

# VirtualGrasp Unity API

[ENUMS] describe VirtualGrasp enums and are listed first only to allow other parts of this pdf to link back to them.

[EVENTS] describe VirtualGrasp events and are very useful to listen to for your own application.

Any of the (88) API functions from an \_API section can be accessed typing 'VG\_Controller.' from C# scripts. They are grouped in different sections for readability.

## [1. \[ENUMS\] \(16\)](#)

## [2. \[EVENTS\] \(13\)](#)

### [3.1 OBJECT\\_SELECTION\\_API \(20\)](#)

### [3.2 VIRTUALGRASP\\_CONTROLLER\\_FUNCTIONS \(14\)](#)

### [3.3 DATABASE\\_API \(3\)](#)

### [3.4 GRASP\\_EDITOR\\_API \(4\)](#)

### [3.5 GRASP\\_SELECTION\\_API \(22\)](#)

### [3.6 NETWORK\\_INTERFACE\\_API \(2\)](#)

A blue rounded rectangular badge with the text "pro\*" in white.

### [3.7 SENSOR\\_INTERFACE\\_API \(10\)](#)

### [3.8 RECORDING\\_INTERFACE\\_API \(11\)](#)

A blue rounded rectangular badge with the text "pro\*" in white.

### [3.9 ENABLE\\_NETWORK\\_API \(2\)](#)

## [ENUMS]

## (Unity API)

### VG\_AvatarType

An enum to describe an avatar type

**DEFAULT:** A default avatar

**REMOTE:** A remote avatar (multiplayer)

**REPLAY:** A replay avatar (AI)

### VG\_BoneType

An enum to describe a bone type, used for accessing of bones from outside the library.

**WRIST:** The wrist bone of a hand

**ELBOW:** The elbow bone of an arm

**SHOULDER:** The shoulder bone of an arm

**CLAVICLE:** The clavicle bone of an arm

**APPROACH:** The approach handle of a grasp

### VG\_EditorAction

Action towards the grasp editor, see EditGrasp()

**PRIMARY\_CURRENT:** Label the current grasp as primary, so it will be the only grasp for this object

**DISABLE\_CURRENT:** Label the current grasp as disabled, so it will not be accessible for static grasping.

**DELETE\_CURRENT:** Currently the same as DISABLE\_CURRENT, since we do not really want to remove grasps.

**DELETE\_ALL\_HAND\_GRASPS:** Delete the HandGrasp entry for a given object and hand hash

**ADD\_CURRENT:** Add the current grasp as a valid one, so it becomes accessible for static grasping.

**CLEAR\_PRIMARY:** Remove the label of the current object's primary grasp, so all grasps will be valid again.

**CLEAR\_DISABLED:** Remove the label of the current object's disabled grasps, so all grasps will be valid again.

### VG\_FingerControlType

An enum to describe how fingers are controlled.

**BY\_NOTHING:** When not grasping, fingers are not controlled at all.

**BY\_SENSOR\_FULL\_DOFS:** When not grasping, fingers are fully controlled by sensor

**BY\_SENSOR\_LOW\_DOFS:** When not grasping, fingers are controlled by sensor, but less DOF.

**BY\_ANIMATION:** When not grasping, fingers are controlled by animation.

**BY\_OSCILLATED\_ANIMATION:** When not grasping, fingers are controlled by oscillating between two state of animations

### VG\_GraspLabel

For labeling grasps (grasp editor functionality).

**DISABLED:** Labels a grasp as disabled

**PRIMARY:** Labels a grasp as primary

**SUCCEEDED:** Labels a grasp as succeeded

**FAILED:** Labels a grasp as failed

**RANK:** TBD

## VG\_GraspType

Animation grasp type enum.

**UNKNOWN\_GRASPTYPE:** Unknown grasp type

**POWER:** Power grasp

**PINCH:** Pinch grasp

**FLAT:** Flat grasp (like on a basketball)

**PUSH:** Push grasp

**GMANUS:** used for sensor animation

**OPENING:** this is for robotic opening grasp in order to grasp inside a hole of the object

**CLOSING:** this is for robotic closing grasp for parallel gripper

**SUCTION\_PIN:** this is for robotic suction pin gripper

## VG\_HandSide

We support two hands per avatar, left and right in this enum.

**LEFT:** Left hand

**UNKNOWN\_HANDSIDE:** Unknown hand side

**RIGHT:** Right hand

## VG\_InteractionType

An enum to describe a hand interaction type (i.e. a mode on grasp visualization).

**TRIGGER\_GRASP:** Original, hand goes to object at grasp position

**PREVIEW\_GRASP:** Grasp is always previewed once object is selected, trigger will allow pick up the object

**PREVIEW\_ONLY:** like PREVIEW\_GRASP, but trigger will not allow pick up the object

**JUMP\_GRASP:** Object jumps to hand when grasp is triggered

**STICKY\_HAND:** Object sticks to hand without grasp when grasp is triggered

**JUMP\_PRIMARY\_GRASP:** Using mechanism like JUMP\_GRASP, but use a grasp that is labeled as primary

## VG\_JointType

Different articulated joint types supported by VG.

**REVOLUTE:** revolute joint constrained around an axis, such as wheel

**PRISMATIC:** prismatic joint constrained along an axis, such as drawer

**FIXED:** fixed, not-moveable joint

**FLOATING:** floating, unconstrained joint

**PLANAR:** planar joint; up to here consistent with joint types in URDF, all 1-DOF joint

**CONE:** 3-DOF ball and socket joint modeled with cone joint limit

## VG\_NetworkSignal

Enum bitmask to compose parts of a NetworkSignal

**None:** Empty signal

**ControllerSignal:** Flag for the controller part of the network signal.

**SensorSignal:** Flag for the sensor part of the network signal.

**TriggerSignal:** Flag for the trigger part of the network signal.

**ObjectSignal:** Flag for the object part of the network signal.

## VG\_QueryGraspMethod

The query grasp method for GetGrasp() function

**BY\_INDEX:** get grasp by index

**BY\_ID:** get grasp by ID

**BY\_TCP:** get grasp by TCP

## VG\_QueryGraspMode

Decide when query grasp if hand moves and how to move hand.

**NO\_MOVE:** will not move internal object and hand

**MOVE\_HAND\_SMOOTHLY:** will move object and hand moves smoothly with GRASP transition

**MOVE\_HAND\_DIRECTLY:** will move object and hand move directly to target grasp pose

## VG\_ReturnCode

ReturnCode for various VirtualGrasp functions. Most functions in this API provide such a return code.

**SUCCESS:** Succeeded in processing function

**DLL\_NOT\_INITIALIZED:** Failed in processing function because library has not been initialized.

**DLL\_FUNCTION\_FAILED:** Failed in processing function because library has not been initialized.

**INVALID\_AVATAR:** Failed in processing function because the provided avatar is invalid.

**INVALID\_LIMB:** Failed in processing function because the provided limb or object is invalid.

**INVALID\_GRASP:** Failed in processing function because the provided grasp is invalid.

**INVALID\_TARGET:** Failed in processing function because the provided target is invalid.

**ARGUMENT\_ERROR:** Failed in processing function because a provided argument is invalid.

**UNSUPPORTED\_FUNCTION:** Failed in processing function because it is unsupported.

**OBJECT\_NO\_GRASPS:** Failed in processing function because there are no static grasps baked.

**OBJECT\_NO\_BAKE:** Failed in processing function because a baking process failed / there is no bake at all.

**LOAD\_GRASP\_DB\_FAILED:** Failed to pass a grasp db file into the library and process it.

**SAVE\_GRASP\_DB\_FAILED:** Failed to export the internal grasp db to a file.

**UNKNOWN\_AVATAR:**

**AVATAR\_BLOCKED:**

## VG\_SensorControlFlags

Enum flag to describe what controller signals a sensor should cover.

**POSITION:** Enable wrist position signal

**ROTATION:** Enable wrist rotation signal

**FINGERS:** Enable finger configuration signals

**GRASP:** Enable grasp trigger signal

**HAPTICS:** Enable haptics signals

## VG\_SensorType

Different sensor (or controller) types that can be used by VirtualGrasp. Note only External Controller is supported.

**NO\_CONTROLLER:** no controller

**LEAP:** Internal Controller (not supported), Leap motion 3D camera

**RAZER\_HYDRA:** Internal Controller (not supported), Razer Hydra controllers

**INTEL\_REALSENSE:** Internal Controller (not supported), Intel Realsense 3D camera

**MANUS:** Internal Controller (not supported), Manus VR gloves

**KNUCKLES:** Internal Controller (not supported), Valve Knuckles controller

**VIVE:** Internal Controller (not supported), HTC Vive controllers, supported through OpenVR

**OCULUS\_TOUCH\_OPENVR:** Internal Controller (not supported), Oculus Touch controllers, supported through OpenVR

**VIVE\_TRACKER:** Internal Controller (not supported), A ViveTracker

**OCULUS\_TOUCH\_OVR:** Internal Controller (not supported), Oculus Touch controllers, through OculusVR.

**EXTERNAL\_CONTROLLER:** External Controller, customized controller

**BEBOP:** Internal Controller (not supported), Bebop VR gloves

## VG\_VrButton

Enum for setting which (VR) controller buttons.

**TRIGGER:** Use the trigger button (usually index finger on the controller) to grasp.

**GRIP:** Use the grip button (usually middle finger on the controller) to grasp.

**GRIP\_OR\_TRIGGER:** Use both the trigger and the grip button (logical OR) to grasp.

## [EVENTS]

## (Unity API)

### **VG\_Controller.OnAfterReset**

The event to call when we have reset all objects in the library.

### **VG\_Controller.OnBeforeReset**

The event to call when we are going to reset all objects in the library.

### **VG\_Controller.OnGraspTriggered**

This event is invoked in the frame when a hand is starting to grasp an object. The VG\_HandStatus it carries includes more information about the interaction.

### **VG\_Controller.OnInitialize**

The event to call when we have successfully initialized the library.

### **VG\_Controller.OnObjectCollided**

This event is invoked when a grasped object is colliding with another object. The VG\_HandStatus it carries includes more information about the interaction.

Tutorials: [VG\\_ExternalControllerManager](#), [VG\\_HintVisualizer](#)

### **VG\_Controller.OnObjectDeselected**

This event is invoked in the frame when a hand is starting to deselect an object. The VG\_HandStatus it carries includes more information about the interaction.

Tutorial: [VG\\_Highlighter](#)

### **VG\_Controller.OnObjectFullyReleased**

This event is invoked in the frame when an object is fully release by all hands. The Transform it carries includes the object that has just been released.

### **VG\_Controller.OnObjectGrasped**

This event is invoked in the frame when a hand has fully grasped an object. The VG\_HandStatus it carries includes more information about the interaction.

### **VG\_Controller.OnObjectPushed**

This event is invoked in the frame when a hand pushing an object. The VG\_HandStatus it carries includes more information about the interaction.

### **VG\_Controller.OnObjectReleased**

This event is invoked in the frame when a hand is starting to release an object. The VG\_HandStatus it carries includes more information about the interaction.

### **VG\_Controller.OnObjectSelected**

This event is invoked in the frame when a hand is starting to select an object. The `VG_HandStatus` it carries includes more information about the interaction.

[Tutorial](#): `VG_Highlighter`

### **VG\_Controller.OnPostUpdate**

This event is invoked in the fixed update loop after VG runs its update. Thus, all other scripts that should update after the VG cycle should listen to this event.

### **VG\_Controller.OnPreUpdate**

This event is invoked in the fixed update loop before VG runs its update. Thus, all other scripts that should update before the VG cycle should listen to this event.

## OBJECT\_SELECTION\_API

(Unity API)

### VG\_Controller.ChangeObjectJoint

Change an object's joint in runtime.

**Transform selectedObject:** The object to change the joint type for.

**[VG\\_JointType](#) new\_jointType:** The joint type to switch to.

**Transform new\_anchor\_transform:** The anchor transform to switch to.

**Vector2 new\_limit:** The new limit of the new joint type.

**float new\_screwRate:** The new screw rate ( $\geq 0$ , in cm per degree) if new\_jointType is Revolute.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

**Remark:** Note that the former joint can be recovered (see RecoverObjectJoint).

**Remark:** If new\_screwRate is set to 0 then do not screw.

### VG\_Controller.ChangeObjectJoint

Change an object's joint and all other articulation parameters in runtime.

**Transform selectedObject:** The object to change the joint for.

**VG\_Articulation articulation:** An articulation describing the new articulation parameters.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

### VG\_Controller.ClearAvatarSpecificObjectSelectionWeights

Clear all avatar specific object selection weights.

**int avatarID:** The avatar id

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

### VG\_Controller.GetGraspButton

Return the currently selected GraspButton.

### VG\_Controller.GetGraspingAvatars

Return the avatar/hand pairs that are currently grasping a specified object.

**Transform objectToCheck:** The object to be checked if it is currently grasped.

**out List<KeyValuePair<int, VG\_HandSide>> hands:** An output list of avatar-handside-pairs describing which hands are currently grasping that object.

**returns int:** Number of hands grasping the object.

### VG\_Controller.GetObjectJointState

Get the current joint state of a single-dof articulated object. For planar joint, the joint state along xaxis of the joint anchor.

**Transform selectedObject:** The object to get the current joint state value for.

**out float jointState:** The returned joint state. Will be set to 0.0f upon error

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call. when selectedObject is null, or VG\_ReturnCode.DLL\_FUNCTION\_FAILED on an unexpected error.



### VG\_Controller.GetObjectJointType

Get object's original or current joint type.

**Transform selectedObject:** The object to get the current joint state value for.

**bool original:** If true, get the original joint type, otherwise the current type.

**out VG\_JointType jointType:** The returned joint type. Will be set to FLOATING upon error.

**returns VG\_ReturnCode:** VG\_ReturnCode describing the error state of the function call. when selectedObject is null, or VG\_ReturnCode.DLL\_FUNCTION\_FAILED on an unexpected error.

### VG\_Controller.GetObjectSecondaryJointState

Get the current secondary joint state along yaxis of joint anchor for planar articulated object.

**Transform selectedObject:** The object to get the current joint state value for.

**out float secondaryJointState:** The returned secondary joint state. Will be set to 0.0f upon error.

**returns VG\_ReturnCode:** VG\_ReturnCode describing the error state of the function call. when selectedObject is null, or VG\_ReturnCode.DLL\_FUNCTION\_FAILED on an unexpected error.

### VG\_Controller.GetSelectableObjects

Return all interactable objects.

**bool excludeHidden:** If to exclude objects that have been hidden in the scene.

**bool excludeUntagged:** If to exclude objects that have been untagged in the scene.

**returns IEnumerable<Transform>:** All interactable objects in the scene.

### VG\_Controller.GetSelectableObjectsFromScene

Return all interactable objects from the editor scene.

**bool excludeHidden:** If to exclude objects that have been hidden in the scene.

**bool excludeUntagged:** If to exclude objects that have been untagged in the scene.

**returns List<Transform>:** All interactable objects in the editor scene.

### VG\_Controller.GetSensorPose

Receive the sensor pose of a given avatar and hand.

**int avatarID:** The avatar to get the pose from.

**VG\_HandSide handSide:** The hand side to get the pose from.

**out Vector3 p:** The returned position.

**out Quaternion q:** The returned rotation.

**bool absolute:** Set True (default) to return the absolute pose, and False to return the relative pose.

**returns VG\_ReturnCode:** VG\_ReturnCode describing the error state of the function call.

### VG\_Controller.GetUnbakedObjects

Return all unbaked objects.

**returns List<Transform>:** A list of all unbaked objects in the scene as Unity Transforms.

### VG\_Controller.JumpGraspObject

Specify an object to be grasped by a hand no matter how far the object is, and object will jump to the hand.

**int avatarID:** The avatar id

**[VG\\_HandSide](#) handSide:** The side of the hand

**Transform obj:** The object that will be jump grasped by this hand

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

### VG\_Controller.RecoverObjectJoint

Recover an object's original joint, after it has been changed by ChangeObjectJoint().

**Transform selectedObject:** The object to recover the joint for.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

### VG\_Controller.SetAvatarSpecificObjectSelectionWeight

Specify the avatar specific object selection weight of an object for interaction.

**int avatarID:** The avatar id

**Transform obj:** Which object to specify weight

**float weight:** Should be  $\geq 0$  value to specify the preferences to select this object. If 0 exclude this object in selection process

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

Remark: Note by default this weight is 1 for all objects.

Remark: Use case is mainly to specify relative selection preferences for cluttered objects.

### VG\_Controller.SetDualHandsOnly

Set if an object can only be manipulated by dual hands from a same avatar.

**Transform selectedObject:** The object to change the dual hand type for.

**bool dualHandsOnly:** If dual hand only.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

### VG\_Controller.SetObjectJointState

Set the current joint to desired state for a single-dof articulated object.

**Transform selectedObject:** The object to set the joint state value for.

**float jointState:** The target joint state. If exceed joint limit will be constrained within limit.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

### VG\_Controller.SetObjectSelectionWeight

Specify the object selection weights for grasping interaction.

**Transform obj:** Which object to specify weight

**float weight:** Should be  $\geq 0$  value to specify the preferences to select this object. If 0 exclude this object in selection process

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

Remark: Note by default this weight is 1 for all objects.

Remark: Use case is mainly to specify relative selection preferences for cluttered objects.

### VG\_Controller.SwitchGraspObject

Instantaneously switch the grasped object to specified object in the function, the object will jump to hand.

**int avatarID:** The avatar id

**[VG\\_HandSide](#) handSide:** The side of the hand

**Transform obj:** The transform of the object to switch to grasp

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

## VG\_Controller.TogglePrimaryGraspOnObject

Instantaneously switch the grasped object, and continuously calling also toggle through primary grasps on this object.

**int avatarID:** The avatar id

**[VG\\_HandSide](#) handSide:** The side of the hand

**Transform obj:** The transform of the object to switch to grasp and toggle primary grasps.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

Remark: The specified object should have JUMP\_PRIMARY\_GRASP interaction type and has added primary grasps in the grasp db.

## VIRTUALGRASP\_CONTROLLER\_FUNCTIONS

(Unity API)

### VG\_Controller.Clear

Reset the plugin.

### VG\_Controller.GetAvatarID

Get the AvatarID of the given skinned mesh renderer

**out int avatarID:** The returned AvatarID.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode.SUCCESS on successfull avatar id fetch, or VG\_ReturnCode.INVALID\_AVATAR if avatar is null.

### VG\_Controller.GetDebugPath

Return the path where VG stores debug files.

**returns string:** The path (platform dependent).

### VG\_Controller.GetHand

Receive a specific hand and its status.

**int avatarID:** The avatar to get the hand status for.

[VG\\_HandSide](#) **side:** The hand side to get the avatar from.

**returns VG\_HandStatus:** A VG\_HandStatus.

### VG\_Controller.GetHands

Receive an enumerator of all registered hands and their status.

**returns List<VG\_HandStatus>:** Enumerator over VG\_HandStatus.

### VG\_Controller.GetSensorControlledAvatarID

Get the AvatarID of the first sensor controlled avatar.

**out int avatarID:** The returned AvatarID. Will be set to -1 upon error.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode.SUCCESS on successfull avatar id fetch, or VG\_ReturnCode.DLL\_FUNCTION\_FAILED on an unexpected error.

Remark: No guarantee on returning the one that was first sensor controlled avatar

### VG\_Controller.Initialize

Initialize the plugin.

### VG\_Controller.IsEnabled

Check if the plugin has been initialized and is ready to use.

### VG\_Controller.IsolatedUpdate

The Update() method has been divided into three parts: IsolatedUpdateDataIn(), IsolatedUpdate() and IsolatedUpdateDataOut() for application of the Burst compiler. IsolatedUpdate() runs the main update loop in VG.

### VG\_Controller.IsolatedUpdateDataIn

The FixedUpdate() method has been divided into three parts: IsolatedUpdateDataIn(), IsolatedUpdate() and IsolatedUpdateDataOut() for application of the Burst compiler. IsolatedUpdateDataIn() isolates data communication from Unity to VG.

### VG\_Controller.IsolatedUpdateDataOut

The Update() method has been divided into three parts: IsolatedUpdateDataIn(), IsolatedUpdate() and IsolatedUpdateDataOut() for application of the Burst compiler. IsolatedUpdateDataOut() isolates data communication from VG to Unity.

### VG\_Controller.Release

Release the plugin.

### VG\_Controller.SaveState

Save the object hierarchy debug state. This is done automatically when closing VirtualGrasp.

### VG\_Controller.UnRegisterAvatar

Unregister avatar during runtime

**int avatarID:** The id of the avatar to be unregistered.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

## DATABASE\_API

## (Unity API)

### VG\_Controller.DeleteGrasp

Deletes object-specific grasp db. Won't delete grasp if there still exists one or more registered objects with objectHash.

**uint objectHash:** Hash of the object to delete.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

**exception ArgumentException:** In case of unidentified objectHash.

### VG\_Controller.GetGrasp

Get grasp information in raw byte format by objectHash.

**uint objectHash:** Hash of the object for which to retrieve the grasp db.

**out VG\_RawDataHandle handle:** Handle with (encrypted) grasp information for object with hash objectHash.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

**exception ArgumentException:** In case of unidentified objectHash.

### VG\_Controller.LoadGrasp

Loads object-specific grasp db.

**byte[] grasp:** Byte stream of object-specific grasp db.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

**exception IOException:** In case of incorrect data format.

## GRASP\_EDITOR\_API

## (Unity API)

## VG\_Controller.EditGrasp

movie

Call grasp editor functionality on a currently selected object and grasp.

**int avatarID:** The avatar to call grasp editor functionality on.

**[VG\\_HandSide](#) handSide:** The hand side to call grasp editor functionality on.

**[VG\\_EditorAction](#) action:** The grasp editor function / action to call.

**Transform obj:** The object to call the action on (if not provided, the object in the hand).

**int grasp:** The grasp ID to call the action on (if not provided, the current grasp of the hand).

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

Tutorial: VG\_GraspStudio

## VG\_Controller.GetGrasp

Receive a grasp in the grasp DB by index.

**Transform selectedObject:** The object to receive a grasp for.

**int avatarID:** The avatar to receive a grasp for.

**[VG\\_HandSide](#) handSide:** The hand side to receive a grasp for.

**int graspIndex:** The index of grasp to receive.

**out Vector3 p:** The received wrist position of the grasp.

**out Quaternion q:** The received wrist orientation of the grasp.

**out VG\_GraspType type:** The received VG\_GraspType of the grasp.

**out VG\_GraspLabel label:** The received VG\_GraspLabel of the grasp.

**[VG\\_QueryGraspMode](#) queryGraspMode:** Can be used to define if and how the grasp should be applied also.

**[VG\\_QueryGraspMethod](#) queryGraspMethod:** Can be used to define how the graspIndex should be interpreted.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

Tutorial: VG\_GraspStudio

## VG\_Controller.GetInteractionTypeForObject

Get the current interaction type assigned to an object.

**Transform selectedObject:** The object to receive the interaction type for.

**returns [VG\\_InteractionType](#):** VG\_InteractionType describing the current interaction type of the object.

## VG\_Controller.GetNumGrasps

Receive the number of grasps for a specific object.

**Transform selectedObject:** The object to get the number of available grasps for.

**int avatarID:** If a valid avatarID together with handSide, receive only the available grasps for this hand (otherwise all available grasps).

**[VG\\_HandSide](#) handSide:** If a valid handSide together with avatarID, receive only the available grasps for this hand (otherwise all available grasps).

**returns int:** The number of grasps for the selected object (either all or for the specified hand).

Tutorial: VG\_HintVisualizer



## GRASP\_SELECTION\_API

(Unity API)

### VG\_Controller.ForceReleaseObject

untested

Force the release of a grasp.

**int avatarID:** The avatar to release grasps on all its hands.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

### VG\_Controller.ForceReleaseObject

untested

Force the release of a grasp.

**int avatarID:** The avatar to release a grasp for.

[VG\\_HandSide](#) **side:** The hand which to release the grasp for.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

### VG\_Controller.GetBone

Return the pose (i.e. position and orientation) of a specific bone.

**int avatarID:** The avatar to get the bone pose from.

[VG\\_HandSide](#) **handSide:** The hand side to get the bone pose from.

[VG\\_BoneType](#) **boneType:** The BoneType to get.

**out Transform t:** The returned pose of the bone.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

Tutorial: [VG\\_HandVisualizer](#)

### VG\_Controller.GetBone

Return the Transform that corresponds to a provided instance ID.

**int transformID:** The instance ID.

**returns Transform:** The Transform that corresponds to the transformID.

Tutorial: [VG\\_HandVisualizer](#)

### VG\_Controller.GetBone

Return the pose (i.e. position and orientation) of a specific bone.

**int avatarID:** The avatar to get the bone pose from.

[VG\\_HandSide](#) **handSide:** The hand side to get the bone pose from.

[VG\\_BoneType](#) **boneType:** The BoneType to get.

**out int instanceID:** The returned ID of the bone transform.

**out Vector3 p:** The returned position of the bone.

**out Quaternion q:** The returned rotation of the bone.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

Tutorial: [VG\\_HandVisualizer](#)

### VG\_Controller.GetBone

Return the pose matrix of a specific bone.

**int avatarID:** The avatar to get the bone pose from.

**[VG\\_HandSide](#) handSide:** The hand side to get the bone pose from.

**[VG\\_BoneType](#) boneType:** The BoneType to get.

**out int instanceID:** The returned ID of the bone transform.

**out Matrix4x4 m:** The returned pose matrix of the bone.

**returns Transform:** The Unity Transform that corresponds to the requested bone.

Tutorial: [VG\\_HandVisualizer](#)

## VG\_Controller.GetFingerBone

Return the pose of a specific finger bone as a matrix.

**int avatarID:** The avatar to get the bone pose from.

**[VG\\_HandSide](#) handSide:** The hand side to get the bone pose from.

**int fingerID:** The finger to get the bone pose from (from 0 as thumb to 4 as pinky).

**int boneID:** The bone index (from 0 as proximal to N as distal) to get the bone pose from. Use -1 for fingertip.

**out int instanceID:** The returned ID of the bone transform.

**out Matrix4x4 m:** The returned pose of the bone.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

Tutorial: [VG\\_HandVisualizer](#)

## VG\_Controller.GetFingerBone

Return the pose (i.e. position and orientation) of a specific finger bone.

**int avatarID:** The avatar to get the bone pose from.

**[VG\\_HandSide](#) handSide:** The hand side to get the bone pose from.

**int fingerID:** The finger to get the bone pose from (from 0 as thumb to 4 as pinky).

**int boneID:** The bone index (from 0 as proximal to N as distal) to get the bone pose from. Use -1 for fingertip.

**out int instanceID:** The returned ID of the bone transform.

**out Vector3 p:** The returned position of the bone.

**out Quaternion q:** The returned rotation of the bone.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

Tutorial: [VG\\_HandVisualizer](#)

## VG\_Controller.GetFingerBone

Reflect the pose of a specific bone on a Transform.

**int avatarID:** The avatar to get the bone pose from.

**[VG\\_HandSide](#) handSide:** The hand side to get the bone pose from.

**int fingerID:** The finger to get the bone pose from (from 0 as thumb to 4 as pinky).

**int boneID:** The bone index (from 0 as proximal to N as distal) to get the bone pose from. Use -1 for fingertip.

**out Transform t:** The returned pose of the bone.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

Tutorial: [VG\\_HandVisualizer](#)

## VG\_Controller.MakeGesture

Make a gesture with a hand.

**int avatarID:** The avatar to make gesture for.

**[VG\\_HandSide](#) side:** The hand which to make gesture for.

**[VG\\_GraspType](#) gesture:** The gesture to make with the [side] hand of avatar [avatarID].

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

## VG\_Controller.ReleaseGesture

Release a gesture on a hand

**int avatarID:** The avatar to release a grasp for.

**[VG\\_HandSide](#) side:** The hand which to release the grasp for.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

## VG\_Controller.SetBlockRelease

Specify if on this hand should block release or not in runtime.

**int avatarID:** The avatar to release a grasp for.

**bool block:** If block release signal or not on this avatar.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

## VG\_Controller.SetBlockRelease

Specify if on this hand should block release or not in runtime.

**int avatarID:** The avatar to release a grasp for.

**[VG\\_HandSide](#) side:** The hand which to release the grasp for.

**bool block:** If block release signal or not on this hand.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

## VG\_Controller.SetGlobalInteractionType

Set the global interaction type method. The interaction type defines how the hand and the object should get together during a grasp.

Remark: This will overwrite the specific grasp interaction type (see [SetInteractionTypeForObject](#)) for all objects.

**[VG\\_InteractionType](#) interactionType:** The method to switch to for all objects.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

## VG\_Controller.SetGlobalThrowAngularVelocityScale

Set the global throw angular velocity scale. The throw angular velocity scale defines how powerful the throw is in terms of rotation movement.

Remark: This will overwrite the specific throw angular velocity scale (see [SetThrowAngularVelocityScaleForObject](#)) for all objects.

**float throwAngularVelocityScale:** The throw angular velocity scale.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

## VG\_Controller.SetGlobalThrowVelocityScale

Set the global throw velocity scale. The throw velocity scale defines how powerful the throw is in terms of linear movement.

Remark: This will overwrite the specific throw velocity scale (see `SetThrowVelocityScaleForObject`) for all objects.

**float throwVelocityScale:** The throw translational velocity scale.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

## VG\_Controller.SetInteractionTypeForObject

Set the interaction type for a selected object. The interaction type defines how the hand and the object should get together during a grasp.

Remark: This will overwrite the global interaction type (see `SetGlobalInteractionType`) for that object.

**Transform selectedObject:** The object to modify the interaction type for.

[VG\\_InteractionType](#) **interactionType:** The interaction type to switch to for the object.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

## VG\_Controller.SetInteractionTypeForSelectedObject

Set the interaction type for a selected object. The interaction type defines how the hand and the object should get together during a grasp.

Remark: This will overwrite the global interaction type (see `SetGlobalInteractionType`) for that object.

**int avatarID:** The avatar which is selecting an object.

[VG\\_HandSide](#) **side:** The hand which is selecting an object.

[VG\\_InteractionType](#) **interactionType:** The interaction type to switch to for the object that is selected by the [side] hand of avatar [avatarID].

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

## VG\_Controller.SetThrowAngularVelocityScaleForObject

Set the throw angular velocity scale for a selected object. The throw angular velocity scale defines how powerful the throw is in terms of rotation movement.

Remark: This will overwrite the global throw angular velocity scale (see `SetGlobalThrowAngularVelocityScale`) for that object.

**Transform selectedObject:** The object to modify the throw velocity scale for.

**float throwAngularVelocityScale:** The throw angular velocity scale.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

## VG\_Controller.SetThrowAngularVelocityScaleForSelectedObject

Set the throw angular velocity scale for a selected object. The throw angular velocity scale defines how powerful the throw is in terms of rotation movement.

Remark: This will overwrite the global throw angular velocity scale (see `SetGlobalThrowAngularVelocityScale`) for that object.

**int avatarID:** The avatar which is selecting an object.

[VG\\_HandSide](#) **side:** The hand which is selecting an object.

**float throwAngularVelocityScale:** The throw angular velocity scale.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

## VG\_Controller.SetThrowVelocityScaleForObject

Set the throw velocity scale for a selected object. The throw velocity scale defines how powerful the throw is in terms of linear movement.

Remark: This will overwrite the global throw velocity scale (see SetGlobalThrowVelocityScale) for that object.

**Transform selectedObject:** The object to modify the throw velocity scale for.

**float throwVelocityScale:** The throw translational velocity scale.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

## VG\_Controller.SetThrowVelocityScaleForSelectedObject

Set the throw velocity scale for a selected object. The throw velocity scale defines how powerful the throw is in terms of linear movement.

Remark: This will overwrite the global throw velocity scale (see SetGlobalThrowVelocityScale) for that object.

**int avatarID:** The avatar which is selecting an object.

[VG\\_HandSide](#) **side:** The hand which is selecting an object.

**float throwVelocityScale:** The throw translational velocity scale.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

## NETWORK\_INTERFACE\_API

(Unity API)

### VG\_Controller.GetBroadcastSignal

pro\*

Receive (from VG) a multiplayer broadcast message as a binary byte array.

[VG\\_NetworkSignal](#) **signals:** A bitmask of network signals to request. Default is All.

**returns byte[]:** The message received by VG.

Tutorial: VG\_NetworkManager

### VG\_Controller.SetBroadcastSignal

pro\*

Set (to VG) a multiplayer broadcast message as a binary byte array.

**byte[] message:** The message (raw bytes) to be sent and processed by VG.

Tutorial: VG\_NetworkManager

## SENSOR\_INTERFACE\_API

(Unity API)

### VG\_Controller.GetGrabStrength

Returns the current grab strength of a hand. The grab strength is 0 for a fully open hand, 1 for a fully closed hand.

**int avatarID:** The avatar to receive the grab strength for.

[VG\\_HandSide](#) **handSide:** The hand side to receive the grab strength for.

**returns float:** The current grab strength of the [side] hand.

### VG\_Controller.GetGrabVelocity

Returns the current grab velocity of a hand. The current velocity of the grab strength (see GetGrabStrength), so negative when the hand is opening, and positive when the hand is closing.

**int avatarID:** The avatar to receive the grab velocity for.

[VG\\_HandSide](#) **handSide:** The hand side to receive the grab velocity for.

**returns float:** The current grab velocity of the [side] hand.

### VG\_Controller.GetPushCircle

Get the push circle for this hand side of an avatar as a visual hint for object selection for push without physics.

**int avatarID:** The avatar to get the push circle for.

[VG\\_HandSide](#) **handSide:** The hand to get the push circle for.

**out Vector3 p:** The push circle's position.

**out Quaternion r:** The push circle's rotation (zaxis is normal).

**out float radius:** Radius of the push circle,

**out bool inContact:** True if contact (i.e. pushing), False otherwise.

**returns Transform:** The selected object's Unity Transform, or null if none.

Tutorial: [VG\\_HintVisualizer](#)

### VG\_Controller.IsMissingSensorData

Check if a hand has invalid sensor data.

**int avatarID:** The avatar to check for.

[VG\\_HandSide](#) **handSide:** The hand side to check for.

**returns bool:** True if sensor data is invalid, False otherwise.

### VG\_Controller.SetAvatarActive

Set the active state of the avatar sensor(s) and mesh.

**int avatarID:** The avatar id.

**bool enableSensors:** If the sensor(s) that control this hand should be active or not.

**bool enableMesh:** If the mesh of this hand should be visible or not.

**Vector3 resetPos:** If an avatar is deactivated, hand positions will be reset to here (default (0,0,0)).

### VG\_Controller.SetCalibrationMode

Enable or disable wrist calibration mode (WCM). During enabled WCM, different ranges of motion of the wrist or grab strength will be calibrated.

Remark: untested

**int avatarID:** The avatar for which to enable/disable WCM.

**bool enabled:** True for enabling WCM, False for disabling it.

## VG\_Controller.SetExternalGrabStrength

Send an external controller grab signal to the plugin (for EXTERNAL\_CONTROLLER sensors).

**int avatarID:** The avatar to set external sensor pose for.

**VG\_HandSide handSide:** The hand side to set external sensor pose for.

**float strength:** The grab strength signal to set.

Tutorial: VG\_ExternalControllerManager

## VG\_Controller.SetFingerCalibrationMode

Enable or disable finger calibration mode (FCM). During enabled FCM, the hand opening range will be calibrated. After disabling it, grasp and release signals will work in this range.

**int avatarID:** The avatar for which to enable/disable FCM.

**bool enabled:** True for enabling FCM, False for disabling it.

## VG\_Controller.SetSensorActive

Set the active state of the sensor(s) that control the specified hand of an instance avatar.

**int avatarID:** The avatar id.

**VG\_HandSide handSide:** The side of the hand (remark: UNKNOWN will not have any effect).

**bool active:** If the sensor(s) that control this hand should be active or not.

**Vector3 resetPos:** If a hand is deactivated, its position will be reset to here (default (0,0,0)).

Remark: By default sensors are all active, and this function can be used in runtime to change this.

## VG\_Controller.SetSensorOffset

Change the sensor offset in runtime. The sensor offset is the offset between the pose that the current sensor is measuring and where the virtual hand is appearing in the scene.

Remark: Also treating left hand (LHS) and right hand (RHS) is considered, so the offset is applied symmetrically.

**int avatarID:** The avatar to set the offset for.

**VG\_SensorType sensor:** The sensor type to change the offset for.

**Vector3? position:** The offset position. Set to null if position should not be modified.

**Vector3? rotation:** The offset rotation. Set to null if rotation should not be modified.

**returns** **VG\_ReturnCode:** VG\_ReturnCode describing the error state of the function call.



## RECORDING\_INTERFACE\_API

(Unity API)

### VG\_Controller.GetReplayAvatarID

pro\*

Get the AvatarID of the first replay avatar.

**out int avatarID:** The returned AvatarID. Will be set to -1 upon error.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

Remark: No guarantee on returning the one that was first registered as replay avatar

### VG\_Controller.GetReplayStartWristPose

pro\*

Get the starting wrist poses for full replay of the whole interaction sequence.

**int avatarID:** The ID of the avatar to play the recording on (note: it has to be an avatar enabled for replay).

**Transform selectedObject:** If provided, the entire sensor recording will transformed in to object's frame. If not, in global frame.

**out Vector3 p\_left:** The position of the left wrist.

**out Quaternion q\_left:** The orientation of the left wrist.

**out Vector3 p\_right:** The position of the right wrist.

**out Quaternion q\_right:** The orientation of the right wrist.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

Remark: LoadRecording need to be called before this to load recorded sensor data.

Remark: SetProcessByRecordedFrame need to be called before this to set this avatar to be enabled for replay.

Tutorial: VG\_Recorder

### VG\_Controller.IsReplaySuccess

pro\*

Check if finished replay had identical response as recorded

**returns bool:** True if replay was identical, False otherwise.

Tutorial: VG\_Recorder

### VG\_Controller.IsReplaying

pro\*

Check if a hand is currently replaying a recorded sensor data.

**int avatarID:** The avatar to check.

[VG\\_HandSide](#) **handSide:** The hand to check.

**returns bool:** True if replaying, False otherwise.

Tutorial: VG\_Recorder

### VG\_Controller.LoadRecording

pro\*

movie

Load recorded sensor data from a file, but do not start replay

**string filename:** The filename to load the recording from.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

Tutorial: VG\_Recorder

## VG\_Controller.ResumeReplay

pro\*

movie

Resume replaying of an avatar.

**int avatarID:** The ID of the avatar to resume replaying the recording on (note: it has to be an avatar enabled for replay).

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

Tutorial: VG\_Recorder

## VG\_Controller.StartRecording

pro\*

movie

Start recording sensor data.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

Tutorial: VG\_Recorder

## VG\_Controller.StartReplay

pro\*

movie

Start full replay of the whole interaction sequence on an avatar.

**int avatarID:** The ID of the avatar to play the recording on (note: it has to be an avatar enabled for replay).

**Transform selectedObject:** If provided, the entire sensor recording will be replayed in this object's frame. If not, in global frame.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

Tutorial: VG\_Recorder

## VG\_Controller.StartReplayOnObject

pro\*

movie

Start replaying a specific interaction segment on one object.

**Transform obj:** The object to play the interaction on.

**int avatarID:** The avatar to play the interaction with.

[VG\\_HandSide](#) **handSide:** The hand to play the interaction with.

**int interactionId:** The ID of the interaction segment to be played on this object.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

Tutorial: VG\_Recorder

## VG\_Controller.StopRecording

pro\*

movie

Stop recording sensor data and store the whole sequence to a file

**string filename:** The filename to save the recording to.

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

Tutorial: VG\_Recorder

## VG\_Controller.StopReplay

pro\*

Stop replay of the recorded interaction sequence on an avatar.

**int avatarID:** The ID of the avatar to play the recording on (note: it has to be an avatar enabled for replay).

**returns** [VG\\_ReturnCode](#): VG\_ReturnCode describing the error state of the function call.

Tutorial: VG\_Recorder

## ENABLE\_NETWORK\_API

(Unity API)

### VG\_Controller.RegisterAvatar

Register a new avatar during runtime.

**SkinnedMeshRenderer avatar:** The skinned mesh renderer of the model that should be registered to VG.

**[VG\\_AvatarType](#) type:** The avatar type this avatar should be.

**out int id:** The new avatar ID will be assigned to this value after registration; -1 if it failed.

**int networkID1:** If networking is used, these will be the networkingIDs of the left hand of the new avatar (we assume max 2 hands per avatar).

**int networkID2:** If networking is used, these will be the networkingIDs of the left hand of the new avatar (we assume max 2 hands per avatar).

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.

### VG\_Controller.RegisterAvatar

Register a new avatar during runtime.

**SkinnedMeshRenderer avatar:** The skinned mesh renderer of the model that should be registered to VG.

**[VG\\_AvatarType](#) type:** The avatar type this avatar should be.

**out int id:** The new avatar ID will be assigned to this value after registration; -1 if it failed.

**returns [VG\\_ReturnCode](#):** VG\_ReturnCode describing the error state of the function call.