y))</p> <p>size: size of the button (in terms of cells (x and y))</p> <p>buttonProperty: button property. Combination of button property values.</p>
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	<p>options: bit-coded options:</p> <ul style="list-style-type: none"> bit 0 set (1): the sensor will be explicitly 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) <p>intParams (input): 4 integer parameters:</p> <ul style="list-style-type: none"> intParams[0]: sensor resolution x intParams[1]: sensor resolution y intParams[2]: reserved. Set to 0 intParams[3]: reserved. Set to 0 <p>floatParams (input): 11 floating point parameters:</p> <ul style="list-style-type: none"> 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 <p>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</p>
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

simDisplayDialogremote 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 <code>sim_dlgstyle_input</code> . 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 <code>sim_dlgstyle_input</code> . 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: <code>sim_message_eventcallback_renderingpass</code> , <code>sim_message_eventcallback_opengl</code> , <code>sim_message_eventcallback_openglframe</code> or <code>sim_message_eventcallback_openglcameraview</code> . 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 implicitly handled (i.e. non-explicit handling): collision detections , distance calculations and proximity sensor calculations. The number of work threads can be changed on-the-fly, given that calculations are not underway. This function should only be called from the main script and is not available from the C-API. See also simWaitForWorkThreads .
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number effectiveWorkThreadCount=simEnableWorkThreads(number desiredWorkThreadCount)
Lua parameters	desiredWorkThreadCount: the desired number of work threads. Set to 0 to disable work threads.
Lua return values	effectiveWorkThreadCount: the number of work threads that are running.

simEndDialog

remote API equivalent: **simxEndDialog**
ROS API equivalent: **simRosEndDialog**

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

simExportMesh

Description	Exports a mesh to a file. See also simImportMesh and simGetShapeMesh
C synopsis	simInt simExportMesh(simInt fileformat,const simChar* pathAndFilename,simInt options,simFloat scalingFactor,simInt elementCount,simFloat** vertices,const simInt* verticesSizes,simInt** indices,const simInt* indicesSizes,simFloat** reserved,simChar** names)
C parameters	fileformat: the fileformat to export to. 0 for OBJ format, 1 for DXF format and 4 for BINARY STL format pathAndFilename: the location of the file to create. options: keep at 0 scalingFactor: the scaling factor to apply to the vertices to export vertices: an array to vertice arrays. See the example below verticesSizes: an array indicating the individual vertice array sizes. See the example below indices: an array to indice arrays. See the example below indicesSizes: an array indicating the individual indice array sizes. See the example below reserved: reserved for future extensions. Keep at NULL. names: an array to mesh names. See the example below USAGE EXAMPLE: <pre>// Exports all shapes in the scene simInt shapeCount=0; while (simGetObjects(shapeCount++,sim_object_shape_type)!=-1); shapeCount--; simFloat** vertices=new simFloat*[shapeCount]; simInt* verticesSizes=new simInt[shapeCount]; simInt** indices=new simInt*[shapeCount]; simInt* indicesSizes=new simInt[shapeCount]; simChar** names=new simChar*[shapeCount]; simInt index=0; while (true) { simInt shapeHandle=simGetObjects(index++,sim_object_shape_type); if (shapeHandle<0) break; simFloat* vert; simInt vertS; simInt* ind; simInt indS; simGetShapeMesh(shapeHandle,&vert,&vertS,&ind,&indS,NULL); vertices[index-1]=vert;</pre>

	<pre> verticesSizes[index-1]=vertS; indices[index-1]=ind; indicesSizes[index-1]=indS; names[index-1]=simGetObjectName(shapeHandle); simFloat m[12]; simGetObjectMatrix(shapeHandle,-1,m); for (simInt i=0;i<vertS/3;i++) { simFloat v[3]={vert[3*i+0],vert[3*i+1],vert[3*i+2]}; simTransformVector(m,v); vert[3*i+0]=v[0]; vert[3*i+1]=v[1]; vert[3*i+2]=v[2]; } } simExportMesh(1,"d:\\example.dxf",0,1,shapeCount,vertices, verticesSizes,indices,indicesSizes,NULL,names); for (simInt i=0;i<shapeCount;i++) { simReleaseBuffer((simChar*)vertices[i]); simReleaseBuffer((simChar*)indices[i]); simReleaseBuffer(names[i]); } delete[] vertices; delete[] verticesSizes; delete[] indices; delete[] indicesSizes; delete[] names; </pre>
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=simExportMesh(number fileformat,string pathAndFilename,number options,number scalingFactor,table_of_table vertices,table_of_table indices,nil,table names)
Lua parameters	<p>fileformat: the fileformat to export to. 0 for OBJ format, 1 for DXF format and 4 for BINARY STL format</p> <p>pathAndFilename: the location of the file to create.</p> <p>options: keep at 0</p> <p>scalingFactor: the scaling factor to apply to the vertices to export</p> <p>vertices: a table of vertice tables. See the example below</p> <p>indices: a table of indice tables. See the example below</p> <p>nil: reserved for future extensions.</p> <p>names: a table of mesh names. See the example below</p> <p>USAGE EXAMPLE (e.g. in a customization script):</p> <pre> -- Exports all shapes in the scene if (exportButtonPressed) then allVertices={} allIndices={} allNames={} shapeIndex=0 while (true) do h=simGetObjects(shapeIndex,sim_object_shape_type) if (h<0) then break end shapeIndex=shapeIndex+1 vertices,indices=simGetShapeMesh(h) m=simGetObjectMatrix(h,-1) for i=1,#vertices/3,1 do v={vertices[3*(i-1)+1],vertices[3*(i-1)+2],vertices[3*(i-1)+3]} v=simMultiplyVector(m,v) vertices[3*(i-1)+1]=v[1] vertices[3*(i-1)+2]=v[2] vertices[3*(i-1)+3]=v[3] end table.insert(allVertices,vertices) table.insert(allIndices,indices) table.insert(allNames,simGetObjectName(h)) end if (#allVertices>0) then simExportMesh(1,"d:\\example.dxf",0,1,allVertices,allIndices,nil,allNames) end end </pre>
Lua return values	Same as C-function

simFileDialog

Description	Opens a dialog that allows selecting a file for save or load operations. See also simMsgBox .
C synopsis	simChar* simFileDialog(simInt dlgType,const simChar* title,const simChar* startPath,const simChar* initName,const simChar* extName,const simChar* ext)
C parameters	<p>dlgType: the file dialog type.</p> <p>title: title of the dialog</p> <p>startPath: the initial path. Indicate an empty string for the path to V-REP's application</p> <p>initName: the initial name. Can be an empty string</p> <p>extName: the extension name, e.g. "text file"</p> <p>ext: the extension, e.g. ".txt"</p>
C return value	pointer to a string representing the selected file name and path. The user is in charge of releasing the

	buffer with simReleaseBuffer .
Lua synopsis	string pathAndName=simFileDialog(number mode,string title,string startPath,string initName,string extName,string ext)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simFindIkPath

Description	Searches for a collision-free path leading a serial manipulator from a start configuration to a goal pose (i.e. via a straight line in the Cartesian space). The function relies on various settings and parameters of V-REP's motion planning functionality . The first time this function is called on the given motion planning object, a preprocessing stage will prepare a calculation structure. This might take a few seconds depending mainly on the number of phase1 nodes. See also simFindMpPath , simGetMpConfigTransition and simCheckIkGroup .
C synopsis	simFloat* simFindIkPath(simInt motionPlanningObjectHandle,const simFloat* startConfig,const simFloat* goalPose,simInt options,simFloat stepSize,simInt* outputConfigsCnt,simFloat* reserved,const simInt* auxIntParams,const simFloat* auxFloatParams)
C parameters	<p>motionPlanningObjectHandle: the handle of a motion planning object. Refer to simGetMotionPlanningHandle.</p> <p>startConfig: the start or initial configuration of the robot (i.e. its joint positions). Should contain x values where x is the number of DoFs of the specified motion planning task.</p> <p>goalPose: a pointer to 7 values specifying the position and orientation of the desired end-effector pose. First 3 values represent the position, final 4 values the orientation as a quaternion (x,y,z,w).</p> <p>options: bit-coded:</p> <ul style="list-style-type: none"> bit0 (1): not used. Keep unset.. bit1 (2): not used. Keep unset. bit2 (4): if set, then the found path will be visualized in yellow. bit3 (8): if set, then some information will be output to the console. bit4 (16): if set, then robot self-interferences will be ignored and calculations can drastically be sped-up. bit5 (32): if set, then robot-environment interferences will be ignored and calculations can drastically be sped-up. bit6 (64): not used. Keep unset. bit7 (128): if set, then the returned Cartesian space distances will be projected distanced (projected into the IK constraint subspace). This option is usually used when the associated IK element has not all possible positional/orientational constraints turned on. bit8 (256): if set, then the returned Cartesian space distances will ignore the orientational distance component. <p>stepSize: the maximum Cartesian space distance between individual collision-free phase2 nodes. The distance calculation will use the Cartesian space metric specified in the motion planning properties.</p> <p>outputConfigsCnt: a pointer to an integer receiving the number of returned configurations. If a single configuration is returned, this means that the specified start and goal configurations are coincident.</p> <p>reserved: reserved. Keep NULL.</p> <p>auxIntParams: reserved. Keep NULL.</p> <p>auxFloatParams: reserved. Keep NULL.</p>
C return value	<p>NULL in case of an error, or when the search failed. Otherwise a buffer of float values that the user is in charge of releasing with simReleaseBuffer. The returned buffer contains:</p> <ul style="list-style-type: none"> the found path (x*n values): n configurations with each x values (x is the number of DoFs of the specified motion planning task). The configurations will include the start and the goal configuration, except when start and goal are coincident, in which case a single configuration is returned. the configuration space distances (n values): for each returned configuration, a distance to the start configuration (following the path). The last of the n values represents the length of the found path in the configuration space. The distance is calculated using the weight specified for each joint in the motion planning properties. the end-effector positions (3*n values): for each returned configuration, the position of the corresponding end-effector (x, y, z). the end-effector quaternions (4*n values): for each returned configuration, the quaternion of the corresponding end-effector (x, y, z, w). the Cartesian space distances (n values): for each returned configuration, a distance to the start pose (following the path). The last of the n values represents the length of the found path in the Cartesian space. The distance is calculated using the Cartesian space metric specified in the motion planning properties.
Lua synopsis	table path,table confSpaceDist,table tipPositions,table tipQuaternions,table cartesianSpaceDist=simFindIkPath(number motionPlanningObjectHandle,table startConfig,table _7 goalPose,number options,number stepSize,table auxIntParams=nil,table auxFloatParams=nil)
Lua parameters	Similar as C-function
Lua return values	Similar as C-function

simFindMpPath

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

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

simFloatingViewAdd

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

simFloatingViewRemove

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

simFollowPath

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

simGetArrayParameter

remote API equivalent: [simxGetArrayParameter](#)
ROS API equivalent: [simRosGetArrayParameter](#)

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

simGetBooleanParameter

remote API equivalent: [simxGetBooleanParameter](#)
ROS API equivalent: [simRosGetBooleanParameter](#)

Description	Retrieves a boolean value. See the Boolean parameter identifiers . See also simSetBooleanParameter , simGetIntegerParameter , simGetFloatingParameter , simGetArrayParameter and simGetStringParameter .
C synopsis	simInt simGetBooleanParameter(simInt parameter)
C parameters	parameter : boolean parameter identifier
C return value	value of the parameter (0 or 1) or -1 in case of an error
Lua synopsis	boolean boolState=simGetBooleanParameter(number parameter)
Lua parameters	Same as C-function
Lua return values	boolState : value of the Boolean parameter, or nil in case of an error

simGetClosestPositionOnPath

Description	Retrieves the intrinsic relative position on a path that is closest to the specified point. The returned value is dependent on the selected path length calculation method for the given path object. See also
-------------	--

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

simGetCollectionHandle

Description	Retrieves a collection handle based on its name. The operation of this function depends on the current name suffix settings (see simGetNameSuffix , simSetNameSuffix , and the section on accessing general-type objects). See also simIsHandleValid .
C synopsis	simInt simGetCollectionHandle(const simChar* collectionName)
C parameters	collectionName : name of the collection
C return value	Handle of the collection, or -1 in case of an error
Lua synopsis	number collectionHandle=simGetCollectionHandle(string collectionName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetCollectionName

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

simGetCollectionObjects

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

simGetCollisionHandle

remote API equivalent: [simxGetCollisionHandle](#)
ROS API equivalent: [simRosGetCollisionHandle](#)

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

simGetConfigurationTree

Description	Retrieves configuration information for a hierarchy tree (object relative positions/orientations, joint/path
-------------	--

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

simGetContactInfo

Description	Retrieves contact point information of a dynamic simulation pass.
C synopsis	simInt simGetContactInfo(simInt dynamicPass,simInt objectHandle,simInt index,simInt* objectHandles,simFloat* contactInfo)
C parameters	dynamicPass : a specific dynamic sub-step index or sim_handle_all. By default a call to simHandleDynamics executes the dynamics engine x times, with x times smaller time steps (where x is a parameter that can be adjusted). At each of those sub-steps, contacts are created and destroyed. With the dynamicPass argument you can select which sub-step you wish to retrieve contacts from (zero-based index), or sim_handle_all to retrieve the contacts of all sub-steps. See also simGetIntegerParameter (sim_intparam_dynamic_step_divider). objectHandle : handle of a specific object you wish to retrieve contacts from, or sim_handle_all to retrieve all contacts in the scene. index : zero-based index of the contact to retrieve. objectHandles : handles of the two objects contacting. The handles might also refer to particle objects that are not treated as regular scene objects. contactInfo : pointer to 6 values, where the 3 first values represent the contact position, and the 3 last values represent the force generated by the contact.
C return value	-1 in case of an error, 0 if no contact was found at the given index or 1 if a contact was returned.
Lua synopsis	table_2 collidingObjects,table_3 collisionPoint,table_3 reactionForce=simGetContactInfo(number dynamicPass,number objectHandle,number index)
Lua parameters	Same as C-function
Lua return values	collidingObjects : handles of the two objects contacting. The handles might also refer to particle objects that are not treated as regular scene objects. collisionPoint : coordinates of the contact reactionForce : vector that represents the force generated by the contact

simGetCustomizationScriptAssociatedWithObject

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

simGetDataOnPath

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

Lua synopsis	number auxFlags,table_4 auxChannels=simGetDataOnPath(number pathHandle,number relativeDistance)
Lua parameters	Same as C-function
Lua return values	Similar as C-function

simGetDialogInput

remote API equivalent: [simxGetDialogInput](#)
 ROS API equivalent: [simRosGetDialogInput](#)

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

simGetDialogResult

remote API equivalent: [simxGetDialogResult](#)
 ROS API equivalent: [simRosGetDialogResult](#)

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

simGetDistanceHandle

remote API equivalent: [simxGetDistanceHandle](#)
 ROS API equivalent: [simRosGetDistanceHandle](#)

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

simGetEulerAnglesFromMatrix

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

simGetExplicitHandling

Description	Retrieves the explicit handling flags for a general object. See also simSetExplicitHandling .
-------------	---

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

simGetFloatingParameterremote API equivalent: [simxGetFloatingParameter](#)ROS API equivalent: [simRosGetFloatingParameter](#)

Description	Retrieves a floating point value. See the floating-point parameter identifiers . See also simSetFloatingParameter , simGetBooleanParameter , simGetIntegerParameter , simGetArrayParameter and simGetStringParameter .
C synopsis	simInt simGetFloatingParameter(simInt parameter,simFloat* floatState)
C parameters	parameter: floating parameter identifier floatState: value of the parameter
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number parameterValue=simGetFloatingParameter(number parameter)
Lua parameters	parameter: parameter identifier (sim_floatparam_...)
Lua return values	parameterValue: value of the parameter or nil in case of an error

simGetFloatSignalremote API equivalent: [simxGetFloatSignal](#)ROS API equivalent: [simRosGetFloatSignal](#)

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

simGetIkGroupHandle

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

simGetIkGroupMatrix

Description	Retrieves various information from an IK group, such as the Jacobian. See also simGetIkGroupHandle .
C synopsis	simFloat* simGetIkGroupMatrix(simInt ikGroupHandle,simInt options,simInt* matrixSize)
C parameters	ikGroupHandle: handle of an IK group options: a value indicating what kind of information to retrieve: 0: to retrieve the last calculated Jacobian. matrixSize[0] is the number of rows (the number of DoFs), matrixSize[1] is the number of columns. See further down how to extract the Jacobian. 1: to retrieve the manipulability value, i.e. $\sqrt{\det(J*JT)}$, where J is the Jacobian, and JT the Jacobian transpose. The returned pointer points onto a single value. matrixSize: the dimensions of the returned matrix (x=rows, y=columns) USAGE EXAMPLE (to correctly interpret the Jacobian): <pre>float jacobianSize[2]; float* jacobian=simGetIkGroupMatrix(ikGroupHandle,0,jacobianSize); // jacobianSize[0] represents the row count of the Jacobian // (i.e. the number of equations or the number of joints involved</pre>

	<pre>// in the IK resolution of the given kinematic chain) // Joints appear from tip to base. // jacobianSize[1] represents the column count of the Jacobian // (i.e. the number of constraints, e.g. x,y,z,alpha,beta,gamma, jointDependency) // The Jacobian data is ordered row-wise, i.e.: // [row0,col0],[row1,col0],...,[rowN,col0],[row0,col1],[row1,col1],etc. for (int i=0;i<jacobianSize[0];i++) { std::string str; for (int j=0;j<jacobianSize[1];j++) { if (str.size()==0) str+=", "; str+=boost::str(boost::format("%.1e") % jacobian[j*jacobianSize[0]+i]); } printf(str.c_str()); } </pre>
C return value	A pointer to x*y float values. The values should be copied since the pointer might not remain valid.
Lua synopsis	table matrix,table_2 matrixSize=simGetIkGroupMatrix(number ikGroupHandle,number options)
Lua parameters	Same as C-function
	<p>USAGE EXAMPLE (to correctly interpret the Jacobian):</p> <pre>jacobian,jacobianSize=simGetIkGroupMatrix(ikGroupHandle,0) -- jacobianSize[1] represents the row count of the Jacobian -- (i.e. the number of equations or the number of joints involved -- in the IK resolution of the given kinematic chain) -- Joints appear from tip to base. -- jacobianSize[2] represents the column count of the Jacobian -- (i.e. the number of constraints, e.g. x,y,z,alpha,beta,gamma, jointDependency,etc.) -- The Jacobian data is ordered row-wise, i.e.: -- [row1,col1],[row2,col1],...,[rowN,col1],[row1,col2],[row2,col2],etc. for i=1,jacobianSize[1],1 do str='' for j=1,jacobianSize[2],1 do if #str~=0 then str=str..' ', ' end str=str..string.format("%.1e",jacobian[(j-1)*jacobianSize[1]+i]) end print(str) end </pre>
Lua return values	Same as C-function

simGetIntegerParameterremote API equivalent: **simxGetIntegerParameter**ROS API equivalent: **simRosGetIntegerParameter**

Description	Retrieves an integer value. See the integer parameter identifiers . See also simSetIntegerParameter , simGetBooleanParameter , simGetFloatingParameter , simGetArrayParameter and simGetStringParameter .
C synopsis	simInt simGetIntegerParameter(simInt parameter,simInt* intState)
C parameters	parameter : integer parameter identifier intState : value of the parameter
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number parameterValue=simGetIntegerParameter(number parameter)
Lua parameters	parameter : parameter identifier (sim_intparam...)
Lua return values	parameterValue : value of the parameter or nil in case of an error

simGetIntegerSignalremote API equivalent: **simxGetIntegerSignal**ROS API equivalent: **simRosGetIntegerSignal**

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

simGetInvertedMatrix

Description	Returns the inverse of a transformation matrix
C synopsis	See simInvertMatrix for the C-equivalent function
C parameters	-
C return value	-
Lua synopsis	table_12 invertedMatrix=simGetInvertedMatrix(table_12 matrix)
Lua parameters	matrix: table to 12 numbers (the last row of the 4x4 matrix (0,0,0,1) is not needed) The x-axis of the orientation component is (matrix[1],matrix[5],matrix[9]) The y-axis of the orientation component is (matrix[2],matrix[6],matrix[10]) The z-axis of the orientation component is (matrix[3],matrix[7],matrix[11]) The position component is (matrix[4],matrix[8],matrix[12])
Lua return values	invertedMatrix: inverse of the input matrix, or nil in case of an error

simGetJointForceremote API equivalent: [simxGetJointForce](#)ROS API equivalent: [simRosGetJointState](#)

Description	Retrieves the force or torque applied to a joint along/about its active axis. This function retrieves meaningful information only if the joint is prismatic or revolute, and is dynamically enabled. With the Bullet engine, this function returns the force or torque applied to the joint motor (torques from joint limits are not taken into account). With the ODE and Vortex engine, this function returns the total force or torque applied to a joint along/about its z-axis. See also simSetJointForce and simReadForceSensor .
C synopsis	simInt simGetJointForce(simInt jointHandle,simFloat* forceOrTorque)
C parameters	jointHandle: handle of the joint forceOrTorque: the force or the torque applied to the joint along/about its z-axis.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number forceOrTorque=simGetJointForce(number jointHandle)
Lua parameters	Same as C-function
Lua return values	forceOrTorque: the force or the torque applied to the joint along/about its z-axis.

simGetJointInterval

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

simGetJointMatrixremote API equivalent: [simxGetJointMatrix](#)ROS API equivalent: [simRosGetJointMatrix](#)

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

simGetJointMode

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

simGetJointPosition

remote API equivalent: [simxGetJointPosition](#)
ROS API equivalent: [simRosGetJointState](#)

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

simGetJointTargetPosition

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

simGetJointTargetVelocity

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

simGetJointType

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

simGetLastError

remote API equivalent: [simxGetLastErrors](#)
ROS API equivalent: [simRosGetLastErrors](#)

Description	Retrieves the last generated error message. By calling this function, the last error message is reset and a subsequent call to this function returns NULL. Errors are memorized on a thread-basis (e.g. threads originating from threaded scripts have each an individual error handler). See also simSetLastError , the sim_intparam_error_report_mode and the error report modes .
C synopsis	simChar* simGetLastError()
C parameters	None
C return value	Error message buffer or NULL if no error message is present. The user has to delete the returned buffer with a call to simReleaseBuffer
Lua synopsis	string lastError=simGetLastError()
Lua parameters	Same as C-function
Lua return values	Same as C-function (but nil instead of NULL, and simReleaseBuffer does not need to be called)

simGetLightParameters

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

simGetLinkDummy

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

simGetMainWindow

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

simGetMaterialId

Description	Retrieves the material ID of a specific material. See also simGetShapeMaterial and simSetShapeMaterial .
C synopsis	simInt simGetMaterialId(const simChar* materialName)
C parameters	materialName : the name of the material ID to be retrieved.
C return value	The material ID, or -1 if the material does not exist or in case of an error
Lua synopsis	number materialId=simGetMaterialId(string materialName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetMechanismHandle

Description	Retrieves the handle of a mechanism to be solved by the geometric constraint solver. The operation of this function depends on the current name suffix settings (see simGetNameSuffix , simSetNameSuffix , and the section on accessing general-type objects). See also simIsValidHandle .
-------------	---

C synopsis	simInt simGetMechanismHandle(const simChar* mechanismName)
C parameters	mechanismName : name of the mechanism
C return value	Handle of the mechanism if operation was successful, -1 otherwise
Lua synopsis	number mechanismHandle=simGetMechanismHandle(string mechanismName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetModelProperty

remote API equivalent: [simxGetModelProperty](#)
 ROS API equivalent: [simRosGetModelProperty](#)

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

simGetModuleName

Description	Retrieves a plugin name that was previously registered with simLoadModule . The simulator normally automatically loads and registers plugins present in the application directory. Users can use the simGetModuleName to verify if a specific module is present
C synopsis	simChar* simGetModuleName(simInt index,sumUChar* moduleVersion)
C parameters	index : index to a module. To list-up all module names, start with index=0 and increment index until return value is NULL moduleVersion : version of the plugin. Can be NULL.
C return value	Name of the module or NULL if no module is available at index position, or in case of an error. The user is in charge of destroying the returned name with simReleaseBuffer
Lua synopsis	string moduleName,number moduleVersion=simGetModuleName(number index)
Lua parameters	index : index to a module. To list-up all module names, start with index=0 and increment index until return value is nil
Lua return values	moduleName : name of the module or nil if no module is available at that index position, or in case of an error. moduleVersion : version of the plugin, or nil if moduleName is also nil

simGetMotionPlanningHandle

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

simGetMpConfigForTipPose

Description	Searches for a serial manipulator configuration that matches a given end-effector position/orientation in space. The function uses V-REP's motion planning functionality . The first time this function is called on the given motion planning object, a preprocessing stage will prepare a calculation structure (i.e. nodes). This might take a few seconds depending mainly on the number of phase1 nodes. See also simFindMpPath .
C synopsis	simInt simGetMpConfigForTipPose(simInt motionPlanningObjectHandle,simInt options,simFloat closeNodesDistance,simInt trialCount,const simFloat* tipPose,simInt maxTimeInMs,simFloat* outputJointPositions,const simFloat* referenceConfigs,simInt referenceConfigCount,const simFloat* jointWeights,const simInt* jointBehaviour,simInt correctionPasses)
C parameters	motionPlanningObjectHandle : the handle of a motion planning object. Refer to simGetMotionPlanningHandle . options : bit-coded: bit0: if set (1), we want to take advantage of previous calls and take into account phase1 nodes we already tried in a previous call. Otherwise, we start a fresh search. bit1: if set (2), we favor configurations that are far away from the provided referenceConfigs (in that case, all provided reference configurations are taken into account). Otherwise, we favor configurations that are close to the provided referenceConfigs (in that case, only the first provided reference configuration is taken into account). bit2: if set (4), we enable a precise mode that will use IK in an earlier search stage. This

	<p>usually produces closer matches with a given reference configuration (referenceConfigs), but is also slower.</p> <p>bit3: if set (8), some information will be output to the console.</p> <p>bit4: if set (16), then robot self interferences will be ignored.</p> <p>bit5: if set (32), then robot-environment interferences will be ignored and calculations can be sped-up.</p> <p>closeNodeDistance: a distance that allows selecting a smaller selection of phase1 nodes, that will then be searched (other nodes will be ignored). If bit0 of options is set, this value is not used. Distance between 2 end-effector poses is calculated using the Cartesian space metric.</p> <p>trialCount: the maximum number of matching trials that will be performed before giving up and returning 0. A trial usually involves selecting a dose phase1 node, and trying from there to reach the desired end-effector pose via IK.</p> <p>tipPose: a pointer to 7 values specifying the position and orientation of the desired end-effector pose. First 3 values represent the position, final 4 values the orientation as a quaternion (x,y,z,w).</p> <p>maxTimeInMs: the maximum time in milliseconds after which the search operation is aborted. Specify 0 for an infinite time.</p> <p>outputJointPositions: a pointer to x joint values, representing the found robot configuration. The user is in charge of allocating this array with x values, where x has to match the DoFs of the defined motion planning task.</p> <p>referenceConfigs: one or several reference configurations, that can be used to refine the desired results (refer also to bit1 of options). referenceConfigs should point onto x*n values, where n is the number of configurations provided, and x the DoFs of the defined motion planning task. Can be NULL.</p> <p>referenceConfigCount: the number of provided reference configurations, n. Can be 0.</p> <p>jointWeights: a pointer to x values: For each joint in the robot, there is a weight that tells how important the joint is in the closest/farthest configuration calculations (see bit1 of options). Normally one would use 1.0f for every joint. But if for instance we want to find configurations that have their first joint close to the specified reference configuration (referenceConfigs), then we would use 0.0f for all entries, except for the first one. Can be NULL for default behaviour (i.e. same as an array filled with 1.0f)</p> <p>jointBehaviour: when trying to find configurations that are close to a provided reference configuration (referenceConfigs) we can specify here how each joint should behave during the IK phase. -1 will lock the joint in the desired position (i.e. the joint will not be involved in the IK), which only makes sense with redundant mechanisms. A positive number indicates the amount in percent (per correctionPasses) the given joint will be corrected towards its desired position. Can be NULL for default behaviour.</p> <p>correctionPasses: the number of correction passes for the jointBehaviour functionality. Can be 0 for default behaviour.</p>
C return value	0 if the calculation failed, otherwise the number of trials before success.
Lua synopsis	number trialsBeforeSuccess,table jointPositions=simGetMpConfigForTipPose(number motionPlanningObjectHandle,number options,number closeNodesDistance,number trialCount,table_7 tipPose,number maxTimeInMs=0,table referenceConfigs=nil,table jointWeights=nil,table jointBehaviour=nil,number correctionPasses=0)
Lua parameters	Similar as C-function
Lua return values	Similar as C-function

simGetMpConfigTransition

Description	<p>Searches for a collision-free path leading a serial manipulator from a start configuration to a goal configuration, by trying to follow a predefined path in the Cartesian space. This is useful for redundant manipulators, but also for safely driving non-redundant manipulators via IK through a singular configuration. The function uses V-REP's motion planning functionality. The first time this function is called on the given motion planning object, a preprocessing stage will prepare a calculation structure. This might take a few seconds depending mainly on the number of phase1 nodes. See also simGetMpConfigForTipPose, simFindMpPath, simFindIkPath and simCheckIkGroup.</p>
C synopsis	<pre>simFloat* simGetMpConfigTransition(simInt motionPlanningObjectHandle,const simFloat* startConfig,const simFloat* goalConfig,simInt options,const simInt* select,simFloat calcStepSize,simFloat maxOutStepSize,simInt wayPointCnt,const simFloat* wayPoints,simInt* outputConfigsCnt,const simInt* auxIntParams,const simFloat* auxFloatParams)</pre>
C parameters	<p>motionPlanningObjectHandle: the handle of a motion planning object. Refer to simGetMotionPlanningHandle.</p> <p>startConfig: the start or initial configuration of the robot (i.e. its joint positions). Should contain x values where x is the number of DoFs of the specified motion planning task.</p> <p>goalConfig: the goal configuration of the robot (i.e. its joint positions). Should contain x values where x is the number of DoFs of the specified motion planning task. You can use simGetMpConfigForTipPose if the goal configuration is not known.</p> <p>options: bit-coded:</p> <ul style="list-style-type: none"> bit0 (1): reserved. Keep unset. bit1 (2): reserved. Keep unset. bit2: if set (4), then the found path will be visualized in yellow. bit3: if set (8), then some information will be output to the console. bit4: if set (16), then robot self-interferences will be ignored and calculations can drastically be sped-up. bit5: if set (32), then robot-environment interferences will be ignored and calculations can drastically be sped-up. bit6 (64): reserved. Keep unset. bit7 (128): reserved. Keep unset. bit8: if set (256), then the returned Cartesian space distances will ignore the orientational distance component. bit9: if set (512), then the specified waypoints will be followed by the end-effector in the Cartesian space. For that to happen, IK will be used too. If no waypoints are specified, then the end-effector will link the start to the goal configuration via a straight path in the Cartesian

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

simGetNameSuffix

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

simGetNavigationMode

Description	Retrieves the navigation and selection mode for the mouse. See also simSetNavigationMode .
C synopsis	simInt simGetNavigationMode()

C parameters	None
C return value	navigation mode if operation was successful, -1 otherwise
Lua synopsis	number navigationMode=simGetNavigationMode()
Lua parameters	Same as C-function
Lua return values	Same as C-function. See also the boolean operators in Lua .

simGetObjectAssociatedWithScript

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

simGetObjectChild

remote API equivalent: [simxGetObjectChild](#)

ROS API equivalent: [simRosGetObjectChild](#)

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

simGetObjectConfiguration

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

simGetObjectCustomData

Description	Retrieves an object's custom data previously added with the simAddObjectCustomData function. See also simReadCustomDataBlock .
C synopsis	simInt simGetObjectCustomData(simInt objectHandle,simInt header,simChar* data)
C parameters	objectHandle : handle of the object header : identifier of the custom data. See simAddObjectCustomData for details. data : pointer to a data buffer allocated by the user. Use simGetObjectCustomDataLength to adjust the buffer size.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	string data=simGetObjectCustomData(number objectHandle,number header)
Lua parameters	Same as C-function
Lua return values	nil if operation was not successful. Otherwise the custom data (might be an empty string if custom data doesn't exist).

simGetObjectCustomDataLength

Description	Retrieves the length of an object's custom data previously added with the simAddObjectCustomData function. See also simGetObjectCustomData .
C synopsis	simInt simGetObjectCustomDataLength(simInt objectHandle,simInt header)
C parameters	objectHandle : handle of the object

	header: identifier of the custom data. See simAddObjectCustomData for details.
C return value	length of the custom data, 0 if there is no data under that header, or -1 in case of an error
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetObjectFloatParameter
 remote API equivalent: [simxGetObjectFloatParameter](#)
 ROS API equivalent: [simRosGetObjectFloatParameter](#)

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

simGetObjectHandle
 remote API equivalent: [simxGetObjectHandle](#)
 ROS API equivalent: [simRosGetObjectHandle](#)

Description	Retrieves an object handle based on its name. The operation of this function depends on the current name suffix settings (see simGetNameSuffix , simSetNameSuffix , and the section on accessing general-type objects). See also simIsHandleValid and simGetObjectUniqueIdentifier .
C synopsis	simInt simGetObjectHandle(const simChar* objectName)
C parameters	objectName: name of object
C return value	handle of object or -1 if operation was not successful
Lua synopsis	number objectHandle=simGetObjectHandle(string objectName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetObjectIntParameter
 remote API equivalent: [simxGetObjectIntParameter](#)
 ROS API equivalent: [simRosGetObjectIntParameter](#)

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

simGetObjectLastSelection

Description	Returns the handle of the last selected object. See also simGetObjectSelection and simGetObjectSelectionSize .
C synopsis	simInt simGetObjectLastSelection()
C parameters	None
C return value	last selected object handle or -1 if selection is empty or operation was not successful

Lua synopsis	number objectHandle=simGetObjectLastSelection()
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetObjectMatrix

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

simGetObjectName

Description	Retrieves the name of an object based on its handle. See also simSetObjectName .
C synopsis	simChar* simGetObjectName(simInt objectHandle)
C parameters	objectHandle: handle of the object
C return value	Name of the object if operation was successful, NULL otherwise. The user is in charge of destroying the returned buffer with simReleaseBuffer
Lua synopsis	string objectName=simGetObjectName(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function (but nil instead of NULL, and simReleaseBuffer does not need to be called)

simGetObjectOrientation

remote API equivalent: [simxGetObjectOrientation](#)
ROS API equivalent: [simRosGetObjectPose](#)

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

simGetObjectParent

remote API equivalent: [simxGetObjectParent](#)
ROS API equivalent: [simRosGetObjectParent](#)

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

simGetObjectPosition

remote API equivalent: [simxGetObjectPosition](#)
ROS API equivalent: [simRosGetObjectPose](#)

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

simGetObjectProperty

Description	Retrieves the main properties of a scene object. See also simSetObjectProperty .
C synopsis	simInt simGetObjectProperty(simInt objectHandle)
C parameters	objectHandle : handle of the object
C return value	object property values , -1 if operation was not successful
Lua synopsis	number property=simGetObjectProperty(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function. See also the boolean operators in Lua .

simGetObjectQuaternion (ROS API equivalent: [simRosGetObjectPose](#))

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

simGetObjects

remote API equivalent: [simxGetObjects](#)
ROS API equivalent: [simRosGetObjects](#)

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

simGetObjectSelection

remote API equivalent: [simxGetObjectSelection](#)
ROS API equivalent: [simRosGetObjectSelection](#)

Description	Retrieves all selected object's handles. See also simGetObjectSelectionSize , simAddObjectToSelection and simRemoveObjectFromSelection .
C synopsis	simInt simGetObjectSelection(simInt* objectHandles)
C parameters	objectHandles : pointer to object handles. Make sure to have at least simGetObjectSelectionSize simInts available

C return value	size of the selection (>=0) if operation was successful, -1 otherwise
Lua synopsis	table selectedObjectHandles=simGetObjectSelection()
Lua parameters	None
Lua return values	selectedObjectHandles: table containing the handles of all selected objects, or nil if no object is selected or in case of an error

simGetObjectSelectionSize

Description	Retrieves the size of the object selection. See also simGetObjectSelection .
C synopsis	simInt simGetObjectSelectionSize()
C parameters	None
C return value	size of the selection, or -1 if operation was not successful
Lua synopsis	number selectionSize=simGetObjectSelectionSize()
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetObjectsInTree

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

simGetObjectSizeFactor

Description	Retrieves the size factor of a scene object. The size factor is different from the real object size. Use this to be able to react to scaling operations. See also simGetObjectSizeValues .
C synopsis	simFloat simGetObjectSizeFactor(simInt objectHandle)
C parameters	objectHandle: handle of the scene object
C return value	size factor or negative value in case of an error
Lua synopsis	number sizeFactor=simGetObjectSizeFactor(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetObjectSizeValues

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

simGetObjectSpecialProperty

Description	Retrieves the special properties of a scene object. See also simSetObjectSpecialProperty .
C synopsis	simInt simGetObjectSpecialProperty(simInt objectHandle)
C parameters	objectHandle: handle of the object
C return value	object special property values , -1 if operation was not successful
Lua synopsis	number property=simGetObjectSpecialProperty(number objectHandle)
Lua parameters	Same as C-function

Lua return values	Same as C-function. See also the boolean operators in Lua .
-------------------	---

simGetObjectStringParameter

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

simGetObjectType

Description	Retrieves the type of an object
C synopsis	simInt simGetObjectType(simInt objectHandle)
C parameters	objectHandle : handle of the object
C return value	type of the object (sim_object_shape_type, sim_object_joint_type, etc. (see the object types) or -1 in case of error
Lua synopsis	number objectType=simGetObjectType(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetObjectUniqueIdentifier

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

simGetObjectVelocity

remote API equivalent: [simxGetObjectVelocity](#)
ROS API equivalent: [simros_strmcmd_get_twist_status](#)

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

simGetOrientationOnPath

Description	Retrieves the absolute interpolated orientation of a point along a path object. See also simGetPositionOnPath , simGetPathPosition and simGetClosestPositionOnPath .
-------------	--

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

simGetPage

Description	Retrieves the current page index (view). See also simSetPage .
C synopsis	simInt simGetPage()
C parameters	None
C return value	page index or -1 in case of an error
Lua synopsis	number pageIndex=simGetPage()
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetPathLength

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

simGetPathPlanningHandle

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

simGetPathPosition

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

simGetPositionOnPath

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

simGetQuaternionFromMatrix

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

simGetRealTimeSimulation

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

simGetRotationAxis

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

simGetSceneCustomData

Description	Retrieves the scene custom data previously added with the simAddSceneCustomData function. See also
-------------	--

	simGetSceneCustomDataLength , simAddSceneCustomData , simReadCustomDataBlock and simPersistentDataRead .
C synopsis	simInt simGetSceneCustomData(simInt header,simChar* data)
C parameters	header : identifier of the custom data. See simAddSceneCustomData for details. data : pointer to a data buffer allocated by the user. Use simGetSceneCustomDataLength to adjust the buffer size.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	string data=simGetSceneCustomData(number header)
Lua parameters	Same as C-function.
Lua return values	nil if operation was not successful. Otherwise the custom data (might be an empty string if custom data doesn't exist).

simGetSceneCustomDataLength

Description	Retrieves the length of the scene custom data previously added with the simAddSceneCustomData function.
C synopsis	simInt simGetSceneCustomDataLength(simInt header)
C parameters	header : identifier of the custom data. See simAddSceneCustomData for details.
C return value	length of the custom data, 0 if there is no data under that header, or -1 in case of an error
Lua synopsis	-
Lua parameters	-
Lua return values	-

simGetScript

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

simGetScriptAssociatedWithObject

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

simGetScriptAttribute

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

simGetScriptExecutionCount

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

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

simGetScriptHandle

Description	Retrieves the handle of a script. The operation of this function depends on the current name suffix settings (see simGetNameSuffix , simSetNameSuffix , and the section on accessing general-type objects). A script doesn't directly have a name assigned, however if the script is a child script and associated with a scene object, then the script name is the name of the associated scene object. See also simIsValidHandle .
C synopsis	simInt simGetScriptHandle(const simChar* scriptName)
C parameters	scriptName : name of the script
C return value	handle of the script if operation was successful, -1 otherwise
Lua synopsis	number scriptHandle=simGetScriptHandle(string scriptName)
Lua parameters	Same as C-function. Alternatively, scriptName can be nil or inexistent, in which case the current script's handle is returned
Lua return values	Same as C-function

simGetScriptName

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

simGetScriptProperty

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

simGetScriptRawBuffer

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

simGetScriptSimulationParameter

Description	Retrieves a main script's or child script's parameter from its simulation parameter list. Useful for simple interaction with the user, or for simple parameter exchange with other scripts. See also simSetScriptSimulationParameter , simUnpackInts and simUnpackFloats .
C synopsis	simChar* simGetScriptSimulationParameter(simInt scriptHandle, const simChar* parameterName, simInt* parameterLength)

C parameters	scriptHandle : handle of the main script or child script, or <code>sim_handle_main_script</code> or <code>sim_handle_all</code> . When <code>scriptHandle</code> is <code>sim_handle_all</code> , the function returns only one matching parameter encountered (other matching parameters might be different) parameterName : name of the parameter to retrieve parameterLength : the number of bytes that compose the value of the parameter (excluding the terminal zero)
C return value	value of the parameter or NULL if <code>parameterName</code> does not exist for the given script, or in case of an error. The user is in charge of releasing the returned value with <code>simReleaseBuffer</code> . The returned pointer points to <code>parameterLength</code> byte values, terminated by a terminal zero (the returned buffer may however contain several embedded zeros).
Lua synopsis	(1) <code>boolean/number/string parameterValue=simGetScriptSimulationParameter(number scriptHandle,string parameterName,boolean forceStringReturn=false)</code> (2) <code>table parameterValues,table scriptHandles=simGetScriptSimulationParameter(number targetScripts,string parameterName,boolean forceStringReturn=false)</code>
Lua parameters	(1) scriptHandle : handle of the script, or <code>sim_handle_main_script</code> or <code>sim_handle_self</code> . (2) targetScripts : <code>sim_handle_all</code> , <code>sim_handle_tree</code> or <code>sim_handle_chain</code> (with <code>sim_handle_tree</code> and <code>sim_handle_chain</code> the calling script is excluded). parameterName : name of the parameter to retrieve. forceStringReturn : forces the return of a string (i.e. raw value). False by default. If false, then the returned string will be converted to nil, false, true, a number or a string as appropriate (and in that order).
Lua return values	(1) parameterValue : value of the parameter, or nil in case of an error (or if that value is nil!). (2) parameterValues : table of parameter values or nil if no such parameter was found or in case of an error. scriptHandles : table of script handles associated with the parameter values (i.e. <code>parameterValue[i]</code> comes from the script with handle <code>scriptHandles[i]</code>) or nil if no such parameter was found or in case of an error. If the returned parameter value is a string, then it might contain any values (also embedded zeros)

simGetScriptText

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

simGetShapeColor

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

simGetShapeGeomInfo

Description	Retrieves geometric information related to a shape. See also simGetShapeMesh .
C synopsis	<code>simInt simGetShapeGeomInfo(simInt shapeHandle,simInt* intData,simFloat* floatData,simVoid* reserved)</code>
C parameters	shapeHandle : handle of the shape. See also simGetObjectHandle . intData (output): pointer to 5 integer values: <code>intData[0]</code> : the pure type of the shape. Undefined if the shape is a grouping. floatData (output): pointer to 5 float values: <code>floatData[0]</code> : X-size or diameter of the pure shape. Undefined if the shape is a grouping or not pure. <code>floatData[1]</code> : Y-size of the pure shape. Undefined if the shape is a grouping or not pure. <code>floatData[2]</code> : Z-size or height of the pure shape. Undefined if the shape is a grouping or not pure. <code>floatData[3]</code> : Inside scaling. Undefined if the shape is a grouping or not pure. reserved : reserved for future extensions. Set to NULL.
C return value	-1 in case of an error, otherwise bit-coded:

	bit0 set (1): shape is a grouping bit1 set (2): shape is pure bit2 set (4): shape is convex
Lua synopsis	number result,number pureType,table_4 dimensions=simGetShapeGeomInfo(number shapeHandle)
Lua parameters	shapeHandle : handle of the shape. See also simGetObjectHandle .
Lua return values	result : -1 in case of an error, otherwise bit-coded: bit0 set (1): shape is a grouping bit1 set (2): shape is pure bit2 set (4): shape is convex pureType : the pure type of the shape. Undefined if the shape is a grouping. dimensions : table to 4 values giving information about the shape's dimensions: dimensions[1]: X-size or diameter of the pure shape. Undefined if the shape is a grouping or not pure. dimensions[2]: Y-size of the pure shape. Undefined if the shape is a grouping or not pure. dimensions[3]: Z-size or height of the pure shape. Undefined if the shape is a grouping or not pure. dimensions[4]: Inside scaling. Undefined if the shape is a grouping or not pure.

simGetShapeMassAndInertia

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

simGetShapeMaterial

Description	Retrieves the material ID of a specific shape. See also simGetMaterialId and simSetShapeMaterial .
C synopsis	simInt simGetShapeMaterial(simInt shapeHandle)
C parameters	shapeHandle : the handle of the shape. See also simGetObjectHandle .
C return value	The material ID, or -1 in case of an error
Lua synopsis	number materialId=simGetShapeMaterial(number shapeHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetShapeMesh

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

simGetShapeTextureId

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

simGetSignalName

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

simGetSimulationPassesPerRenderingPass

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

simGetSimulationState

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

simGetSimulationTime

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

simGetSimulationTimeStep

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

simGetSimulatorMessage

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

simGetStringParameter

remote API equivalent: [simxGetStringParameter](#)
ROS API equivalent: [simRosGetStringParameter](#)

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

simGetStringSignal

remote API equivalent: [simxGetStringSignal](#)
ROS API equivalent: [simRosGetStringSignal](#)

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

simGetSystemTime

Description	Retrieves the system time. The system time is the time in seconds that elapsed since Windows or V-REP was started, depending on the system. See also simGetSystemTimeInMs .
C synopsis	simFloat simGetSystemTime()
C parameters	None
C return value	system time in seconds, or a negative value (-1.0) in case of an error
Lua synopsis	number systemTime=simGetSystemTime()
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetSystemTimeInMilliseconds (DEPRECATED)

Description	DEPRECATED. See simGetSystemTimeInMs instead.
-------------	---

simGetSystemTimeInMs

Description	Retrieves a time in milliseconds.
C synopsis	<code>simUInt simGetSystemTimeInMs(simInt previousTime)</code>
C parameters	<p>previousTime: value that indicates how the command should operate: >=0: the function returns a time difference with previousTime. PreviousTime must have been previously retrieved with the -1 argument below. -1: the function returns a time relative to an arbitrary time. Use this to measure time differences within V-REP -2: the function returns a time as follows:</p> <pre>// On Windows: returnedValue=TimeGetTime(); // On MacOS / Linux: struct timeval now; gettimeofday(&now,NULL); returnValue=now.tv_sec*1000+now.tv_usec/1000;</pre>
C return value	a time in milliseconds as described here above.
Lua synopsis	<code>number systemTimeOrTimeDiff=simGetSystemTimeInMs(number previousTime)</code>
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetTextureId

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

simGetThreadId

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

simGetUIButtonLabel

Description	Gets the up-state label of a custom user interface button (useful to retrieve the text in an edit-box style button). See also simSetUIButtonLabel .
C synopsis	<code>simChar* simGetUIButtonLabel(simInt uiHandle,simInt buttonHandle)</code>
C parameters	<p>uiHandle: custom user interface handle buttonHandle: handle of a button inside the specified custom user interface</p>
C return value	label of the button, or NULL in case of an error. The user is in charge of releasing the returned string with simReleaseBuffer
Lua synopsis	<code>string label=simGetUIButtonLabel(number uiHandle,number buttonHandle)</code>
Lua parameters	Same as C-function
Lua return values	Same as C-function (but nil instead of NULL, and simReleaseBuffer does not need to be called)

simGetUIButtonProperty

remote API equivalent: [simxGetUIButtonProperty](#)
 ROS API equivalent: [simRosGetUIButtonProperty](#)

Description	Retrieves the properties of a custom user interface button. See also simSetUIButtonProperty .
C synopsis	<code>simInt simGetUIButtonProperty(simInt uiHandle,simInt buttonHandle)</code>
C parameters	<p>uiHandle: custom user interface handle buttonHandle: handle of a button inside the specified custom user interface</p>
C return value	button property values, -1 if operation was not successful
Lua synopsis	<code>number property=simGetUIButtonProperty(number uiHandle,number buttonHandle)</code>
Lua parameters	Same as C-function
Lua return values	Same as C-function. See also the boolean operators in Lua .

simGetUIButtonSize

Description	Retrieves the size (in terms of cell numbers) of a custom user interface button
C synopsis	simInt simGetUIButtonSize(simInt uiHandle,simInt buttonHandle,simInt* size)
C parameters	uiHandle : custom user interface handle buttonHandle : handle of a button inside the specified custom user interface size : size (in terms of cells (x and y))
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_2 size=simGetUIButtonSize(number uiHandle,number buttonHandle)
Lua parameters	uiHandle : custom user interface handle buttonHandle : handle of a button inside the specified custom user interface
Lua return values	size : table containing the x and y-size of the button (in terms of cells), or nil in case of an error

simGetUIEventButtonremote API equivalent: [simxGetUIEventButton](#)ROS API equivalent: [simRosGetUIEventButton](#)

Description	Gets the button handle and some auxiliary values of the last occurred event in that custom user interface. The function will then clear the event. When a custom user interface button is pressed, a slider is moved or an edit box is changed, an event is registered and stored in the custom user interface. See also the sim_message_ui_button_state_change message.
C synopsis	simInt simGetUIEventButton(simInt uiHandle,simInt* auxiliaryValues)
C parameters	uiHandle : custom user interface handle auxiliaryValues : 2 values (can be NULL). value1: the button property value2: for sliders: slider state (0-1000), for stay down buttons: down state (0 or 1), for up/down event buttons: up/down events (0 or 1)
C return value	button handle or -1 if no event occurred or if operation was not successful
Lua synopsis	number buttonHandle, table_2 auxiliaryValues=simGetUIEventButton(number uiHandle)
Lua parameters	uiHandle : custom user interface handle
Lua return values	buttonHandle : button handle or -1 if no event occurred or if operation was not successful auxiliaryValues : 2 values. Same as for the C-function. See also the boolean operators in Lua .

simGetUIHandleremote API equivalent: [simxGetUIHandle](#)ROS API equivalent: [simRosGetUIHandle](#)

Description	Retrieves the handle of a custom user interface. The operation of this function depends on the current name suffix settings (see simGetNameSuffix , simSetNameSuffix , and the section on accessing general-type objects). See also simIsHandleValid .
C synopsis	simInt simGetUIHandle(const simChar* uiName)
C parameters	uiName : name of the custom user interface
C return value	handle of custom user interface if value >=0, -1 if operation was not successful
Lua synopsis	number uiHandle=simGetUIHandle(string uiName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetUIPosition

Description	Retrieves the position of a custom user interface. See also simSetUIPosition .
C synopsis	simInt simGetUIPosition(simInt uiHandle,simInt* position)
C parameters	uiHandle : handle of the custom user interface position : pointer to 2 values (x and y)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	table_2 position=simGetUIPosition(number uiHandle)
Lua parameters	uiHandle : handle of the custom user interface
Lua return values	position : table of 2 values (x and y) or nil in case of an error

simGetUIProperty

Description	Retrieves the properties of a custom user interface. See also simSetUIProperty .
C synopsis	simInt simGetUIProperty(simInt uiHandle)
C parameters	uiHandle : handle of a custom user interface
C return value	Custom user interface property values, -1 if operation was not successful
Lua synopsis	number property=simGetUIProperty(number uiHandle)
Lua parameters	Same as C-function

Lua return values	Same as C-function. See also the boolean operators in Lua .
-------------------	---

simGetUISliderremote API equivalent: [simxGetUISlider](#)ROS API equivalent: [simRosGetUISlider](#)

Description	Gets the slider position of a custom user interface button (must be slider-type button). See also simSetUISlider .
C synopsis	simInt simGetUISlider(simInt uiHandle,simInt buttonHandle)
C parameters	uiHandle : handle of a custom user interface buttonHandle : handle of a button inside the specified custom user interface
C return value	slider position (value between 0 and 1000) or -1 if operation was not successful
Lua synopsis	number sliderPosition=simGetUISlider(number uiHandle,number buttonHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simGetVelocity

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

simGetVisionSensorDepthBufferremote API equivalent: [simxGetVisionSensorDepthBuffer](#)ROS API equivalent: [simRosGetVisionSensorDepthBuffer](#)

Description	Retrieves the depth buffer (or a portion of it) of a vision sensor. Use simGetVisionSensorResolution to know the resolution of the full depth buffer. The returned data doesn't make sense if simHandleVisionSensor wasn't called previously (simHandleVisionSensor is called by default in the main script if the vision sensor is not tagged as explicit handling).
C synopsis	simFloat* simGetVisionSensorDepthBuffer(simInt sensorHandle)
C parameters	sensorHandle : handle of the vision sensor
C return value	depth buffer (buffer size is resolutionX*resolutionY) or NULL in case of an error. The user is in charge of releasing the returned buffer with simReleaseBuffer . Returned values are in the range of 0-1 (0=closest to sensor, 1=farthest from sensor)
Lua synopsis	table depthBuffer=simGetVisionSensorDepthBuffer(number sensorHandle,number posX=0,number posY=0,number sizeX=0,number sizeY=0)
Lua parameters	sensorHandle : handle of the vision sensor posX / posY : position of the depth buffer portion to retrieve. Zero by default. sizeX / sizeY : size of the depth buffer portion to retrieve. Zero by default, which means that the full depth buffer should be retrieved
Lua return values	depthBuffer : table containing depth values (table size is sizeX*sizeY) or nil in case of an error. Returned values are in the range of 0-1 (0=closest to sensor, 1=farthest from sensor)

simGetVisionSensorCharImageremote API equivalent: [simxGetVisionSensorImage](#)ROS API equivalent: [simRosGetVisionSensorImage](#)

Description	Retrieves the rgb-image (or a portion of it) of a vision sensor. The returned data doesn't make sense if simHandleVisionSensor wasn't called previously (simHandleVisionSensor is called by default in the main script if the vision sensor is not tagged as explicit handling). See also simGetVisionSensorImage and simSetVisionSensorCharImage .
C synopsis	simUChar* simGetVisionSensorCharImage(simInt sensorHandle,simInt* resolutionX,simInt* resolutionY)
C parameters	sensorHandle : handle of the vision sensor resolutionX/resolutionY : the returned vision sensor resolution
C return value	image buffer (buffer size is resolutionX*resolution*3) or NULL in case of an error. The user is in charge of releasing the returned buffer with simReleaseBuffer . Returned values are in the range of 0-255 (0=min. intensity, 255=max. intensity)
Lua synopsis	string imageBuffer,number resolutionX,number resolutionY=simGetVisionSensorCharImage(number

	sensorHandle,number posX=0,number posY=0,number sizeX=0,number sizeY=0)
Lua parameters	sensorHandle : handle of the vision sensor posX / posY : position of the image portion to retrieve. Zero by default. sizeX / sizeY : size of the image portion to retrieve. Zero by default, which means that the full image should be retrieved
Lua return values	imageBuffer : nil in case of an error. Otherwise a string containing rgb values (table size is sizeX*sizeY*3, rgb values in the range 0-255)

simGetVisionSensorImageremote API equivalent: [simxGetVisionSensorImage](#)ROS API equivalent: [simRosGetVisionSensorImage](#)

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

simGetVisionSensorResolution

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

simGroupShapes

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

simHandleChildScript (REPLACED)

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

simHandleChildScripts

Description	Executes non-threaded child scripts located in the current scene hierarchy branch. By default child scripts will be executed in a cascaded fashion and this command should only be called from the main script , except for special situations (in that case refer also to the function call)
-------------	---

	simSetScriptAttribute (scriptHandle,sim_childscriptattribute_automaticcascadingcalls,false)).
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number executedScriptCount=simHandleChildScripts(number callType,...=nil)
Lua parameters	callType : the child script call type: can be any arguments that you want to pass to the child scripts that will be called. Inside the called child scripts, the arguments can be retrieved with arguments=...
Lua return values	executedScriptCount: number of executed non-threaded child scripts

simHandleCollision

Description	Handles (check for collision, etc.) a registered collision object. Collision objects can be registered while editing a scene. See also simReadCollision , simResetCollision , simCheckCollision and simCheckCollisionEx .
C synopsis	simInt simHandleCollision(simInt collisionObjectHandle)
C parameters	collisionObjectHandle : handle of the collision object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will handle all registered collision objects, while sim_handle_all_except_explicit will only handle those that are not marked as "explicit handling")
C return value	number of collisions or -1 if operation was not successful
Lua synopsis	number collisionCount=simHandleCollision(number collisionObjectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simHandleDistance

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

simHandleDynamics

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

simHandleGeneralCallbackScript

Description	Calls the general callback script .
C synopsis	simInt simHandleGeneralCallbackScript(simInt callbackId,simInt callbackTag,simVoid* reserved)
C parameters	callbackId : the callback ID that will be submitted to the general callback script as sim_callback_id variable. For user defined callbacks, use an ID starting at sim_callbackid_userdefined. callbackTag : the callback tag that will be submitted to the general callback script as sim_callback_tag variable. The interpretation of this tag depends on the callback ID. reserved : reserved for future extensions. Set to NULL.
C return value	the value the general callback script returned. If there is no general callback script, then the returned value will be -1.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simHandleGraph

Description	Handles a graph object (i.e. records current values of registered data streams). Graphs and data streams can be added/registered while editing a scene. See also simResetGraph .
-------------	--

C synopsis	simInt simHandleGraph(simInt graphHandle,simFloat simulationTime)
C parameters	graphHandle: handle of the graph object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will handle all graph objects, while sim_handle_all_except_explicit will only handle those that are not marked as "explicit handling") simulationTime: simulation time. Usually you want to record data stream at the end of a simulation pass to record actualized value: then set simulationTime to simGetSimulationTime()+simGetSimulationTimeStep() .
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simHandleGraph(number graphHandle,number simulationTime)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simHandleIkGroup

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

simHandleJoint

Description	Handles the motion of a joint, when the joint is in motion mode. See also simResetJoint .
C synopsis	simInt simHandleJoint(simInt jointHandle,simFloat deltaTime)
C parameters	jointHandle: handle of the joint object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will handle all joint objects, while sim_handle_all_except_explicit will only handle those that are not marked as "explicit handling") deltaTime: time variation between previous call (typically simGetSimulationTimeStep())
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simHandleJoint(number jointHandle,number deltaTime)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simHandleMainScript

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

simHandleMechanism

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

simHandleMill

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

simHandleModule

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

simHandlePath

Description	Handles the motion along a path object (if the functionality is enabled for the given path). This function performs similarly with the simHandleJoint command, but for path objects. See also simResetPath .
C synopsis	simInt simHandlePath(simInt pathHandle,simFloat deltaTime)
C parameters	pathHandle : handle of the path object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will handle all path objects, while sim_handle_all_except_explicit will only handle those that are not marked as "explicit handling") deltaTime : time variation between previous call (typically simGetSimulationTimeStep())
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simHandlePath(number pathHandle,number deltaTime)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simHandleProximitySensor

Description	Handles (performs sensing, etc. of) a registered proximity sensor object. See also simReadProximitySensor , simCheckProximitySensor , simCheckProximitySensorEx and simResetProximitySensor .
C synopsis	simInt simHandleProximitySensor(simInt sensorHandle,simFloat* detectedPoint,simInt* detectedObjectHandle,simFloat* detectedSurfaceNormalVector)
C parameters	sensorHandle : handle of a proximity sensor object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will handle all proximity sensor objects, while sim_handle_all_except_explicit will only handle those that are not marked as "explicit handling") detectedPoint : coordinates of the closest detected point (x, y and z: detectedPoint[0]-detectedPoint[2]) relative to the sensor reference frame, and distance to the detected point (1 value: detectedPoint[3]). Can be NULL detectedObjectHandle : handle of the object that was detected. Can be NULL detectedSurfaceNormalVector : normal vector (normalized) of the detected surface. Relative to the sensor reference frame. Can be NULL When several proximity sensors are handled at the same time (e.g. with the sim_handle_all argument), then the output values are relative to the closest detection distance
C return value	0 if nothing was detected, -1 in case of an error. In a future release, a more detailed return value might be available
Lua synopsis	number result,number distance,table_3 detectedPoint,number detectedObjectHandle,table_3 detectedSurfaceNormalVector=simHandleProximitySensor(number sensorHandle)

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

simHandleSensingChildScripts (REPLACED)

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

simHandleVarious

Description	Handles various functionalities (e.g. camera tracking during simulation, etc.). Should only be called once per simulation pass if possible.
C synopsis	simInt simHandleVarious()
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	number result=simHandleVarious()
Lua parameters	Same as C-function
Lua return values	Same as C-function

simHandleVisionSensor

Description	Handles (performs sensing, etc. of) a registered vision sensor object. Calling this function from one thread, then later from another thread is not recommended and might slow down the simulation. See also simReadVisionSensor , simCheckVisionSensor , simCheckVisionSensorEx and simResetVisionSensor .
C synopsis	simInt simHandleVisionSensor(simInt visionSensorHandle, simFloat** auxValues, simInt** auxValuesCount)
C parameters	<p>visionSensorHandle: handle of a vision sensor object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will handle all vision sensor objects, while sim_handle_all_except_explicit will only handle those that are not marked as "explicit handling")</p> <p>auxValues: auxiliary values returned from the applied filters (refer to the documentation for details). By default V-REP returns one packet of 15 auxiliary values: the minimum of {intensity, red, green, blue, depth value}, the maximum of {intensity, red, green, blue, depth value}, and the average of {intensity, red, green, blue, depth value}. If additional filter components return values, then they will be appended as packets to the first packet. AuxValues can be NULL. The user is in charge of releasing the auxValues buffer with simReleaseBuffer(*auxValues). If visionSensorHandle is sim_handle_all or sim_handle_all_except_explicit, nothing is returned in auxValues.</p> <p>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).</p> <p>USAGE EXAMPLE:</p> <pre>float* auxValues=NULL; int* auxValuesCount=NULL; float averageColor[3]={0.0f,0.0f,0.0f}; if (simHandleVisionSensor(visionSensorHandle,&auxValues,&auxValuesCount)>=0) { if ((auxValuesCount[0]>0) (auxValuesCount[1]>=15)) { averageColor[0]=auxValues[11]; averageColor[1]=auxValues[12]; averageColor[2]=auxValues[13]; } simReleaseBuffer((char*)auxValues); simReleaseBuffer((char*)auxValuesCount); } </pre>
C return value	number of detections (number of vision sensors that triggered a detection), -1 in case of an error
Lua synopsis	number detectionCount, table auxiliaryValuePacket1, table auxiliaryValuePacket2, etc.=simHandleVisionSensor(number visionSensorHandle)
Lua parameters	Same as C-function
Lua return values	detectionCount: number of detections (number of vision sensors that triggered a detection), -1 in case of an error

auxiliaryValuePacket1: default auxiliary value packet (same as for the C-function)
auxiliaryValuePacket2: additional auxiliary value packet (e.g. from a filter component)
auxiliaryValuePacket3: etc. (the function returns as many tables as there are auxiliary value packets)

simImportMesh

Description	Imports a mesh from a file. See also simExportMesh , simImportShape and simCreateMeshShape
C synopsis	simInt simImportMesh(simInt fileformat,const simChar* pathAndFilename,simInt options,simFloat identicalVerticeTolerance,simFloat scalingFactor,simFloat*** vertices,simInt** verticesSizes,simInt*** indices,simInt** indicesSizes,simFloat*** reserved,simChar*** names)
C parameters	<p>fileformat: the fileformat to import from. 0 for OBJ format, 1 for DXF format, 2 for 3DS format, 3 for ASCII STL format and 4 for BINARY STL format</p> <p>pathAndFilename: the location of the file to import.</p> <p>options: bit-coded: bit0 set (1): keep identical vertices, bit1 set (2): keep identical triangles, bit2 set (4): don't correct triangle windings</p> <p>identicalVerticeTolerance: the distance from which two distinct vertices will be merged. Bit0 of options should be cleared for this to have an effect</p> <p>scalingFactor: the scaling factor to apply to the imported vertices</p> <p>vertices: an array to vertice arrays. The import operation may generate several meshes depending on the fileformat. The user is in charge of releasing the memory. See the example below</p> <p>verticesSizes: an array indicating the individual vertice array sizes. The user is in charge of releasing the memory. See the example below</p> <p>indices: an array to indice arrays. The import operation may generate several meshes depending on the fileformat. The user is in charge of releasing the memory. Can be NULL. See the example below</p> <p>indicesSizes: an array indicating the individual indice array sizes. The user is in charge of releasing the memory. Can be NULL if indices is also NULL. See the example below</p> <p>reserved: reserved for future extensions. Keep at NULL.</p> <p>names: an array to mesh names extracted from the file. The import operation may generate several meshes depending on the fileformat. The user is in charge of releasing the memory. See the example below</p> <p>USAGE EXAMPLE:</p> <pre> simFloat** vertices; simInt* verticesSizes; simInt** indices; simInt* indicesSizes; simChar** names; simInt elementCount=simImportMesh(1,"d:\\example.dxf",0,0.0001f,1.0f,&vertices, &verticesSizes,&indices,&indicesSizes,NULL,&names); if (elementCount>0) { const float grey[3]={0.5f,0.5f,0.5f}; for (int i=0;i<elementCount;i++) { simInt shapeHandle=simCreateMeshShape(2,20.0f*3.1415f/180.0f,vertices[i], verticesSizes[i],indices[i],indicesSizes[i],NULL); simSetObjectName(shapeHandle,names[i]); simSetShapeColor(shapeHandle,"",sim_colorcomponent_ambient,grey); simReleaseBuffer(names[i]); simReleaseBuffer((simChar*)indices[i]); simReleaseBuffer((simChar*)vertices[i]); } simReleaseBuffer((simChar*)names); simReleaseBuffer((simChar*)indicesSizes); simReleaseBuffer((simChar*)indices); simReleaseBuffer((simChar*)verticesSizes); simReleaseBuffer((simChar*)vertices); } </pre>
C return value	Number of imported meshes, or 0 or -1 if the operation was not successful
Lua synopsis	table_of_table vertices,table_of_table indices,nil,table names=simImportMesh(number fileformat,string pathAndFilename,number options,number identicalVerticeTolerance,number scalingFactor)
Lua parameters	Same as C-function
Lua return values	<p>vertices: a table to vertice tables, or nil if operation was not successful. The import operation may generate several meshes depending on the fileformat. See the example below</p> <p>indices: a table to indice tables, or nil if operation was not successful. The import operation may generate several meshes depending on the fileformat. See the example below</p> <p>nil: return value is reserved for future extensions</p> <p>names: a table to mesh names extracted from the file, or nil if operation was not successful. The import operation may generate several meshes depending on the fileformat. See the example below</p> <p>USAGE EXAMPLE (e.g. in a customization script):</p> <pre> if (importButtonPressed) then vertices,indices,reserved,names=simImportMesh(1,"d:\\example.dxf",0,0.0001,1) if (vertices) then for i=1,#vertices,1 do h=simCreateMeshShape(2,20*math.pi/180,vertices[i],indices[i]) simSetShapeColor(h,"",sim_colorcomponent_ambient,{0.5,0.5,0.5}) simSetObjectName(h,names[i]) end end end </pre>

simImportShape

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

simInitializePathSearch

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

simInsertPathCtrlPoints

Description	Inserts one or several control points into a path object . See also simCutPathCtrlPoints and simCreatePath .
C synopsis	simInt simInsertPathCtrlPoints(simInt pathHandle,simInt options,simInt startIndex,simInt ptCnt,const simVoid* ptData)
C parameters	pathHandle : the handle of the path. Refer also to simGetObjectHandle . options : bit-coded: bit0: if set (1), then the path will be closed (given that there are enough control points in the path) startIndex : the zero-based index where the first new control point should be inserted. ptCnt : the number of control points to insert. ptData (input) : a buffer of ptCnt*11 values (float or int). Each new control point should have its properties described with following 11 values: ptData[0]-ptData[2] (float values): the position of the control point (x,y,z), relative to the path object ptData[3]-ptData[5] (float values): the orientation of the control point in Euler angles (alpha,beta,gamma), relative to the path object ptData[6] (float value): the relative velocity at the control point ptData[7] (float value): the virtual distance at the control point ptData[8] (int value): the number of Bezier points at the control point ptData[9] (float value): the Bezier interpolation factor 1 at the control point ptData[10] (float value): the Bezier interpolation factor 2 at the control point
C return value	-1 if operation was not successful.
Lua synopsis	number result=simInsertPathCtrlPoints(number pathHandle,number options,number startIndex,number ptCnt,table ptData)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simInterpolateMatrices

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

simInvertMatrix

Description	Inverts a transformation matrix. This function in the Lua-API has a different name: simGetInvertedMatrix
C synopsis	simInt simInvertMatrix(simFloat* matrix)
C parameters	matrix : pointer to 12 simFloat values representing the matrix that should be inverted (the last row of the 4x4 matrix (0,0,0,1) is not needed) The x-axis of the orientation component is (matrix[0],matrix[4],matrix[8]) The y-axis of the orientation component is (matrix[1],matrix[5],matrix[9]) The z-axis of the orientation component is (matrix[2],matrix[6],matrix[10]) The position component is (matrix[3],matrix[7],matrix[11])
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	-
Lua parameters	-
Lua return values	-

simIsHandleValid

Description	Checks whether a general object handle is still valid. When a general object is destroyed (e.g. programmatically or via the user interface), then its related handle is not valid anymore and will trigger an error when used. Use this function to avoid triggering an error. See also simGetObjectHandle , simGetUIHandle , simGetCollectionHandle , simGetCollisionHandle , simGetDistanceHandle , simGetMechanismHandle , simGetIkGroupHandle , simGetScriptHandle , simGetPathPlanningHandle and simGetObjectUniqueIdentifier .
C synopsis	simInt simIsHandleValid(simInt generalObjectHandle,simInt generalObjectType)
C parameters	generalObjectHandle : handle of a general-type object (e.g. scene object, collision object, distance object, etc.) generalObjectType : type of the general object. Refer to the general object types . Can be -1, in which case the specified handle is checked for validity in all types (handles of different types never overlap)
C return value	-1 if operation was not successful, 0 if the handle is not valid anymore, or 1 if the handle is still valid.
Lua synopsis	number result=simIsHandleValid(number generalObjectHandle,number generalObjectType=-1)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simIsObjectInSelection

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

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

simIsRealTimeSimulationStepNeeded

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

simIsScriptExecutionThreaded

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

simIsScriptRunningInThread (DEPRECATED)

Description	DEPRECATED. Refer to simIsScriptExecutionThreaded instead.
-------------	--

simJointGetForce (DEPRECATED)

Description	DEPRECATED. Refer to simGetJointForce instead.
-------------	--

simLaunchExecutable

Description	Launches an executable. Similar to os.execute or io.popen, but is system independent.
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result=simLaunchExecutable(string filename,string parameters="",number showStatus=1)
Lua parameters	filename: file name of the executable parameters: optional input arguments showStatus: 0 to hide the application's window, 1 to show it. Works only with Windows OS.
Lua return values	result: -1 if operation was not successful. Under Windows OS, if the application could not be launched, return value is -1. Under Mac OS or Linux, return value might be different from -1 even if the application could not be launched.

simLaunchThreadedChildScripts

Description	Starts or restarts (if appropriately flagged) all threaded child scripts . This command should only be called from the main script . Refer also to following related functions: simSwitchThread , simSetThreadSwitchTiming , simSetThreadAutomaticSwitch , simSetThreadIsFree , simSetThreadResumeLocation and simResumeThreads .
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number launchCount=simLaunchThreadedChildScripts()
Lua parameters	-
Lua return values	launchCount: number of launched threaded child scripts

simLoadModel

remote API equivalent: [simxLoadModel](#)
ROS API equivalent: [simRosLoadModel](#)

Description	Loads a previously saved model, and selects it. See also simSaveModel , simLoadUI , simLoadScene , and simSetBooleanParameter with sim_boolparam_scene_and_model_load_messages .
C synopsis	simInt simLoadModel(const simChar* filename)

C parameters	filename: model filename. The filename extension is required ("ttm")
C return value	-1 if operation was not successful. Otherwise the handle of the model base object.
Lua synopsis	number result=simLoadModel(string filename)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simLoadModule

Description	Loads a V-REP plugin. This should be done in the main client application , just after simRunSimulator was called. See also simSendMessage and simUnloadModule .
C synopsis	simInt simLoadModule(const simChar* filenameAndPath,const simChar* pluginName)
C parameters	filenameAndPath: file name and path of the plugin pluginName: name of the plugin. If the file name is v_repExtXXX.dll, then the name should be XXX
C return value	handle of the plugin if value is 0 or positive. otherwise: -3: plugin could not be loaded -2: plugin is missing entry points -1: plugin could not initialize
Lua synopsis	-
Lua parameters	-
Lua return values	-

simLoadScene

remote API equivalent: [simxLoadScene](#)
ROS API equivalent: [simRosLoadScene](#)

Description	Loads a previously saved scene. If current scene is not empty, a new scene will be created before switching to it and loading the scene. See also simSaveScene , simLoadModel , simLoadUI , simCloseScene and simSetBooleanParameter with sim_boolparam_scene_and_model_load_messages .
C synopsis	simInt simLoadScene(const simChar* filename)
C parameters	filename: scene filename. The filename extension is required ("tst")
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=simLoadScene(string filename)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simLoadUI

remote API equivalent: [simxLoadUI](#)
ROS API equivalent: [simRosLoadUI](#)

Description	Loads a previously saved custom user interface file (*.ttb-files). See also simSaveUI , simLoadModel , simLoadScene , and simSetBooleanParameter with sim_boolparam_scene_and_model_load_messages . When interacting with a UI from an add-on , you might have to add the property flag sim_ui_property_pauseactive to the UI, in order to also have it respond while simulation is paused.
C synopsis	simInt simLoadUI(const simChar* filename,int maxCount,int* uiHandles)
C parameters	filename: UI filename. The filename extension is required ("ttb") maxCount: the maximum number of items that the uiHandles pointer can hold uiHandles: a pointer to integer values that will receive the handles of the loaded UIs (a single UI file can hold an unlimited number of UI elements)
C return value	-1 if operation was not successful, otherwise the number of UI elements loaded.
Lua synopsis	table handles=simLoadUI(string filename)
Lua parameters	Same as C-function
Lua return values	handles: a table containing the handles of all loaded UI elements, or nil in case of an error

simLockInterface (DEPRECATED)

Description	DEPRECATED. Has no effect anymore.
-------------	------------------------------------

simLockResources

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

C return value	-1 if the lock has failed. Otherwise a lock handle. A locked should always eventually be released, otherwise you will experience a dead-lock or crash.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simModifyGhost

Description	Modifies or removes a ghost object previously added with simAddGhost .
C synopsis	simInt simModifyGhost(simInt ghostGroup,simInt ghostId,simInt operation,simFloat floatValue,simInt options,simInt optionsMask,const simFloat* colorOrTransformation)
C parameters	<p>ghostGroup: an identifier that allows, together with ghostId, identifying which ghosts to modify.</p> <p>ghostId: an identifier that allows, together with ghostGroup, identifying which ghosts to modify. If -1, then all ghosts that match ghostGroup will be modified.</p> <p>operation: a value indicating the operation to perform:</p> <ul style="list-style-type: none"> 0: no operation is performed, and the return value indicates the number of ghosts that match the identifiers. 1: removes the specified ghosts. The return value indicates the number of removed ghosts. 2: sets the start time (via floatValue) of the specified ghosts. The return value indicates the number of modified ghosts. 3: sets the end time (via floatValue) of the specified ghosts. The return value indicates the number of modified ghosts. 4: shifts the start time (via floatValue) of the specified ghosts. The return value indicates the number of modified ghosts. 5: shifts the end time (via floatValue) of the specified ghosts. The return value indicates the number of modified ghosts. 6: shifts the start and end times (via floatValue) of the specified ghosts. The return value indicates the number of modified ghosts. 7: scales the start time (via floatValue) of the specified ghosts. The return value indicates the number of modified ghosts. 8: scales the end time (via floatValue) of the specified ghosts. The return value indicates the number of modified ghosts. 9: scales the start and end times (via floatValue) of the specified ghosts. The return value indicates the number of modified ghosts. 10: modifies the attributes (via options and optionsMask) of the specified ghosts. The return value indicates the number of modified ghosts. 11: pre-multiplies (via colorOrTransformation) the transformations of the specified ghosts. The return value indicates the number of modified ghosts. 12: post-multiplies (via colorOrTransformation) the transformations of the specified ghosts. The return value indicates the number of modified ghosts. 13: modifies (via colorOrTransformation) the color of the specified ghosts. The return value indicates the number of modified ghosts. 14: sets the transparency factor (via floatValue) of the specified ghosts. Only ghosts with custom colors will be influenced. The return value indicates the number of modified ghosts. <p>floatValue: a floating point value that is used to modify the specified ghosts. See operation here above.</p> <p>options: the attributes used to modify the specified ghosts (see operation here above). Attributes are bit-coded:</p> <ul style="list-style-type: none"> bit0 reserved bit1 set (2)=the provided start- and end-times will be played-back in real-time bit2 set (4)=preserve the original colors bit3 reserved bit4 set (16)=create an invisible ghost bit5 set (32)=backface culling for the ghost (only when using custom colors) <p>optionsMask: a mask allowing to select options to set.</p> <p>colorOrTransformation: a pointer to 7 or 12 float values: when operation is 11 or 12, 7 values indicating a transformation should be provided (x,y,z,qx,qy,qz,qw). when operation is 13, 12 values indicating a color should be provided (ambient_diffuse RGB, 3 reserved values (set to zero), specular RGB and emissive RGB). Can be NULL when operation is different from 11, 12 or 13.</p>
C return value	-1 if operation was not successful, otherwise a value depending on the operation performed.
Lua synopsis	number result=simModifyGhost(number ghostGroup,number ghostId,number operation,number floatValue,number options=nil,number optionsMask=nil,table colorOrTransformation=nil)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simModifyPointCloud

Description	Modifies a point cloud previously added with simAddPointCloud .
C synopsis	simInt simModifyPointCloud(simInt pointCloudHandle,simInt operation,const simInt* intParam,const simFloat* floatParam)
C parameters	<p>pointCloudHandle: the handle of the point cloud to modify (i.e. the handle returned by simAddPointCloud).</p> <p>operation: a value indicating the operation to perform on the point cloud:</p> <ul style="list-style-type: none"> 0=removes the point cloud. <p>intParam: reserved for future extensions. Keep at NULL.</p> <p>floatParam: reserved for future extensions. Keep at NULL.</p>

C return value	-1 in case of an error or if the operation failed.
Lua synopsis	number result=simModifyPointCloud(number pointCloudHandle,number operation)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simMoveToJointPositions (DEPRECATED)

Description	DEPRECATED. See simRMLMoveToJointPositions instead.
-------------	---

simMoveToObject

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

simMoveToPosition (DEPRECATED)

Description	DEPRECATED. Use simRMLMoveToPosition instead.
-------------	---

simMsgBox

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

simMultiplyMatrices

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

Lua parameters	Same as C-function
Lua return values	resultMatrix : the result matrix (a table containing 12 values (the last row (0,0,0,1) is omitted)). Table values in Lua are indexed from 1, not 0!

simMultiplyVector

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

simOpenModule

Description	"Opens" a plugin (allowing it to reserve resources at the start of a simulation). This command can only be called from the main script. Call it from the main script in the first simulation pass (usually with <code>sim_handle_all</code> argument). simOpenModule 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	(1) number result=simOpenModule(number sim_handle_all) (2) number result=simOpenModule(string moduleName)
Lua parameters	<code>sim_handle_all</code> : indicates that all plugins should be opened <code>moduleName</code> : the name of the plugin that should be opened
Lua return values	result : -1 in case of an error, otherwise result is the number of plugins that executed the command.

simPackBytes

Description	Packs a table of byte numbers into a string. See also simUnpackBytes , simPackInts , simPackWords and simPackFloats .
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	string data=simPackBytes(table byteNumbers)
Lua parameters	byteNumbers : a table containing byte numbers. Invalid byte number will be packed in an undefined manner.
Lua return values	data : a string (values between 0 and 255) that contains the byte numbers from the table if operation was successful, nil otherwise

simPackFloats

Description	Packs a table of floating-point numbers into a string. See also simUnpackFloats , simPackInts , simPackWords and simPackBytes .
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	string data=simPackFloats(table floatingNumbers)
Lua parameters	floatingNumbers : a table containing floating-point numbers. Non-numbers will be packed as zero values
Lua return values	data : a string (values between 0 and 255) that contains packed floating-point numbers from the table if operation was successful, nil otherwise

simPackInts

Description	Packs a table of integer numbers into a string. See also simUnpackInts , simPackFloats , simPackWords and simPackBytes .
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	string data=simPackInts(table integerNumbers)
Lua parameters	integerNumbers : a table containing integer numbers. Non-numbers will be packed as zero values
Lua return values	data : a string (values between 0 and 255) that contains packed integer numbers from the table if operation was successful, nil otherwise

simPackWords

Description	Packs a table of word numbers into a string. See also simUnpackWords , simPackFloats , simPackInts and simPackBytes .
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	string data=simPackWords(table wordNumbers)
Lua parameters	wordNumbers : a table containing word numbers. Invalid word numbers will be packed in an undefined manner.
Lua return values	data : a string (values between 0 and 255) that contains packed word numbers from the table if operation was successful, nil otherwise

simPauseSimulation

remote API equivalent: [simxPauseSimulation](#)
 ROS API equivalent: [simRosPauseSimulation](#)

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

simPerformPathSearchStep

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

simPersistentDataRead

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

simPersistentDataWrite

Description	Writes a persistent data block. Persistent data, valid across all opened simulator scenes, remains until the simulator ends, or until it is cleared by writing an empty data block. If the options flag is set appropriately, then persistent data can also be stored on file, and be automatically reloaded next time V-REP starts. See also simPersistentDataRead , simSetStringSignal , simSetIntegerSignal , simSetFloatSignal and simAddSceneCustomData .
C synopsis	simInt simPersistentDataWrite(const simChar* dataName,const simChar* dataValue,simInt dataLength,simInt options)
C parameters	dataName : name of the data block dataValue : content of the data block (which may contain any value, including embedded zeros). If

	dataValue is an empty string, then the data block is cleared (if present). dataLength : the size of the data block. options : bit-coded. If bit 0 is set (1), then the data is also stored on file ("system/persistentData.dat"), and automatically reloaded next time V-REP start.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simPersistentDataWrite(string dataName,string dataValue,number options=0)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simQuitSimulator

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

simReadCollision

remote API equivalent: [simxReadCollision](#)
ROS API equivalent: [simRosReadCollision](#)

Description	Reads the collision state of a registered collision object. This function doesn't perform collision detection, it merely reads the result from a previous call to simHandleCollision (simHandleCollision is called in the default main script). See also simResetCollision , simCheckCollision and simCheckCollisionEx .
C synopsis	simInt simReadCollision(simInt collisionObjectHandle)
C parameters	collisionObjectHandle : handle of the collision object
C return value	collision state (0 or 1), or -1 if operation was not successful, or if simHandleCollision was never called, or if simResetCollision was previously called.
Lua synopsis	number collisionState=simReadCollision(number collisionObjectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simReadCustomDataBlock

Description	Reads custom data that is stored inside of an object, a script or a scene (i.e. the data is be part of the object, the script or the scene). See also simWriteCustomDataBlock .
C synopsis	simChar* simReadCustomDataBlock(simInt objectHandle,const simChar* dataName,simInt* dataSize)
C parameters	objectHandle : handle of the object or script where the data is stored, or sim_handle_scene if the data is stored in the scene. dataName : a string that identifies the data. dataSize : a pointer to an integer receiving the size of the returned buffer.
C return value	the custom data block, or nil in case of an error (or if the data is not present). The user is in charge of releasing the returned buffer with simReleaseBuffer .
Lua synopsis	string customDataBlock=simReadCustomDataBlock(number objectHandle,string dataName)
Lua parameters	Same as C-function. In addition, you can specify sim_handle_self for the objectHandle argument, if your target is the current script.
Lua return values	Same as C-function.

simReadDistance

remote API equivalent: [simxReadDistance](#)
ROS API equivalent: [simRosReadDistance](#)

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

simReadForceSensor
 remote API equivalent: [simxReadForceSensor](#)
 ROS API equivalent: [simRosReadForceSensor](#)

Description	Reads the force and torque applied to a force sensor (filtered values are read), and its current state ('unbroken' or 'broken'). See also simBreakForceSensor and simGetJointForce .
C synopsis	<code>simInt simReadForceSensor(simInt objectHandle,simFloat* forceVector,simFloat* torqueVector)</code>
C parameters	objectHandle : handle of the object (must be a force sensor) forceVector : pointer to 3 values (applied forces along the sensor's x, y and z-axes). Can be NULL torqueVector : pointer to 3 values (applied torques about the sensor's x, y and z-axes). Can be NULL
C return value	-1 in case of an error, otherwise bit-coded: bit 0 set (1): force and torque data is available, otherwise it is not (yet) available (e.g. when not enough values are present for the filter) bit 1 set (2): force sensor is broken, otherwise it is still intact ('unbroken')
Lua synopsis	<code>number result, table_3 forceVector,table_3 torqueVector=simReadForceSensor(number objectHandle)</code>
Lua parameters	Same as C-function
Lua return values	result : -1 in case of an error, otherwise bit-coded (same as for the C-function return value) forceVector : table holding 3 values (applied forces along the sensor's x, y and z-axes) torqueVector : table holding 3 values (applied torques about the sensor's x, y and z-axes)

simReadProximitySensor
 remote API equivalent: [simxReadProximitySensor](#)
 ROS API equivalent: [simRosReadProximitySensor](#)

Description	Reads the state of a proximity sensor. This function doesn't perform detection, it merely reads the result from a previous call to simHandleProximitySensor (<code>simHandleProximitySensor</code> is called in the default main script). See also simCheckProximitySensor , simCheckProximitySensorEx and simResetProximitySensor .
C synopsis	<code>simInt simReadProximitySensor(simInt sensorHandle,simFloat* detectedPoint,simInt* detectedObjectHandle,simFloat* detectedSurfaceNormalVector)</code>
C parameters	sensorHandle : handle of a proximity sensor object detectedPoint : coordinates of the closest detected point (x, y and z: <code>detectedPoint[0]-detectedPoint[2]</code>) relative to the sensor reference frame, and distance to the detected point (1 value: <code>detectedPoint[3]</code>). Can be NULL detectedObjectHandle : handle of the object that was detected. Can be NULL detectedSurfaceNormalVector : normal vector (normalized) of the detected surface. Relative to the sensor reference frame. Can be NULL
C return value	detection state (0 or 1), or -1 in case of an error, or if simHandleProximitySensor was never called, or if simResetProximitySensor was previously called.
Lua synopsis	<code>number result,number distance,table_3 detectedPoint,number detectedObjectHandle,table_3 detectedSurfaceNormalVector=simReadProximitySensor(number sensorHandle)</code>
Lua parameters	Same as C-function
Lua return values	result : detection state (0 or 1), or -1 in case of an error, or if simHandleProximitySensor was never called, or if simResetProximitySensor was previously called. distance : distance to the detected point if result is 1, nil otherwise detectedPoint : table of 3 numbers indicating the relative coordinates of the detected point if result is 1, nil otherwise detectedObjectHandle : handle of the object that was detected if result is 1, nil otherwise detectedSurfaceNormalVector : normal vector (normalized) of the detected surface. Relative to the sensor reference frame. Is nil if result is different from 1

simReadTexture

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

simReadVisionSensor
 remote API equivalent: [simxReadVisionSensor](#)
 ROS API equivalent: [simRosReadVisionSensor](#)

Description	Reads the state of a vision sensor. This function doesn't perform detection, it merely reads the result
-------------	---

	from a previous call to simHandleVisionSensor (<code>simHandleVisionSensor</code> is called in the default main script). See also simCheckVisionSensor , simCheckVisionSensorEx and simResetVisionSensor .
C synopsis	<code>simInt simReadVisionSensor(simInt visionSensorHandle,simFloat** auxValues,simInt** auxValuesCount)</code>
C parameters	visionSensorHandle : handle of a vision sensor object auxValues : auxiliary values returned from the applied filters . By default V-REP returns one packet of 15 auxiliary values:the minimum of {intensity, red, green, blue, depth value}, the maximum of {intensity, red, green, blue, depth value}, and the average of {intensity, red, green, blue, depth value}. If additional filter components return values, then they will be appended as packets to the first packet. AuxValues can be NULL. The user is in charge of releasing the auxValues buffer with simReleaseBuffer>(*auxValues) . auxValuesCount : contains information about the number of auxiliary value packets and packet sizes returned in auxValues. The first value is the number of packets, the second is the size of packet1, the third is the size of packet2, etc. Can be NULL if auxValues is also NULL. The user is in charge of releasing the auxValuesCount buffer with simReleaseBuffer>(*auxValuesCount) . See simHandleVisionSensor for a usage example
C return value	detection state (0 or 1), or -1 in case of an error, or if simHandleVisionSensor was never called, or if simResetVisionSensor was previously called.
Lua synopsis	<code>number result,table auxiliaryValuePacket1,table auxiliaryValuePacket2,etc.=simReadVisionSensor(number visionSensorHandle)</code>
Lua parameters	visionSensorHandle : handle of a vision sensor object
Lua return values	result : detection state (0 or 1), or -1 in case of an error, or if simHandleVisionSensor was never called, or if simResetVisionSensor was previously called. auxiliaryValuePacket1 : default auxiliary value packet (same as for the C-function) auxiliaryValuePacket2 : additional auxiliary value packet (e.g. from a filter component) auxiliaryValuePacket3 : etc. (the function returns as many tables as there are auxiliary value packets)

simReceiveData

Description	Receives wireless data (in a simulation). See also simSendData and simTubeOpen . Cannot be called from add-ons . Wireless receptions can be visualized globally via the environment dialog , or individually as in following example: <pre>simSetBooleanParameter(sim_boolparam_force_show_wireless_reception,true) data=simReceiveData(...) simSetBooleanParameter(sim_boolparam_force_show_wireless_reception,false)</pre>
C synopsis	<code>simReceiveData(simInt dataHeader,const simChar* dataName,simInt antennaHandle,simInt index,simInt* dataLength,simInt* senderID,simInt* dataHeaderR,simChar** dataNameR)</code>
C parameters	dataHeader : number indicating who "designed" the communication message. Can also be -1, in which case messages with any dataHeader will be retrieved (not recommended, unless index is different from -1). For details on the dataHeader, also refer to simAddObjectCustomData . dataName : name indicating the type of message. Can be nil, in which case messages with any dataName will be retrieved (not recommended, unless index is different from -1) antennaHandle : handle of the scene object that should operate as the antenna for this reception. If <code>sim_handle_default</code> is specified, a reception antenna at (0,0,0) is simulated. index : zero-based index of the message to read. If -1 is indicated, the first message that matches the dataHeader and dataName is read and removed. Otherwise messages are just read. dataLength : length of the received data (if returned pointer is not NULL) senderID : identifier of the sender. Can be the handle of a script if the message was sent from a script, or can be 0 if the message was sent from the non-Lua API. Can be NULL. dataHeaderR : dataHeader of the data that was read. Can be NULL. dataNameR : dataName of the data that was read. Can be NULL. The user is in charge of releasing the buffer with simReleaseBuffer>(*dataNameR) .
C return value	pointer to the received data, or NULL if no data is available or in case of an error. The user is in charge of releasing the returned buffer with simReleaseBuffer .
Lua synopsis	<code>string data,number senderID,number dataHeader,string dataName=simReceiveData(number dataHeader=-1,string dataName=nil,number antennaHandle=sim_handle_self,number index=-1)</code>
Lua parameters	dataHeader : number indicating who "designed" the communication message. Can also be -1, in which case messages with any dataHeader will be retrieved (not recommended, unless index is different from -1). This value can be omitted (-1 will be used). For details on the dataHeader, also refer to simAddObjectCustomData . dataName : name indicating the type of message. Can be nil, in which case messages with any dataName will be retrieved (not recommended, unless index is different from -1). This value can be omitted (nil will be used) antennaHandle : handle of the scene object that should operate as the antenna for this reception. If <code>sim_handle_default</code> is specified, a reception antenna at (0,0,0) is simulated. If <code>sim_handle_self</code> is specified, the object associated with the current child script is used as the antenna. This value can be omitted (<code>sim_handle_self</code> will be used) index : zero-based index of the message to read. If -1 is indicated, the first message that matches the dataHeader and dataName is read and removed. Otherwise messages are just read. This value can be omitted (-1 will be used)
Lua return values	data : string containing the received data, or nil in case of an error or if no data is present. If received data is packed, see also simUnpackInts and simUnpackFloats senderID : identifier of the sender. Can be the handle of a script if the message was sent from a script, or can be 0 if the message was sent from the non-Lua API. dataHeader : dataHeader of the data that was read.

dataName: dataName of the data that was read.
--

simRefreshDialogs

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

simRegisterContactCallback

Description	Registers or unregisters a contact callback function for customized contact handling. Several plugins can register a callback, and the callbacks will be cascaded if not interrupted.
C synopsis	simInt simRegisterContactCallback(simInt(*callBack)(simInt,simInt,simInt,simInt*,simFloat*))
C parameters	<p>callback address. Specify the function you wish to register/unregister as callback. When the function is already registered, it will be unregistered. Following describes the callback's arguments:</p> <p>first argument (int): handle of the first object that is involved in the contact. Could also be the handle of a dynamic particle</p> <p>second argument (int): handle of the second object that is involved in the contact. Could also be the handle of a dynamic particle</p> <p>third argument (int): the selected physics engine</p> <p>The forth and fifth argument (allocated by V-REP) allow you to specify how the contact should behave (see also the respective Bullet, ODE and Vortex reference manuals):</p> <p>forth argument (int*)</p> <p>[0] should be set to zero (indicates the function version, for future extensions)</p> <p>[1] Bullet: not used (keep 0). ODE: contactCount (max. number of contact points to generate). Vortex: not used (keep 0).</p> <p>[2] Bullet: not used (keep 0). Vortex: not used (keep 0). ODE: contactMode (can be bit-wise combined):</p> <p>1=dContactMu2 2=dContactFDir1 4=dContactBounce 8=dContactSoftERP 16=dContactSoftCFM 32=dContactMotion1 64=dContactMotion2 128=dContactSlip1 256=dContactSlip2 512=dContactApprox1_1 1024=dContactApprox1_2 2048=dContactApprox1</p> <p>Fifth argument (float*)</p> <p>[0] Bullet: combined friction. ODE: mu. Vortex: not used (keep0).</p> <p>[1] Bullet: combined restitution. ODE: mu2. Vortex: not used (keep0).</p> <p>[2] Bullet: not used (keep 0.0). ODE: bounce. Vortex: not used (keep0).</p> <p>[3] Bullet: not used (keep 0.0). ODE: bounce_vel. Vortex: not used (keep0).</p> <p>[4] Bullet: not used (keep 0.0). ODE: soft_erp. Vortex: not used (keep0).</p> <p>[5] Bullet: not used (keep 0.0). ODE: soft_cfm. Vortex: not used (keep0).</p> <p>[6] Bullet: not used (keep 0.0). ODE: motion1. Vortex: not used (keep0).</p> <p>[7] Bullet: not used (keep 0.0). ODE: motion2. Vortex: not used (keep0).</p> <p>[8] Bullet: not used (keep 0.0). ODE: motionN. Vortex: not used (keep0).</p> <p>[9] Bullet: not used (keep 0.0). ODE: slip1. Vortex: not used (keep0).</p> <p>[10] Bullet: not used (keep 0.0). ODE: slip2. Vortex: not used (keep0).</p> <p>[11] Bullet: not used (keep 0.0). ODE: fdir1[0]. Vortex: not used (keep0).</p> <p>[12] Bullet: not used (keep 0.0). ODE: fdir1[1]. Vortex: not used (keep0).</p> <p>[13] Bullet: not used (keep 0.0). ODE: fdir1[2]. Vortex: not used (keep0).</p> <p>The callback return value can be:</p> <p>-1: the callback doesn't process the contact and hands the contact to the next callback. If no callback processes it, the default contact handling is used</p> <p>0: the contact is ignored and the two objects won't have a collision response</p> <p>>0: the contact will be handled with above's values (the default values are overridden)</p>
C return value	-1 if operation was not successful. 0 if the callback was unregistered, 1 if the callback was registered
Lua synopsis	-
Lua parameters	-
Lua return values	-

simRegisterCustomLuaFunction

Description	Registers a customized lua function. This function is useful for plugins (or the main client application) that wish to provide their own or customized Lua functions. See also simRegisterCustomLuaVariable .
C synopsis	<code>simInt simRegisterCustomLuaFunction(const simChar* funcName,const simChar* callTips,const simInt* inputArgumentTypes,simVoid(*callback)(struct SLuaCallBack* p))</code>
C parameters	<p>funcName: name of the function. Avoid using too simple function names, otherwise they might dash with other plugins. Also, always use the "simExt" prefix (e.g. "simExtMyCustomFunction").</p> <p>callTips: call tips: string (or several strings separated by '@') that indicates the input/output argument type/size. Call tips appear in the script editor when the function was typed followed by "(" . callTips Can be NULL</p> <p>inputArgumentTypes: array indicating the desired input arguments. This is important so that the simulator knows how to convert between Lua types (e.g. Lua value"4.2" can be converted to int 4, to float 4.2, to string "4.2" or to Boolean 1). Can be NULL, in that case no input argument is forwarded to the callback address. inputArgumentTypes[0] represents the number of input arguments we wish to be forwarded, inputArgumentTypes[1] is the type of the first argument, inputArgumentTypes[2] is the type of the second argument, etc. An input argument type can be <code>sim_lua_arg_bool</code>, <code>sim_lua_arg_int</code>, <code>sim_lua_arg_float</code>, <code>sim_lua_arg_string</code> or <code>sim_lua_arg_charbuff</code>, that can be combined () with <code>sim_lua_arg_table</code> if table values are desired. And exception to this is <code>sim_lua_arg_charbuff</code>, which cannot be combined with <code>sim_lua_arg_table</code>.</p> <p>callback: callback address that is called when the "funcName" function is called from Lua with the right arguments. The callback's first argument is a SLuaCallBack structure that holds:</p> <p>simInt objectID: handle of the object that the calling script is attached to, or -1 if the calling script is not a child script</p> <p>simChar* inputBool: pointer to all Boolean input arguments</p> <p>simInt* inputInt: pointer to all integer input arguments</p> <p>simFloat* inputFloat: pointer to all floating point input arguments</p> <p>simChar* inputChar: pointer to all string input arguments. Strings are separated by the "zero-char"</p> <p>simChar* inputCharBuff: pointer to all char buffer input arguments.</p> <p>simInt inputArgCount: number of input arguments.</p> <p>simInt* inputArgTypeAndSize: pointer to input argument's type and size (e.g. with "inputArgCount==2" we could have "inputArgTypeAndSize[0]==sim_lua_arg_int sim_lua_arg_table", "inputArgTypeAndSize[1]==3", "inputArgTypeAndSize[2]==sim_lua_arg_char", "inputArgTypeAndSize[3]==1". This would mean that we have two input arguments: (1) an integer table of size 3 and (2) a string)</p> <p>simChar* outputBool: similar to inputBool, but for output values. The user is in charge of allocating the buffer with correct size with simCreateBuffer (outputBool=new char[n] will not work!!). The simulator will automatically release the buffer when the callback returns.</p> <p>simInt* outputInt: similar to inputInt, but for output values. The user is in charge of allocating the buffer with correct size with simCreateBuffer (outputInt=new int[n] will not work!!). The simulator will automatically release the buffer when the callback returns.</p> <p>simFloat* outputFloat: similar to inputFloat, but for output values. The user is in charge of allocating the buffer with correct size with simCreateBuffer (outputFloat=new float[n] will not work!!). The simulator will automatically release the buffer when the callback returns.</p> <p>simChar* outputChar: similar to inputChar, but for output values. The user is in charge of allocating the buffer with correct size with simCreateBuffer (outputChar=new char[n] will not work!!). The simulator will automatically release the buffer when the callback returns. If you return 2 strings "ab" and "cde", the buffer should look like: "ab@cde@" (@ being the zero char).</p> <p>simChar* outputCharBuff: similar to inputCharBuff, but for output values. The user is in charge of allocating the buffer with correct size with simCreateBuffer (outputCharBuff=new char[n] will not work!!). The simulator will automatically release the buffer when the callback returns. If you return 2 char buffers, they should lay appended to each other in this buffer.</p> <p>simInt outputArgCount: similar to inputArgCount, but for output values</p> <p>simInt* outputArgTypeAndSize: similar to inputArgTypeAndSize, but for output values. The user is in charge of allocating the buffer with correct size with simCreateBuffer (outputArgTypeAndSize=new int[n] will not work!!). The simulator will automatically release the buffer when the callback returns.</p> <p>Values are stored in input or output arrays in the order they appear as arguments or return values e.g. with input arguments: number_int 1,table_2_float {2.0,3.0},string 'hello',number_int 4,string 'bye' we would have following input arrays:</p> <pre>inputBool==NULL inputInt=={1,4} inputFloat=={2.0,3.0} inputChar=="hello@bye@" inputArgCount==5 inputArgTypeAndSize==(sim_lua_arg_int,1,sim_lua_arg_float sim_lua_arg_table, 2,sim_lua_arg_string,1, sim_lua_arg_int,1, sim_lua_arg_string,1)</pre> <p>simChar waitUntilZero: DEPRECATED. Use simSwitchThread instead.</p> <p>Make sure to also have a look at the various plugin projects (in the installation directory) to get an idea how to register custom Lua commands</p>
C return value	1 if function was registered, 0 if function was replaced (when that function name already existed), -1 in case of an error
Lua synopsis	-
Lua parameters	-
Lua return values	-

simRegisterCustomLuaVariable

Description	Registers a customized lua variable. Lua doesn't directly support constants, so the variables registered with this function can be changed during runtime. However, at each simulation start, the variables registered with this function are reset to their original values. See also simRegisterCustomLuaFunction .
C synopsis	<code>simInt simRegisterCustomLuaVariable(const simChar* varName,const simChar* varValue)</code>
C parameters	varName: name of the variable varValue: value of the variable
C return value	1 if the variable was registered, 0 if the variable was replaced because it already existed, -1 in case of an error
Lua synopsis	-
Lua parameters	-
Lua return values	-

simRegisterJointCtrlCallback

Description	Registers or unregisters a joint control callback function for customized joint control handling. Several plugins can register a callback, and the callbacks will be cascaded if not interrupted.
C synopsis	<code>simInt simRegisterJointCtrlCallback(simInt(*callback)(simInt,simInt,simInt,const simInt*,const simFloat*,simFloat*))</code>
C parameters	<p>callback address. Specify the function you wish to register/unregister as callback. When the function is already registered, it will be unregistered. Following describes the callback's arguments:</p> <p>first argument (int): handle of the joint second argument (int): the selected physics engine third argument (int): a version indicator. Zero for now.</p> <p>The forth, fifth and sixth arguments are allocated by V-REP. Arguments 4 and 5 describe the joint state, argument 6 are the callback return values:</p> <p>forth argument (const int*) [0] bit-coded: 1=this callback is called for the first time (if the joint is dynamically reset during the simulation, this might be true more often) 2=the joint is revolute 4=the joint is cyclic (has no limits) [1] the current dynamics calculation pass. 0-9 by default. See next item for details [2] the number of dynamics calculation passes for each "regular" simulation pass. 10 by default (i.e. 10*5ms=50ms which is the default simulation time step)</p> <p>Fifth argument (const float*) [0] the current position of the joint [1] the desired position of the joint [2] error: targetPos-currentPos (with revolute cyclic joints we take the shortest cyclic distance) [3] the last force or torque that acted on this joint along/around its axis. With Bullet, torques from joint limits are not taken into account [4] the step size used for the dynamics calculations (by default 5ms) [5] the joint lower limit [6] the joint upper limit [7] the joint target velocity (as set in the user interface) [8] the joint maximum force/torque (as set in the user interface) [9] the joint velocity upper limit (as set in the user interface)</p> <p>Sixth argument (float*). If you want to control the joint from your callback, you need to set those values. [0] the maximum force/torque that the joint will be able to exert [1] the velocity to apply to the joint</p> <p>The callback return value can be: -1: the callback doesn't handle the joint and hands it to the next callback. If no callback handles it, the default joint controller handling routine is used (or the joint control callback script is called, if enabled) 0: the joint is free (no force/torque is applied) >0: the values of above's 6th argument will be applied to the joint (the default values (or the joint control callback script) are overridden)</p>
C return value	-1 if operation was not successful. 0 if the callback was unregistered, 1 if the callback was registered
Lua synopsis	-
Lua parameters	-
Lua return values	-

simReleaseBuffer (remote API equivalent: [simxReleaseBuffer](#))

Description	Releases a buffer previously created with simCreateBuffer or a buffer returned by the simulator. You will have to cast some buffer pointers to (simChar*).
C synopsis	<code>simInt simReleaseBuffer(simChar* buffer)</code>

C parameters	buffer: buffer to be released
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	-
Lua parameters	-
Lua return values	-

simReleaseScriptRawBuffer

Description	Removes (destroys) a script's raw buffer. All raw buffers are automatically released at the end of a simulation, but depending on the extent of use of the raw buffers, it might be a good idea to release them when not needed anymore. See also simGetScriptRawBuffer and simSetScriptRawBuffer .
C synopsis	simInt simReleaseScriptRawBuffer(simInt scriptHandle,simInt bufferHandle)
C parameters	scriptHandle: handle of the script. Can be sim_handle_main_script or sim_handle_all (in that case the bufferHandle is automatically also sim_handle_all) bufferHandle: handle of the raw buffer or sim_handle_all to release all raw buffers of the given script
C return value	-1 if operation was not successful (an unexisting buffer will not trigger an error). In a future release, a more differentiated return value might be available
Lua synopsis	number result=simReleaseScriptRawBuffer(number scriptHandle,number bufferHandle)
Lua parameters	scriptHandle: handle of the script. Can be sim_handle_main_script, sim_handle_self, or sim_handle_all, sim_handle_tree and sim_handle_chain (in the last 3 cases the bufferHandle is automatically also sim_handle_all). With sim_handle_tree or sim_handle_chain, the calling script is not included. bufferHandle: handle of the raw buffer or sim_handle_all to release all raw buffers of the given script
Lua return values	Same as C-function

simRemoveBanner

Description	Removes a previously added banner. See also simAddBanner
C synopsis	simInt simRemoveBanner(simInt bannerID)
C parameters	bannerID: handle of a previously added banner. sim_handle_all removes all banners from the scene
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simRemoveBanner(number bannerID)
Lua parameters	bannerID: handle of a previously added banner. sim_handle_all removes all banners from the scene, that were created in a script (banners created from the C interface are not removed)
Lua return values	Same as C-function

simRemoveCollection

Description	Removes a collection from the scene
C synopsis	simInt simRemoveCollection(simInt collectionHandle)
C parameters	collectionHandle: handle of the collection to remove. sim_handle_all removes all collections from the scene
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simRemoveCollection(number collectionHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simRemoveDrawingObject

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

simRemoveModel

remote API equivalent: [simxRemoveModel](#)
ROS API equivalent: [simRosRemoveModel](#)

Description	Removes a model from the scene. See also simRemoveObject .
C synopsis	simInt simRemoveModel(simInt objectHandle)
C parameters	objectHandle : handle of the model (i.e. object tagged as model) to remove.
C return value	-1 if operation was not successful, otherwise the number of removed objects (a model might contain several objects)
Lua synopsis	number removedCnt=simRemoveModel(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simRemoveObjectremote API equivalent: [simxRemoveObject](#)ROS API equivalent: [simRosRemoveObject](#)

Description	Removes an object from the scene. See also simRemoveModel .
C synopsis	simInt simRemoveObject(simInt objectHandle)
C parameters	objectHandle : handle of the object to remove. <code>sim_handle_all</code> removes all objects from the scene
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simRemoveObject(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simRemoveObjectFromSelection

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

simRemoveParticleObject

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

simRemoveScript

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

simRemoveUIremote API equivalent: [simxRemoveUI](#)

ROS API equivalent: [simRosRemoveUI](#)

Description	Removes a custom user interface. See also simCreateUI .
C synopsis	simInt simRemoveUI(simInt uiHandle)
C parameters	uiHandle : handle of the custom user interface to remove. sim_handle_all removes all custom user interfaces
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=simRemoveUI(number uiHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simReorientShapeBoundingBox

Description	Reorients the bounding box of a shape.
C synopsis	simInt simReorientShapeBoundingBox(simInt shapeHandle,simInt relativeToHandle,simInt reservedSetToZero)
C parameters	shapeHandle : the handle of the shape that will be reoriented. See also simGetObjectHandle . relativeToHandle : the handle of an object relative to which the shape should be reoriented, or -1 to align the bounding box with the world, or sim_handle_self to build the smallest bounding box around the object. reservedSetToZero : reserved for future extensions. Set to 0.
C return value	-1 if operation was not successful. 0 if the bounding box could not be reoriented (the bounding box of pure shapes cannot be reoriented), otherwise 1.
Lua synopsis	number result=simReorientShapeBoundingBox(number shapeHandle,number relativeToHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simResetCollision

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

simResetDistance

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

simResetDynamicObject

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

Lua synopsis	number result=simResetDynamicObject(number objectHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simResetGraph

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

simResetJoint

Description	Resets a joint's motion parameters as they were before starting the simulation. Has an effect only for joint in motion mode. See also simHandleJoint .
C synopsis	simInt simResetJoint(simInt jointHandle)
C parameters	jointHandle : handle of the joint object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will reset all joint objects, while sim_handle_all_except_explicit will only reset those that are not marked as "explicit handling")
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simResetJoint(number jointHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simResetMill

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

simResetMilling

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

simResetPath

Description	Resets the motion parameters along a path object as they were before starting the simulation. See also simHandlePath .
C synopsis	simInt simResetPath(simInt pathHandle)
C parameters	pathHandle : handle of the path object or sim_handle_all or sim_handle_all_except_explicit. (sim_handle_all will reset all path objects, while sim_handle_all_except_explicit will only reset those that are not marked as "explicit handling")
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available

Lua synopsis	number result=simResetPath(number pathHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simResetProximitySensor

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

simResetScript

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

simResetVisionSensor

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

simResumeThreads

Description	In conjunction with simSetThreadResumeLocation , simResumeThreads allows specifying when and in which order threads are resumed. By default, V-REP doesn't use "regular" threads, but something similar to hybrid threads (which behave like coroutines, but can also behave like regular threads). This allows much more flexibility and execution control of the threads. Once a thread switched to another thread, it will resume execution at the beginning of next simulation pass by default. In order to also have full synchronization control between threads, you can assign a resume location and order to each thread. When simResumeThreads(x) is called, all threads that were assigned a resume location of x will be resumed. See also simSetThreadResumeLocation , simSetThreadSwitchTiming , simSwitchThread and simSetThreadIsFree . This function can only be called in the main script .
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number count=simResumeThreads(number location)
Lua parameters	location: indicates a location value associated with threads (through the simSetThreadResumeLocation function). Only threads with the same location value will be resumed.
Lua return values	result: number of resumed threads, or -1 in case of an error.

simRMLMoveToJointPositions

Description	Moves (actuates) several joints at the same time using the Reflexxes Motion Library type IV . This function can only be called from child scripts running in a thread (since this is a blocking operation) and is not available from the C-API. See also simRMLMoveToPosition , and simRMLPos .
C synopsis	-
C parameters	-
C return value	-

Lua synopsis	number result,table newPos,table newVel,table newAccel,number timeLeft=simRMLMoveToJointPositions(table jointHandles,number flags,table currentVel,table currentAccel,table maxVel,table maxAccel,table maxJerk,table targetPos,table targetVel,table direction)
Lua parameters	jointHandles : handles of the joints to actuate flags : RML flags. -1 for default flags. currentVel : the current velocity of the joints. Can be nil in which case a velocity vector of 0 is used. currentAccel : the current acceleration of the joints. Can be nil in which case an acceleration vector of 0 is used. maxVel : the maximum allowed velocity of the joints maxAccel : the maximum allowed acceleration of the joints maxJerk : the maximum allowed jerk of the joints targetPos : the desired target positions of the joints targetVel : the desired velocity of the joints at the target. Can be nil in which case a velocity vector of 0 is used. direction : the desired rotation direction for cyclic revolute joints: 0 for the shortest distance, -x for a movement towards negative values, +x for a movement towards positive values (n=(x-1) represents the number of additional turns). Can be nil or omitted, in which case a value of 0 is used for all joints.
Lua return values	result : 1 if the function call was successful newPos : the new positions of the joints newVel : the new velocities of the joints newAccel : the new accelerations of the joints timeLeft : the time left for additional calculations in current simulation time step

simRMLMoveToPosition

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

simRMLPos

Description	Executes a call to the Reflexxes Motion Library type IV . The Reflexxes Motion Library type IV provides instantaneous trajectory generation capabilities for motion control systems. This function prepares a position-based trajectory generation object, that can then be calculated with simRMLStep . When this object is not needed anymore, remove it with simRMLRemove . See also simRMLVel , simRMLMoveToPosition and simRMLMoveToJointPositions .
C synopsis	simInt simRMLPos(simInt dofs,simDouble smallestTimeStep,simInt flags,const simDouble* currentPosVelAccel,const simDouble* maxVelAccelJerk,const simBool* selection,const simDouble* targetPosVel,simVoid* auxData)
C parameters	dofs : the number of degrees of freedom (N). smallestTimeStep : the smallest expected cycle time. Use a value of 0.0001 (0.1ms). flags : RML flags. -1 for default flags. currentPosVelAccel : the current position, velocity and acceleration. Arrange values as

	<p>{pos1,pos2,...,posN,vel1,vel2,...,velN,accel1,accel2,...,accelN}</p> <p>maxVelAccelJerk: the maximum allowed velocity, acceleration and jerk. Arrange values as {vel1,vel2,...,velN,accel1,accel2,...,accelN,jerk1,jerk2,...,jerkN}</p> <p>selection: the selection vector (one value for each DoF). For a default behaviour, fill the vector with non-zero values.</p> <p>targetPosVel: the target position and velocity. Arrange values as {pos1,pos2,...,posN,vel1,vel2,...,velN}</p> <p>auxData: can be NULL. Otherwise in/out extension data. The first byte indicates how many additional in/out values we wish to set/get. Following auxiliary values can be set/get:</p> <p>value 1 (input): Bytes 2-5 (int): set to 1 if you wish this object to be automatically destroyed at simulation end (which is the default case when called from the main script or a child script).</p>
C return value	-1 in case of an error, otherwise the handle of the created object.
Lua synopsis	number handle=simRMLPos(number dofs,number smallestTimeStep,number flags,table currentPosVelAccel,table maxVelAccelJerk,table selection,table targetPosVel)
Lua parameters	Refer to the C-function documentation
Lua return values	Refer to the C-function documentation

simRMLPosition (DEPRECATED)

Description	DEPRECATED. See simRMLPos instead.
-------------	--

simRMLRemove

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

simRMLStep

Description	Executes a call to the Reflexxes Motion Library type IV . The Reflexxes Motion Library type IV provides instantaneous trajectory generation capabilities for motion control systems. This function steps forward a trajectory generation algorithm previously prepared via simRMLPos or simRMLVel .
C synopsis	simInt simRMLStep(simInt handle,simDouble timeStep,simDouble* newPosVelAccel,simVoid* auxData,simVoid* reserved)
C parameters	<p>handle: the handle of the object created via simRMLPos or simRMLVel.</p> <p>timeStep: the cycle time</p> <p>newPosVelAccel: the new position, velocity and acceleration (output values). Values are arranged as {pos1,pos2,...,posN,vel1,vel2,...,velN,accel1,accel2,...,accelN}</p> <p>auxData: can be NULL. Otherwise in/out extension data. The first byte indicates how many additional in/out values we wish to set/get. Following auxiliary values can be set/get:</p> <p>value 1 (output): Bytes 2-9 (double): returns the synchronization time (the time needed to reach the desired state. This time does not include the cycle time of the current call to simRMLStep)</p> <p>reserved: reserved for future extensions. Set to NULL.</p>
C return value	-1 in case of an error, otherwise the return value of function RMLPosition or RMLVelocity in the library (0=final state not yet reached, 1=final state reached, otherwise an error code)
Lua synopsis	number result,table newPosVelAccel,number synchronizationTime=simRMLStep(number handle,number timeStep)
Lua parameters	Refer to the C-function documentation
Lua return values	Refer to the C-function documentation

simRMLVel

Description	Executes a call to the Reflexxes Motion Library type IV . The Reflexxes Motion Library type IV provides instantaneous trajectory generation capabilities for motion control systems. This function prepares a velocity-based trajectory generation object, that can then be calculated with simRMLStep . When this object is not needed anymore, remove it with simRMLRemove . See also simRMLPos .
C synopsis	simInt simRMLVel(simInt dofs,simDouble smallestTimeStep,simInt flags,const simDouble* currentPosVelAccel,const simDouble* maxAccelJerk,const simBool* selection,const simDouble* targetVel,simVoid* auxData)
C parameters	<p>dofs: the number of degrees of freedom (N).</p> <p>smallestTimeStep: the smallest expected cycle time. Use a value of 0.0001 (0.1ms).</p> <p>flags: RML flags. -1 for default flags.</p>

	<p>currentPosVelAccel: the current position, velocity and acceleration. Arrange values as {pos1,pos2,...,posN,vel1,vel2,...,velN,accel1,accel2,...,accelN}</p> <p>maxAccelJerk: the maximum allowed acceleration and jerk. Arrange values as {accel1,accel2,...,accelN,jerk1,jerk2,...,jerkN}</p> <p>selection: the selection vector (one value for each DoF). For a default behaviour, fill the vector with non-zero values.</p> <p>targetVel: the target velocity (one value for each DoF)</p> <p>auxData: can be NULL. Otherwise in/out extension data. The first byte indicates how many additional in/out values we wish to set/get. Following auxiliary values can be set/get: value 1 (input): Bytes 2-5 (int): set to 1 if you wish this object to be automatically destroyed at simulation end (which is the default case when called from the main script or a child script).</p>
C return value	-1 in case of an error, otherwise the handle of the created object.
Lua synopsis	number handle=simRMLVel(number dofs,number smallestTimeStep,number flags,table currentPosVelAccel,table maxAccelJerk,table selection,table targetVel)
Lua parameters	Refer to the C-function documentation
Lua return values	Refer to the C-function documentation

simRMLVelocity (DEPRECATED)

Description	DEPRECATED. See simRMLVel instead.
-------------	--

simRotateAroundAxis

Description	Rotates a transformation matrix around a random axis in space. This function, when used in combination with simGetRotationAxis , can be used to build interpolations between transformation matrices. See also simGetObjectMatrix and simSetObjectMatrix
C synopsis	simInt simRotateAroundAxis(const simFloat* matrixIn,const simFloat* axis,const simFloat* axisPos,simFloat angle,simFloat* matrixOut)
C parameters	<p>matrixIn: the transformation matrix to rotate</p> <p>axis: the axis vector in absolute coordinates to rotate around</p> <p>axisPos: the position of the rotation axis in absolute coordinates</p> <p>angle: the amount of rotation to perform</p> <p>matrixOut: the returned transformed (rotated) matrix</p>
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=simRotateAroundAxis(table_12 matrixIn,table_3 axis,table_3 axisPos,number angle)
Lua parameters	Same as C-function
Lua return values	matrixOut: the transformed (rotated) matrix, or nil in case of an error

simRunSimulator

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

simSaveModel

Description	Saves a model (an object marked as "Object is model base" and all other objects in its hierarchy tree). Any existing file with same name will be overwritten. See also simLoadModel , simSaveUI and simSaveScene .
C synopsis	simInt simSaveModel(int baseOfModelHandle,const simChar* filename)
C parameters	<p>baseOfModelHandle: handle of an object marked as "Object is model base"</p> <p>filename: model filename. The filename extension is required (".ttm")</p>
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=simSaveModel(number baseOfModelHandle,string filename)

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

simSaveScene

Description	Saves a scene. Any existing file with same name will be overwritten. See also simLoadScene , simCloseScene , simSaveModel and simSaveUI .
C synopsis	simInt simSaveScene(const simChar* filename)
C parameters	filename : scene filename. The filename extension is required (".tst")
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=simSaveScene(string filename)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSaveUI

Description	Saves one or several custom user interfaces into a UI file (*.ttb-file). Any existing file with same name will be overwritten. See also simLoadUI , simSaveModel and simSaveScene .
C synopsis	simInt simSaveUI(int count,const int* uiHandles,const simChar* filename)
C parameters	count : the number of UIs to save. Can be 0, in which case all UIs in the scene will be saved uiHandles : a pointer to UI handles to save. Can be NULL if count is 0 filename : UI filename. The filename extension is required (".ttb")
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=simSaveUI(table uiHandles,string filename)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simScaleObject

Description	Scales specified objects in a non-isometric fashion, if possible (i.e. some objects can be fully isometrically scaled, others have 2 or all 3 axes linked). See also simScaleObjects for isometric scaling.
C synopsis	simInt simScaleObject(simInt objectHandle,simFloat xScale,simFloat yScale,simFloat zScale,simInt options)
C parameters	objectHandle : the handle of the object to scale. xScale/yScale/zScale : the scaling factors along the object's x, y and z-axis. options : reserved for future extension. Keep at 0.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simScaleObject(number objectHandle,number xScale,number yScale,number zScale,number options=0)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simScaleObjects

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

simScaleSelectedObjects (DEPRECATED)

Description	DEPRECATED. See simScaleObjects instead.
-------------	--

simSearchPath

Description	Searches for a path. A registered path planning object is required (path planning objects can be
-------------	--

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

simSendData

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

	emissionAngle2 : same as C-function. Can be omitted (in that case, 2pi is used) persistence : same as C-function. Can be omitted (in that case, 0.0 is used)
Lua return values	Same as C-function

simSendMessage

Description	Sends a message to plugins. This function should normally only be used from the main client application side. See also simLoadModule and simBroadcastMessage .
C synopsis	simVoid* simSendMessage(simInt message,simInt* auxiliaryData,simVoid* customData,simInt* replyData)
C parameters	message : the message to send. Refer to sim_message_eventcallback -type messages. auxiliaryData : pointer to 4 integers. auxiliaryData[0] should be a unique identifier different from 0. Use 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.
C return value	Pointer to custom reply data that can be used by a module to reply to a broadcasted message. The module that replies is in charge of allocating the data with simCreateBuffer and the original broadcaster is in charge of releasing that data with simReleaseBuffer . A reply to a broadcasted message is triggered by a module that writes a value different from -1 into auxiliaryData[0]-auxiliaryData[3], thus aborting further broadcast of the original message and returning to the broadcaster. If the return value is different from NULL, the broadcast is also interrupted.
Lua synopsis	-
Lua parameters	-
Lua return values	-

simSerialCheck

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

simSerialClose

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

simSerialOpen

Description	Opens a serial port (RS-232) for communication. When called from a script, the function can only be called when the simulation is running (and in that case the port is automatically closed at simulation stop). See also simSerialClose , simSerialSend , simSerialCheck and simSerialRead .
C synopsis	simInt simSerialOpen(simChar* portString,simInt baudRate,simVoid* reserved1,simVoid* reserved2)
C parameters	portString : a string specifying the port to open. Under Windows, use something similar to "\\.\COM1". Under MacOS and Linux, use something similar to "/dev/*" (check the "/dev" folder to know what file to specify). Under Linux, you might have to launch V-REP with super user privileges in order to access the serial port. baudRate : the baudrate reserved1 : reserved for future extension. Keep at NULL. reserved2 : reserved for future extension. Keep at NULL.
C return value	-1 if operation was not successful, otherwise a port handle
Lua synopsis	number result=simSerialOpen(string portString,number baudRate)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSerialRead

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

simSerialSend

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

simSetArrayParameter

remote API equivalent: [simxSetArrayParameter](#)
ROS API equivalent: [simRosSetArrayParameter](#)

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

simSetBooleanParameter

remote API equivalent: [simxSetBooleanParameter](#)
ROS API equivalent: [simRosSetBooleanParameter](#)

Description	Sets a boolean parameter . See also simGetBooleanParameter , simSetIntegerParameter , simSetArrayParameter and simSetFloatingParameter .
C synopsis	simInt simSetBooleanParameter(simInt parameter,simBool boolState)
C parameters	parameter : Boolean parameter identifier boolState : new boolean state for the parameter
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simSetBooleanParameter(number parameter,boolean boolState)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetCollectionName

Description	Sets the name of a collection based on its handle. See also simGetCollectionName .
C synopsis	simInt simSetCollectionName(simInt collectionHandle,const simChar* collectionName)

C parameters	collectionHandle: handle of the collection collectionName: new name of the collection
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simSetCollectionName(number collectionHandle,string collectionName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetConfigurationTree

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

simSetExplicitHandling

Description	Sets the explicit handling flags for a general object. See also simGetExplicitHandling .
C synopsis	simInt simSetExplicitHandling(simInt generalObjectHandle,int explicitFlags)
C parameters	generalObjectHandle: handle of a general object (can be a scene object, a collision object, a distance object, etc.) explicitFlags: the explicit handling flags. For now only bit 0 is used
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simSetExplicitHandling(number generalObjectHandle,number explicitFlags)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetFloatingParameter

remote API equivalent: [simxSetFloatingParameter](#)
ROS API equivalent: [simRosSetFloatingParameter](#)

Description	Sets a floating point parameter . See also simGetFloatingParameter , simSetBooleanParameter , simSetArrayParameter and simSetIntegerParameter .
C synopsis	simInt simSetFloatingParameter(simInt parameter,simFloat floatState)
C parameters	parameter: floating parameter identifier floatState: new state for the parameter
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simSetFloatingParameter(number parameter,number floatState)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetFloatSignal

remote API equivalent: [simxSetFloatSignal](#)
ROS API equivalent: [simRosSetFloatSignal](#)

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

simSetGraphUserData

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

simSetIkElementProperties

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

simSetIkGroupProperties

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

simSetIntegerParameter

remote API equivalent: [simxSetIntegerParameter](#)
ROS API equivalent: [simRosSetIntegerParameter](#)

Description	Sets an integer parameter . See also simGetIntegerParameter , simSetBooleanParameter , simSetArrayParameter and simSetFloatingParameter .
C synopsis	simInt simSetIntegerParameter(simInt parameter,simInt intState)
C parameters	parameter : integer parameter identifier intState : new state for the parameter
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simSetIntegerParameter(number parameter,number intState)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetIntegerSignal

remote API equivalent: [simxSetIntegerSignal](#)
ROS API equivalent: [simRosSetIntegerSignal](#)

Description	Sets the value of an integer signal. If that signal is not yet present, it is added. Signals created in the
-------------	---

	main script or in a child script are automatically cleared at simulation end. See also simGetIntegerSignal , simClearIntegerSignal , simSetFloatSignal , simSetStringSignal , simWaitForSignal and simPersistentDataWrite .
C synopsis	simInt simSetIntegerSignal(const simChar* signalName,simInt signalValue)
C parameters	signalName : name of the signal signalValue : value of the signal
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simSetIntegerSignal(string signalName,number signalValue)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetJointForce

remote API equivalent: [simxSetJointForce](#)
ROS API equivalent: [simRosSetJointForce](#)

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

simSetJointInterval

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

simSetJointMode

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

simSetJointPosition

remote API equivalent: [simxSetJointPosition](#)
ROS API equivalent: [simRosSetJointPosition](#)

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

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

simSetJointTargetPositionremote API equivalent: [simxSetJointTargetPosition](#)ROS API equivalent: [simRosSetJointTargetPosition](#)

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

simSetJointTargetVelocityremote API equivalent: [simxSetJointTargetVelocity](#)ROS API equivalent: [simRosSetJointTargetVelocity](#)

Description	Sets the intrinsic target velocity of a non-spherical joint. This command makes only sense when the joint mode is: (a) motion mode: the joint's motion handling feature must be enabled (simHandleJoint must be called (is called by default in the main script), and the joint motion properties must be set in the joint settings dialog), (b) torque/force mode: the dynamics functionality and the joint motor have to be enabled (position control should however be disabled). See also simGetJointTargetVelocity .
C synopsis	simInt simSetJointTargetVelocity(simInt objectHandle,simFloat targetVelocity)
C parameters	objectHandle : handle of the joint object targetVelocity : target velocity of the joint (linear or angular velocity depending on the joint-type).
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simSetJointTargetVelocity(number objectHandle,number targetVelocity)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetLastError

Description	Sets a custom error message. This function is useful for plugins which wish to generate custom error messages. Errors are set and memorized on a thread-basis (e.g. threads originating from threaded scripts have each an individual error handler). See also simGetLastError , the <code>sim_intparam_error_report_mode</code> and the error report modes .
C synopsis	simInt simSetLastError(const simChar* funcName,const simChar* errorMessage)
C parameters	funcName : name of the function where the error originated errorMessage : error message
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	-
Lua parameters	-
Lua return values	-

simSetLightParameters

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

simSetLinkDummy

Description	Defines (or breaks) a dummy-dummy link pair. Useful to create dynamic loop closure constraints on the
-------------	---

	fly (among others). See also simGetLinkDummy .
C synopsis	simInt simSetLinkDummy(simInt dummyHandle,simInt linkedDummyHandle)
C parameters	dummyHandle : handle of the first dummy in the dummy-dummy link pair. linkedDummyHandle : handle of the second dummy in the dummy-dummy link pair. Set to -1 to unlink the first dummy.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simGetLinkDummy(number dummyHandle,number linkedDummyHandle)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetModelProperty

remote API equivalent: [simxSetModelProperty](#)
ROS API equivalent: [simRosSetModelProperty](#)

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

simSetModuleMenuItemState

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

simSetNameSuffix

Description	<p>Sets the name suffix adjustment number (for detailed information on this, read also the section on accessing general-type objects). In V-REP, all objects are identified by a name and a handle. When an object (scene object or general-type object) is copied at the same time as a child script, the newly created object's name will become "oldName#0", should the same object be pasted another time, the next name will be "oldName#1", etc.</p> <p>From within a child script, retrieving object handles is performed by automatically appending a name suffix to the object name (each script gets initialized with the name suffix number of the object it is attached to). This allows to copy-paste objects and scripts without having to manually adjust the scripts (the scripts will automatically know which object they have to access based on the set name suffix). From within a script, most of the time you won't need to set the name suffix, but in some special cases you might want to temporarily disable it (e.g. "myChildScript#42" (which has its name suffix automatically set to 42) copied itself together with its attached robot ("myRobot#42") and now from within "myChildScript#42" you want to shift "myRobot#43" to avoid collision. In that case set the name suffix to 43, shift "myRobot" (retrieve its handle with simGetObjectHandle("myRobot") then set the suffix back to 42). From within a script, the simSetNameSuffix command is influencing only current script.</p> <p>When accessing the API from outside of a script however, the name adjustment mechanism needs to be adjusted manually (make sure you reset the name suffix to its initial state after you are done retrieving handles). Imagine you have one robot in your scene that is named "robot". You can access the robot from a C/C++ application with simGetObjectHandle("robot"). If the robot is duplicated, the second robot's name will be "robot#0", the third will be "robot#1", etc. From within you C/C++ application you can now access all robots with the same code, you just need to adjust the name suffix number. E.g. simSetNameSuffix(42), then simGetObjectHandle("robot") will retrieve the handle of "robot#42". Once you are done accessing objects, reset the name suffix number to -1 (simSetNameSuffix(-1)).</p> <p>Setting the name suffix to -1 disables the name adjustment mechanism (default when accessing the API from outside of a script)</p> <p>See also the simGetNameSuffix function.</p>
C synopsis	simInt simSetNameSuffix(simInt nameSuffixNumber)
C parameters	nameSuffixNumber : a number starting from -1 (-1 is for no suffix, 0 is for the 0 suffix, etc.)

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

simSetNavigationMode

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

simSetObjectConfiguration

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

simSetObjectFloatParameter

remote API equivalent: [simxSetObjectFloatParameter](#)
 ROS API equivalent: [simRosSetObjectFloatParameter](#)

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

simSetObjectIntParameter

remote API equivalent: [simxSetObjectIntParameter](#)
 ROS API equivalent: [simRosSetObjectIntParameter](#)

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

simSetObjectMatrix

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

simSetObjectName

Description	Sets the name of an object based on its handle. See also simGetObjectName .
C synopsis	simInt simSetObjectName(simInt objectHandle,const simChar* objectName)
C parameters	objectHandle : handle of the object objectName : name of the object
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simSetObjectName(number objectHandle,string objectName)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetObjectOrientation

remote API equivalent: [simxSetObjectOrientation](#)
ROS API equivalent: [simRosSetObjectQuaternion](#)

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

simSetObjectParent

remote API equivalent: [simxSetObjectParent](#)
ROS API equivalent: [simRosSetObjectParent](#)

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

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

simSetObjectPosition

remote API equivalent: [simxSetObjectPosition](#)
ROS API equivalent: [simRosSetObjectPosition](#)

Description	Sets the position (x, y and z-coordinates) of an object. Dynamically simulated objects will implicitly be reset before the command is applied (i.e. similar to calling simResetDynamicObject just before). See also simGetObjectPosition , simSetObjectOrientation and simSetObjectMatrix .
C synopsis	simInt simSetObjectPosition(simInt objectHandle,simInt relativeToObjectHandle,const simFloat* position)
C parameters	objectHandle : handle of the object relativeToObjectHandle : indicates relative to which reference frame the position is specified. Specify -1 to set the absolute position, sim_handle_parent to set the position relative to the object's parent, or an object handle relative to whose reference frame the position is specified. position : coordinates of the object (x, y and z)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simSetObjectPosition(number objectHandle,number relativeToObjectHandle,table_3 position)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetObjectProperty

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

simSetObjectQuaternion (ROS API equivalent: [simRosSetObjectQuaternion](#))

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

simSetObjectSizeValues

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

simSetObjectSpecialProperty

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

simSetObjectStringParameter

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

simSetPage

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

simSetPathPosition

Description	Sets the intrinsic position of a path object (i.e. the position along the path). The position is given in meters, but the actual position is dependent on the selected path length calculation method for the given path object. This function is the equivalent of simSetJointPosition , but for a path object. See also simGetPathPosition .
C synopsis	simInt simSetPathPosition(simInt objectHandle,simFloat position)
C parameters	objectHandle : handle of the path object position : linear position on the path given in meters (but dependent on the selected path length calculation method)
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simSetPathPosition(number objectHandle,number position)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetPathTargetNominalVelocity

Description	Sets the intrinsic target nominal velocity of a path object. The path's motion handling feature must be enabled for this to have an effect (i.e. simHandlePath must be called and the path motion properties be set in the path settings dialog). The path intrinsic velocity is expressed in meters/seconds, but depending on the selected path length calculation method , that velocity might be different from a "regular" linear velocity.
C synopsis	simInt simSetPathTargetNominalVelocity(simInt objectHandle,simFloat targetNominalVelocity)
C parameters	objectHandle : handle of the path object targetNominalVelocity : the target nominal velocity (which is the velocity we wish, but that velocity might be modulated between the path's Bezier points depending on the selected velocity profile). Moreover, the targetNominalVelocity is expressed in meters/seconds, but depending on the selected path length calculation method, that velocity might be different from a "regular" linear velocity.
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=simSetPathTargetNominalVelocity(number objectHandle,number targetNominalVelocity)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetScriptAttribute

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

simSetScriptRawBuffer

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

simSetScriptSimulationParameter

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

simSetScriptText

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

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

simSetShapeColor

Description	Sets the color (or transforms it) of one or several shapes. See also simGetShapeColor .
C synopsis	simInt simSetShapeColor(simInt shapeHandle,simChar* colorName,simInt colorComponent,simFloat* rgbData)
C parameters	<p>shapeHandle: handle of the shape, or <code>sim_handle_all</code> if the command should be directed at all shapes</p> <p>colorName: name of a color. Can be NULL, but if a name is provided, only shapes (or sub-entities of them) with a same color name will be modified. By specifying special names, the color can directly be transformed in the Hue-Saturation-Lightness (HSL) space:</p> <p>"@0": all specified shape outside colors will be transformed such as: HSL={H+rgbData[0],S+rgbData[1],L+rgbData[2]}, where H operates in a cyclic manner.</p> <p>"@1": all specified shape inside colors will be transformed such as: HSL={H+rgbData[0],S+rgbData[1],L+rgbData[2]}, where H operates in a cyclic manner.</p> <p>"@2": all specified shape edge colors will be transformed such as: HSL={H+rgbData[0],S+rgbData[1],L+rgbData[2]}, where H operates in a cyclic manner.</p> <p>colorComponent: a color component</p> <p>rgbData: red, green and blue components of the color (3 values), or the transparency value (1 value)</p>
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=simSetShapeColor(number shapeHandle,string colorName,number colorComponent,table_3 rgbData)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetShapeMassAndInertia

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

simSetShapeMaterial

Description	Sets the material ID of a specific shape. You might have to also call simResetDynamicObject for changes to take effect. See also simGetMaterialId and simGetShapeMaterial .
C synopsis	simInt simSetShapeMaterial(simInt shapeHandle,simInt materialId)
C parameters	<p>shapeHandle: the handle of the shape. See also simGetObjectHandle.</p> <p>materialId: the ID of the material. See also simGetMaterialId.</p>
C return value	-1 in case of an error.
Lua synopsis	number result=simSetShapeMaterial(number shapeHandle,number materialId)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetShapeTexture

Description	Applies (or removes) a texture to a shape. See also simGetTextureId , simGetShapeTextureId and simCreateTexture .
C synopsis	simInt simSetShapeTexture(simInt shapeHandle,simInt textureId,simInt mappingMode,simInt options,const simFloat* uvScaling,const simFloat* position,const simFloat* orientation)
C parameters	<p>shapeHandle: the handle of the shape.</p> <p>textureId: the ID of the texture. See also simGetTextureId, simGetShapeTextureId and simCreateTexture.</p> <p>mappingMode: the texture mapping mode.</p> <p>options: bit-coded:</p>

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

simSetSimulationPassesPerRenderingPass

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

simSetSphericalJointMatrix

remote API equivalent: [simxSetSphericalJointMatrix](#)
ROS API equivalent: [simRosSetSphericalJointMatrix](#)

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

simSetStringParameter

Description	Sets a string parameter . See also simGetStringParameter , simSetBooleanParameter , simSetArrayParameter , simSetFloatingParameter and simSetIntegerParameter .
C synopsis	simInt simSetStringParameter(simInt parameter,const simChar* stringState)
C parameters	parameter: string parameter identifier stringState: new state for the parameter
C return value	1 if operation was successful. -1 if parameter is not known
Lua synopsis	number result=simSetStringParameter(number parameter,string stringState)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetStringSignal

remote API equivalent: [simxSetStringSignal](#)
ROS API equivalent: [simRosSetStringSignal](#)

Description	Sets the value of a string signal. If that signal is not yet present, it is added. Signals created in the main script or in a child script are automatically cleared at simulation end. See also simGetStringSignal , simClearStringSignal , simSetIntegerSignal , simSetFloatSignal , simWaitForSignal and simPersistentDataWrite .
C synopsis	simInt simSetStringSignal(const simChar* signalName,const simChar* signalValue,simInt stringLength)
C parameters	signalName: name of the signal signalValue: value of the signal (which may contain any value, including embedded zeros)

	stringLength: the size of the string value.
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simSetStringSignal(string signalName,string signalValue)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetThreadAutomaticSwitch

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

simSetThreadIsFree

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

simSetThreadResumeLocation

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

simSetThreadSwitchTiming

Description	Allows specifying a switching time for the thread in which the current script runs. If the current script doesn't run in a thread (i.e. if it runs in the application main thread), this function has no effect. By default, V-REP doesn't use "regular" threads, but something similar to hybrid threads (which behave like coroutines, but can also behave like regular threads). This allows much more flexibility and execution control of the threads: each thread (except for the main or application thread) has a switch timing associated, which specifies how long the thread will run before switching to other threads (the execution
-------------	---

	duration per calculation pass). By default this value is 2 millisecond, but this function allows changing that value (on a thread-basis). Acceptable values are between 0 and 200. For complete control over the switching moment, see also simSetThreadAutomaticSwitch , simSwitchThread , simSetThreadIsFree and simSetThreadResumeLocation .
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	number result=simSetThreadSwitchTiming(number deltaTimeInMilliseconds)
Lua parameters	deltaTimeInMilliseconds : desired non-stop execution time before a switching occurs. A value of x will let the thread execute for x-1 to x milliseconds before switching to another thread.
Lua return values	result : 1 if the timing was set, 0 if it was not set (e.g. because the function was called from the main or application thread), or -1 in case of an error.

simSetUIButtonArrayColor

Description	Sets the color of an array element of a custom user interface. simCreateUIButtonArray needs to be called previously
C synopsis	simInt simSetUIButtonArrayColor(simInt uiHandle,simInt buttonHandle,const simInt* position,const simFloat* color)
C parameters	uiHandle : custom user interface handle buttonHandle : button handle position : position in the array from left to right and top to bottom (x and y) color : color of the array element (red, green and blue values, 0=minimum intensity, 1=maximum intensity)
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=simSetUIButtonArrayColor(number uiHandle,number buttonHandle,table_2 position,table_3 color)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetUIButtonColor

Description	Sets the up-state, down-state and label colors of a custom user interface button
C synopsis	simInt simSetUIButtonColor(simInt uiHandle,simInt buttonHandle,const simFloat* upStateColor,const simFloat* downStateColor,const simFloat* labelColor)
C parameters	uiHandle : custom user interface handle buttonHandle : button handle upStateColor : color of the button when up (red, green and blue values, 0=minimum intensity, 1=maximum intensity). Can be NULL downStateColor : color of the button when down (red, green and blue values, 0=minimum intensity, 1=maximum intensity). Can be NULL labelColor : color of the button label (red, green and blue values, 0=minimum intensity, 1=maximum intensity). 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=simSetUIButtonColor(number uiHandle,number buttonHandle,table_3 upStateColor,table_3 downStateColor,table_3 labelColor)
Lua parameters	uiHandle : Same as C-function buttonHandle : Same as C-function upStateColor : Same as C-function. Can be nil or omitted downStateColor : Same as C-function. Can be nil or omitted labelColor : Same as C-function. Can be nil or omitted
Lua return values	Same as C-function

simSetUIButtonLabel

remote API equivalent: [simxSetUIButtonLabel](#)
ROS API equivalent: [simRosSetUIButtonLabel](#)

Description	Sets the up-state and down-state labels of a custom user interface button. See also simGetUIButtonLabel .
C synopsis	simInt simSetUIButtonLabel(simInt uiHandle,simInt buttonHandle,const simChar* upStateLabel,const simChar* downStateLabel)
C parameters	uiHandle : custom user interface handle buttonHandle : button handle upStateLabel : pointer to a string containing the label of the button when it is up. Can be NULL. downStateLabel : pointer to a string containing the label of the button when it is down. 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=simSetUIButtonLabel(number uiHandle,number buttonHandle,string upStateLabel,string downStateLabel)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetUIButtonProperty

remote API equivalent: [simxSetUIButtonProperty](#)
 ROS API equivalent: [simRosSetUIButtonProperty](#)

Description	Sets the properties of a custom user interface button. See also simGetUIButtonProperty .
C synopsis	simInt simSetUIButtonProperty(simInt uiHandle,simInt buttonHandle,simInt buttonProperty)
C parameters	uiHandle: custom user interface handle buttonHandle: button handle buttonProperty: button property. See the button property values . Combine them with the "or"-operator
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simSetUIButtonProperty(number uiHandle,number buttonHandle,number buttonProperty)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetUIButtonTexture

Description	Sets (or actualizes) the texture of a custom user interface button. This function is useful to display custom dynamic images (e.g. video camera images) inside of a custom user interface button. See also simSetVisionSensorImage .
C synopsis	simInt simSetUIButtonTexture(simInt uiHandle,simInt buttonHandle,const simInt* size,const simChar* textureData)
C parameters	uiHandle: handle of the custom user interface buttonHandle: handle of the button size: pointer to the x- and y-resolution of the texture. Can be NULL, in which case the texture is removed from the button. textureData: rgb data of the texture. Texture data will be copied internally, so the textureData can be released directly 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	-
Lua parameters	-
Lua return values	-

simSetUIPosition

Description	Sets the position of a custom user interface. See also simGetUIPosition .
C synopsis	simInt simSetUIPosition(simInt uiHandle,const simInt* position)
C parameters	uiHandle: handle of the custom user interface position: pointer to 2 values (x and y)
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=simSetUIPosition(number uiHandle,table_2 position)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetUIProperty

Description	Sets the properties of a custom user interface. See also simGetUIProperty .
C synopsis	simInt simSetUIProperty(simInt uiHandle,simInt uiProperty)
C parameters	uiHandle: custom user interface handle uiProperty: custom user interface property. See the custom user interface properties . Combine them with the "or"-operator
C return value	-1 if operation was not successful. In a future release, a more differentiated return value might be available
Lua synopsis	number result=simSetUIProperty(number uiHandle,number uiProperty)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetUISlider

remote API equivalent: [simxSetUISlider](#)
 ROS API equivalent: [simRosSetUISlider](#)

Description	Sets the slider position of a custom user interface button (must be a slider-type button). See also simGetUISlider .
C synopsis	simInt simSetUISlider(simInt uiHandle,simInt buttonHandle,simInt position)
C parameters	uiHandle: custom user interface handle

	buttonHandle: button handle position: slider position. valid values are between 0 and 1000
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=simSetUISlider(number uiHandle,number buttonHandle,number position)
Lua parameters	Same as C-function
Lua return values	Same as C-function

simSetVisionSensorCharImageremote API equivalent: **simxSetVisionSensorImage**ROS API equivalent: **simRosSetVisionSensorImage**

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

simSetVisionSensorImageremote API equivalent: **simxSetVisionSensorImage**ROS API equivalent: **simRosSetVisionSensorImage**

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

simSimplifyMpPath

Description	Simplifies a path retrieved via the motion planning function simFindMpPath . The function uses V-REP's motion planning functionality .
C synopsis	simFloat* simSimplifyMpPath(simInt motionPlanningObjectHandle,const simFloat* pathBuffer,simInt configCnt,simInt options,simFloat stepSize,simInt increment,simInt* outputConfigsCnt,simInt maxTimeInMs,simFloat* reserved,const simInt* auxIntParams,const simFloat* auxFloatParams)
C parameters	motionPlanningObjectHandle: the handle of a motion planning object. Refer to simGetMotionPlanningHandle . pathBuffer: the buffer returned by the simFindMpPath function.. configCnt: the number of configuration nodes the pathBuffer contains. options: bit-coded: bit0: not used. Keep unset. bit1: not used. Keep unset. bit2: if set (4), then the found path will be visualized in yellow. bit3: if set (8), then some information will be output to the console. bit4: if set (16), then robot self-interferences will be ignored and calculations can drastically be sped-up. bit5: if set (32), then robot-environment interferences will be ignored and calculations can drastically be sped-up.

	<p>bit6: not used, keep unset. bit7: not used, keep unset. bit8: if set (256), then the returned Cartesian space distances will ignore the orientational distance component.</p> <p>stepSize: the maximum configuration space distance between individual collision-free phase2 nodes. the distance calculation will use the weight specified for each joint in the motion planning properties. increment: a value indicating how fine-grainer the simplification should be. 1 for best quality. outputConfigsCnt: a pointer to an integer receiving the number of returned configurations. maxTimeInMs: the maximum time in milliseconds after which the simplification operation is aborted. Specify 0 for an infinite time. reserved: reserved. Keep NULL. auxIntParams: reserved. Keep NULL. auxFloatParams: reserved. Keep NULL.</p>
C return value	<p>NULL in case of an error, or when the search failed. Otherwise a buffer of float values that the user is in charge of releasing with simReleaseBuffer. The returned buffer contains:</p> <p>the found path (x*n values): n configurations with each x values (x is the number of DoFs of the specified motion planning task). The configurations will include the start and the goal configuration, except when start and goal are coincident, in which case a single configuration is returned.</p> <p>the configuration space distances (n values): for each returned configuration, a distance to the start configuration (following the path). The last of the n values represents the length of the found path in the configuration space. The distance is calculated using the weight specified for each joint in the motion planning properties. the end-effector positions (3*n values): for each returned configuration, the position of the corresponding end-effector (x, y, z). the end-effector quaternions (4*n values): for each returned configuration, the quaternion of the corresponding end-effector (x, y, z, w). the Cartesian space distances (n values): for each returned configuration, a distance to the start pose (following the path). The last of the n values represents the length of the found path in the Cartesian space. The distance is calculated using the Cartesian space metric specified in the motion planning properties.</p>
Lua synopsis	<pre>table path,table confSpaceDist,table tipPositions,table tipQuaternions,table cartesianSpaceDist=simSimplifyMpPath(number motionPlanningObjectHandle,table path,number configCnt,number options,number stepSize,number increment,number maxTimeInMs=0,table auxIntParams=nil,table auxFloatParams=nil)</pre>
Lua parameters	Similar as C-function
Lua return values	Similar as C-function

simStartSimulationremote API equivalent: [simxStartSimulation](#)ROS API equivalent: [simRosStartSimulation](#)

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

simStopSimulationremote API equivalent: [simxStopSimulation](#)ROS API equivalent: [simRosStopSimulation](#)

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

simSwitchThread

Description	Allows specifying the exact moment at which the current thread should switch to another thread. If the current script doesn't run in a thread (i.e. if it runs in the application main thread), this function has no effect. By default, V-REP doesn't use "regular" threads, but something similar to hybrid threads (which behave like coroutines, but can also behave like regular threads). This allows much more flexibility and execution control of the threads: each thread (except for the main or application thread) has a switch timing associated, which specifies how long the thread will run before switching to other threads. By default this value is 2 millisecond, but can be modified with simSetThreadSwitchTiming . That timing can be shortened with simSwitchThread . Use with care when calling this function from a plugin. See also the
-------------	--

	simSetThreadAutomaticSwitch , simSetThreadResumeLocation and simSetThreadIsFree functions.
C synopsis	simInt simSwitchThread()
C parameters	None
C return value	1 if the thread was switched (the current thread gave control to other threads until the next calculation pass), 0 if it was not switched (e.g. because the function was called from the main or application thread, or from a thread started by the user), or -1 in case of an error.
Lua synopsis	number result=simSwitchThread()
Lua parameters	None
Lua return values	result: 1 if the thread was switched (the current thread gave control to other threads until the next calculation pass), 0 if it was not switched (e.g. because the function was called from the main or application thread), or -1 in case of an error.

simTransformVector

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

simTubeClose

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

simTubeOpen

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

simTubeRead

Description	Receives a data packet from a communication tube previously opened with simTubeOpen . The tube needs to be connected (see simTubeStatus).
C synopsis	simChar* simTubeRead(simInt tubeHandle,simInt* dataLength)
C parameters	tubeHandle: the handle of the tube that was returned by the simTubeOpen function.

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

simTubeStatus

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

simTubeWrite

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

simUngroupShape

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

simUnloadModule

Description	Unloads a V-REP plugin. This should be done just before ending the simulator. See also simSendMessage and simLoadModule .
C synopsis	simInt simUnloadModule(simInt pluginhandle)
C parameters	pluginhandle : handle of the plugin
C return value	0 if plugin was not unloaded (e.g. because it was not loaded, or because it was loaded more than once). Different from 0 if the plugin was successfully unloaded
Lua synopsis	-
Lua parameters	-
Lua return values	-

simUnlockResources

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

simUnpackBytes

Description	Unpacks a string (or part of it) into a table of byte numbers. See also simPackBytes , simUnpackInts , simUnpackWords and simUnpackFloats .
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	table byteNumbers=simUnpackBytes(string data,number startIndex=0,number count=0)
Lua parameters	data : a string (values between 0 and 255) that contains byte numbers startIndex : the zero-based index from which on data should be unpacked (from data[startIndex]). Can be omitted in which case 0 is used. count : the amount of bytes that should be unpacked. Can be omitted in which case 0 is used (which indicates that the maximum of bytes should be unpacked from the indicated startIndex).
Lua return values	byteNumbers : a table containing byte numbers if operation was successful, nil otherwise

simUnpackFloats

Description	Unpacks a string (or part of it) into a table of floating-point numbers. See also simPackFloats , simUnpackInts , simUnpackWords and simUnpackBytes .
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	table floatingNumbers=simUnpackFloats(string data,number startIndex=0,number count=0)
Lua parameters	data : a string (values between 0 and 255) that contains packed floating-point numbers startIndex : the zero-based index from which on data should be unpacked (from data[4*startIndex+1]). Can be omitted in which case 0 is used count : the amount of floats that should be unpacked. Can be omitted in which case 0 is used (which indicates that the maximum of floats should be unpacked from the indicated startIndex)
Lua return values	floatingNumbers : a table containing unpacked floating-point numbers if operation was successful, nil otherwise

simUnpackInts

Description	Unpacks a string (or part of it) into a table of integer numbers. See also simPackInts , simUnpackFloats , simUnpackWords and simUnpackBytes .
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	table integerNumbers=simUnpackInts(string data,number startIndex=0,number count=0)
Lua parameters	data : a string (values between 0 and 255) that contains packed integer numbers startIndex : the zero-based index from which on data should be unpacked (from data[4*startIndex+1]). Can be omitted in which case 0 is used count : the amount of ints that should be unpacked. Can be omitted in which case 0 is used (which indicates that the maximum of ints should be unpacked from the indicated startIndex)
Lua return values	integerNumbers : a table containing unpacked integer numbers if operation was successful, nil otherwise

simUnpackWords

Description	Unpacks a string (or part of it) into a table of word numbers. See also simPackWords , simUnpackFloats , simUnpackInts and simUnpackBytes .
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	table wordNumbers=simUnpackWords(string data,number startIndex=0,number count=0)
Lua parameters	data : a string (values between 0 and 255) that contains packed word numbers startIndex : the zero-based index from which on data should be unpacked (from data[2*startIndex+1]). Can be omitted in which case 0 is used.

	count: the amount of words that should be unpacked. Can be omitted in which case 0 is used (which indicates that the maximum of words should be unpacked from the indicated startIndex).
Lua return values	wordNumbers: a table containing unpacked word numbers if operation was successful, nil otherwise

simWait

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

simWaitForSignal

Description	Waits for a signal. Signals are cleared at simulation start. This function will first check whether an integer signal with that name is present, then if a float signal with that name is present and finally if a string signal with that name is present. The function only returns when the signal is present (defined). This function can only be called from child scripts running in a thread (since this is a blocking operation) and is not available from the C-API. See also simGetIntegerSignal , simGetFloatSignal , simGetStringSignal and simWait .
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	Number/string signalValue=simWaitForSignal(string signalName) The function is equivalent to:
	<pre> signalValue=nil while (signalValue==nil) do signalValue=simGetIntegerSignal(signalName) or simGetFloatSignal(signalName) or simGetStringSignal(signalName) if (signalValue==nil) then simSwitchThread() end end end </pre>
Lua parameters	signalName: name of the signal
Lua return values	signalValue: value of the signal, or nil when the command is cancelled by stopping the simulation

simWaitForWorkThreads

Description	Waits until work threads have finished their tasks. This function should only be called from the main script and is not available from the C-API. See also simEnableWorkThreads .
C synopsis	-
C parameters	-
C return value	-
Lua synopsis	simWaitForWorkThreads()
Lua parameters	none
Lua return values	none

simWriteCustomDataBlock

Description	Adds or removes custom data to be stored and saved together with an object, a script or a scene (i.e. the data will be part of the object, the script or the scene). If the data name starts with the string "@tmp", then the data will not be saved during a scene or model save operation. See also simReadCustomDataBlock .
C synopsis	simInt simWriteCustomDataBlock(simInt objectHandle,const simChar* dataName,const simChar* data,simInt dataSize)
C parameters	objectHandle: handle of the object or script where you want to store your data, or sim_handle_scene if you wish to store the data with the scene.

	<p>dataName: a string that identifies the data. An empty string will remove all custom data blocks. data: your custom data. If NULL, the current data under the specified dataName will be removed. dataSize: the size of your custom data</p> <p>The data will be copied to an internal buffer inside of the object or scene, and next time a scene or model is saved, will also be saved. The data buffer can be released after this call.</p>
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=simWriteCustomDataBlock(number objectHandle,string dataName,string data)
Lua parameters	Same as C-function. In addition, you can specify sim_handle_self for the objectHandle argument, if your target is the current script.
Lua return values	Same as C-function.

simWriteTexture

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