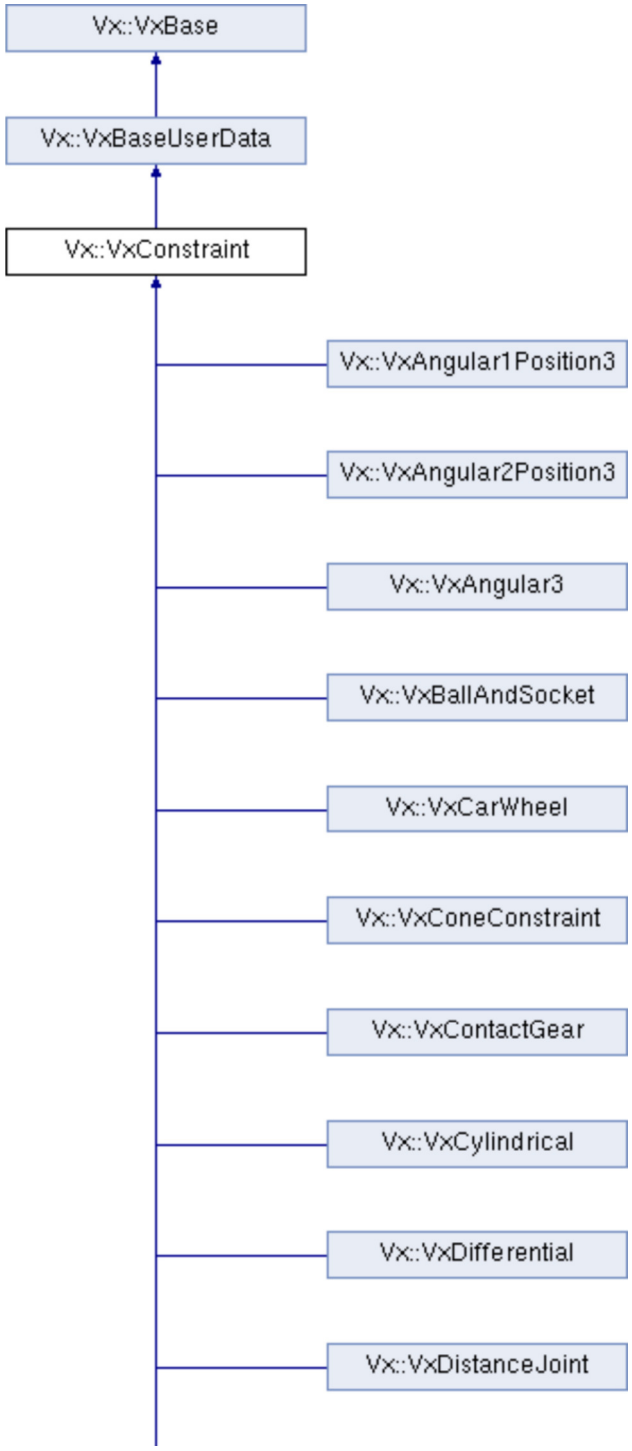
Vx::VxConstraint Class Reference

VxConstraint is the abstract base class for all constraints defined in Vortex. More...

`#include <VxConstraint.h>`

Inheritance diagram for Vx::VxConstraint:


Vx::VxBase
Vx::VxBaseUserData
Vx::VxConstraint
Vx::VxAngular1Position3
Vx::VxAngular2Position3
Vx::VxAngular3
Vx::VxBallAndSocket
Vx::VxCarWheel
Vx::VxConeConstraint
Vx::VxContactGear
Vx::VxCylindrical
Vx::VxDifferential
Vx::VxDistanceJoint

List of all members.

Public Types

enum	**Event** { **EVENT_ADD_PART** = 0, **EVENT_REMOVE_PART** }
enum	**CoordinateTypeEnum** { **kCoordinateLinear** = 0, **kCoordinateAngular**, **kCoordinateDistance** } *CoordinateTypeEnum describes the type of coordinated coordinate correspond to.* *More...*
enum	**CoordinateControlEnum** { **kControlFree**, **kControlMotorized**, **kControlLocked** } *A controllable coordinate can be controlled in different ways.* *More...*
enum	**LimitEnum** *Use LimitEnum enum with the limits interface.* *More...*
typedef int	**CoordinateID** *Controllable Coordinate Index.*
typedef int	**ConstraintEquationID** *Index of the constraint equation, the individual fundamental constraints forming the entire constraint.*

Public Member Functions

void	**setName** (const char *name) *Sets the constraint name.*
bool	**isOfType** (int classType) const *Returns true if the constraint type is of the specified class.*
void	**enable** (bool val) *Enables or disables the constraint.*
bool	**isEnabled** () const *Tests whether a constraint is enabled or disabled.*
bool	**isEnabledInternal** () const *Tests whether a constraint is internally enabled or disabled.*
void	**resetDynamics** () const *Resets all dynamical values associated with a constraint to their default states.*
int	**getCoordinateCount** () const *Returns the number of constraint coordinates (degrees of freedom) that are controllable (can have limits and motors).*

bool	**isCoordinateControllable** (**CoordinateID** coordinate) const *Returns true if this constraint's coordinate with index coordinate is controllable, i.e.*
CoordinateTypeEnum	**getCoordinateType** (**CoordinateID** coordinate) const *Returns an enumerated type describing whether the controllable coordinate with index coordinate is a position, a pure angle, or mixed type ((!) * kCoordinateDistance).*
bool	**isAngular** (**CoordinateID** coordinate) const *Returns true if the controllable coordinate with index coordinate is of angular type.*
bool	**setControl** (**CoordinateID** coordinate, **CoordinateControlEnum** control) *Specifies the type of controller applied to the given controllable coordinate.*
CoordinateControlEnum	**getControl** (**CoordinateID** coordinate) const *Returns the enumeration value of the type of controller currently applied to the given controllable coordinate.*
int	**getCoordinateReferencePartIndex** (**CoordinateID** coordinate) const *A coordinate refers to a specific axis to compute its position.*
virtual void	**setCoordinateReferencePartIndex** (**CoordinateID** coordinate, int partReferenceIndex) *Allows changing the reference part index.*
void	**getPartForce** (int partIndex, **VxReal3** v) const *Returns the net constraint force applied to the constrained **VxPart** with index partIndex by this constraint during the last time step.*
void	**getPartTorque** (int partIndex, **VxReal3** v) const *Returns the net constraint torque applied to the constrained **VxPart** with index partIndex by this constraint during the last time step.*
VxAssemblyBase *	**getAssembly** () const *Returns the assembly the constraint belong to.*
VxUniverse *	**getUniverse** () const *Returns the universe the constraint belong to.*
void	**setWeakOneFrame** () *Indicates to the constraint solver that the constraint is weak for the coming step.*
void	**setWeak** (bool b) *Indicates to the constraint solver that the constraint is permanently weak.*
bool	**getWeakOneFrame** () *Returns true if the constraint is weak for the coming step only.*
bool	**getWeak** () *returns true if the constraint is permanently weak.*
void	**setForcePartKinematicResponseIndex** (int partIndex) *Sets index of part for which the constraint will consider as being kinematic so that no force will be added to it.*

Attachment Setup Methods

Generic Methods that allow the user to specify constraint attachments.

int	**getMaxPartCount** () const

	*Returns the maximum number of **VxPart** objects that can be involved by this constraint.*
int	**getPartCount** () const *Returns the number of non-nil **VxPart** objects currently added to the constraint.*
virtual void	**setPart** (int partIndex, **VxPart** *part) *Sets the **VxPart** with index partIndex to the given part.*
void	**setParts** (**VxPart** *part1, **VxPart** *part2) *Sets the first and the second part (index 0 and 1) attached to this constraint.*
void	**setPartAndAttachmentRel** (int partIndex, **VxPart** *part, const **VxReal3** rpos, const **VxReal3** rpaxis, const **VxReal3** rsaxis) *Sets the **VxPart** with index partIndex for this constraint to that pointed to by * part, and sets the attachment frame position, primary axis, and secondary axis to the values rpos, rpaxis, and rsaxis, respectively, in the coordinate system of the given **VxPart**.*
void	**setPartAndAttachment** (int partIndex, **VxPart** *part, const **VxReal3** pos, const **VxReal3** paxis, const **VxReal3** saxis=0) *Sets the **VxPart** with index partIndex for this constraint to that pointed to by part, and sets the attachment frame position, primary axis, and secondary axis to the values rpos, rpaxis, and rsaxis, respectively, in the world coordinate system.*
VxPart *	**getPart** (const int partIndex) *Returns the part referenced by partIndex that is constrained by this constraint.*
const **VxPart** *	**getPart** (const int partIndex) const *Returns the part referenced by partIndex that is constrained by this constraint.*
virtual void	**setPartAttachmentPosition** (int partIndex, const **VxReal3** pos) *Sets the center of the attachment frame for the constrained **VxPart** with index partIndex in this constraint.*
virtual void	**setPartAttachmentPosition** (int partIndex, **VxReal** x, **VxReal** y, **VxReal** z) *Sets the attachment point of the **VxPart** with index partIndex in the constraint.*
virtual void	**getPartAttachmentPosition** (int partIndex, **VxReal3** pos) *Returns the attachment postion of **VxPart** with index partIndex in the world coordinate system.*
virtual void	**setPartAttachmentAxis** (int partIndex, const **VxReal3** primary) *Sets the primary attachment axes in the **VxPart** with index partIndex in this constraint, in the world reference frame.*
virtual void	**setPartAttachmentAxes** (int partIndex, const **VxReal3** primary, const **VxReal3** secondary) *Sets the primary, p, and secondary, s, constraint axes for **VxPart** with index partIndex in this constraint, in the world reference frame.*
virtual void	**getPartAttachmentAxes** (int partIndex, **VxReal3** primary, **VxReal3** secondary) *Returns the attachment axes in of the **VxPart** with index partIndex in this constraint, in the world reference frame.*
void	**updateAttachmentFromPart** (int partIndex) *Resets the attachment position and axes from given part.*
void	**setPartAttachmentPositionRel** (int partIndex, const **VxReal3** pos) *Sets the attachment postion of **VxPart** with index partIndex in this constraint, directly in the coordinate system of the **VxPart**.*
void	**setPartAttachmentPositionRel** (int partIndex, **VxReal** x, **VxReal** y, **VxReal** z) *Sets the attachment postion of **VxPart** with index partIndex in this constraint, directly in the coordinate system of the **VxPart**.*
virtual void	**setPartAttachmentAxesRel** (int partIndex, const **VxReal3** primary, const **VxReal3** secondary) *Sets the primary and the secondary attachment axis in the **VxPart** with index partIndex in this constraint, in the reference frame of the given **VxPart**.*
virtual void	**setPartAttachmentAxisRel** (int partIndex, const **VxReal3** primary)

	*Sets the primary attachment axes in the **VxPart** with index partIndex in this constraint, in the reference frame of the given **VxPart**.*
void	**getPartAttachmentPositionRel** (int partIndex, **VxReal3** pos) const *Returns the attachment postion of **VxPart** with index partIndex in the coordinate system of the **VxPart**.*
void	**getPartAttachmentAxesRel** (int partIndex, **VxReal3** primary, **VxReal3** secondary) const *Returns the attachment axes in of the **VxPart** with index partIndex in this constraint, in the reference frame of the given **VxPart**, overwriting the primary and vectors with.*
void	**setPartAttachmentPositionCOMRel** (int partIndex, const **VxReal3** pos) *Sets the attachment postion of **VxPart** with index partIndex in this constraint, directly in the coordinate system of the **VxPart** center of mass.*
void	**setPartAttachmentPositionCOMRel** (int partIndex, **VxReal** x, **VxReal** y, **VxReal** z) *Sets the attachment postion of **VxPart** with index partIndex in this constraint, directly in the coordinate system of the **VxPart** center of mass.*
void	**getPartAttachmentPositionCOMRel** (int partIndex, **VxReal3** pos) const *Returns the attachment postion of **VxPart** with index partIndex in the coordinate system of the **VxPart** center of mass.*

Motor Coordinate Control Methods

Methods which act on a controllable coordinate in order to set its motor parameters.

void	**setMotorParameters** (**CoordinateID** coordinate, **VxReal** desiredVelocity, **VxReal** maxForce, **VxReal** loss=0) *Sets the parameters for a limited-force motor on the controllable coordinate with index coordinate.*
void	**setMotorDesiredVelocity** (**CoordinateID** coordinate, **VxReal** desiredVelocity) *Sets the desired velocity of the limited-force motor for the controllable coordinate with index coordinate.*
void	**setMotorMaximumForce** (**CoordinateID** coordinate, **VxReal** maxForce) *Sets the maximum allowed force of the limited-force motor for the controllable coordinate with index coordinate.*
void	**setMotorMinimumAndMaximumForce** (**CoordinateID** coordinate, **VxReal** minForce, **VxReal** maxForce) *Sets the maximum allowed force of the limited-force motor for the controllable coordinate with index coordinate.*
void	**setMotorLoss** (**CoordinateID** coordinate, **VxReal** loss) *Sets the kinetic loss coefficient of the limited-force motor for the controllable coordinate with index coordinate.*
VxReal	**getMotorDesiredVelocity** (**CoordinateID** coordinate) const *Returns the desired velocity of the limited-force motor for the controllable coordinate with index coordinate.*
VxReal	**getMotorMaximumForce** (**CoordinateID** coordinate) const *Returns the value of the maximum force (or torque) of the limited-force motor for the controllable coordinate with index coordinate.*
VxReal	**getMotorMinimumForce** (**CoordinateID** coordinate) const *Returns the value of the minimum force (or torque) of the limited-force motor for the controllable coordinate with index coordinate.*
VxReal	**getMotorLoss** (**CoordinateID** coordinate) const *Returns the value of kinetic loss of the limited-force motor for the controllable coordinate with index coordinate.*
bool	**isMotorized** (**CoordinateID** coordinate) const *Returns true if a limited-force motor constraint for controllable coordinate with index coordinate has been set and is active.*
void	**setMotorStopAtLock** (**CoordinateID** coordinate, bool b) *Force the coordinate control to switch from kControlMotorized to kControlLocked when constraint position meets lock position.*

bool	**getMotorStopAtLock** (**CoordinateID** coordinate) const *Returns true if the coordinate control switches to lock if constraint position meets lock position.*

Lock Coordinate Control Methods

Methods which act on a controllable coordinate in order to lock the coordinate to a given value and set other lock parameters such as stiffness.

void	**setLockParameters** (**CoordinateID** coordinate, **VxReal** lockPosition, **VxReal** maxForce, **VxReal** stiffness=VX_INFINITY, **VxReal** damping=0, **VxReal** velocity=0) *Configures a lock controller for the controllable coordinate with index coordinate.*
void	**setLockPosition** (**CoordinateID** coordinate, **VxReal** lockValue) *Locks the controllable degree of freedom with index coordinate at the position lockValue.*
void	**setLockVelocity** (**CoordinateID** coordinate, **VxReal** velValue) *Constrains the controllable degree of freedom with index coordinate to change with velocity velValue.*
void	**setLockMaximumForce** (**CoordinateID** coordinate, **VxReal** value) *Sets the maximum force that vortex will apply to enforce the position lock constraint on the controllable coordinate with index coordinate.*
void	**setLockMinimumAndMaximumForce** (**CoordinateID** coordinate, **VxReal** minValue, **VxReal** maxValue) *Sets the minimum and maximum force that vortex will apply to enforce the position lock constraint on the controllable coordinate with index coordinate.*
void	**getLockMinimumAndMaximumForce** (**CoordinateID** coordinate, **VxReal** *minValue, **VxReal** *maxValue) const
void	**setLockStiffness** (**CoordinateID** coordinate, **VxReal** stiffness) *Sets the constraint stiffness on the lock constraint of controllable coordinate with index coordinate.*
void	**setLockDamping** (**CoordinateID** coordinate, **VxReal** damping) *Sets the constraint damping on the lock constraint of controllable coordinate with index coordinate.*
void	**setLockStiffnessAndDamping** (**CoordinateID** coordinate, **VxReal** stiffness, **VxReal** damping) *Sets the constraint relaxation paramters of the lock constraint for the controllable coordinate with index coordinate.*
VxReal	**getLockPosition** (**CoordinateID** coordinate) const *Returns target coordinate for the lock constraint on the controllable coordinate with index coordinate.*
VxReal	**getLockVelocity** (**CoordinateID** coordinate) const *Returns the current value of the moving lock constraint on the controllable coordinate with index coordinate.*
VxReal	**getLockMaximumForce** (**CoordinateID** coordinate) const *Returns the value of the maximum allowed force for the lock constraint on the controllable coordinate with index coordinate.*
VxReal	**getLockStiffness** (**CoordinateID** coordinate) const *Returns the constraint stiffness on the lock constraint of controllable coordinate with index coordinate.*
VxReal	**getLockDamping** (**CoordinateID** coordinate) const *Returns the constraint damping on the lock constraint of controllable coordinate with index coordinate.*
bool	**isLocked** (**CoordinateID** coordinate) const *Returns true if a lock constraint for controllable coordinate with index coordinate has been set and is active.*

Coordinate Position, Velocity and Force Methods

Methods to read coordinate positions and other physical data.

void	**setCoordinateCurrentPosition** (**CoordinateID** coordinate, **VxReal** newPos) *Sets the the zero reference for the controllable degree of freedom with index coordinateID, so that the current coordinate value is newPosOffset after this method is invoked.*
VxReal	**getCoordinateCurrentPosition** (**CoordinateID** coordinate) const *Returns the current coordinate value for the degree of freedom with index coordinateID.*
VxReal	**recalculateCoordinateCurrentPosition** (**CoordinateID** coordinateID) *Force coordinate position to be recalculated.*
VxReal	**getCoordinateVelocity** (**CoordinateID** coordinate) const *Returns the velocity of controllable coordinate with index coordinate.*
VxReal	**getCoordinateForce** (**CoordinateID** coordinate) const *Returns the force applied by the motor, lock and/or range limit constraint for the controllable coordinate with index coordinate, during last step.*
void	**setForceCalculateCoordinatePosition** (**CoordinateID** coordinate, bool b) *Forces the computation of the coordinate of the controllable degree of freedom with index coordinate, even if this is not strictly required because range limits, limited-force, or lock controllers have been configured and activated.*
VxReal	**getCoordinateOffset** (**CoordinateID** coordinate) const *Gives direct access to the coordinate offset between desired geometrical value of the position and the desired position.*
void	**setCoordinateOffset** (**CoordinateID** coordinate, **VxReal** offset) *Gives direct access to the coordinate offset between desired geometrical value of the position and the desired position.*
void	**setCoordinateInternalPosition** (**CoordinateID** coordinate, **VxReal** pos) *This will set the internal position of the coordinate.*
VxReal	**getCoordinateInternalPosition** (**CoordinateID** coordinate) const *Returns the coordinate internal position.*

Limit Coordinate Control Methods

Methods which act on a controllable coordinate in order to set bounds on the coordinate position and related parameters.

void	**setLowerLimit** (**CoordinateID** coordinate, **VxReal** limitPos, **VxReal** limitVel=0, **VxReal** restitution=0, **VxReal** stiffness=sDefaultStiffness, **VxReal** damping=sDefaultDamping) *Sets the configuration of the lower range limit on the controllable constraint coordinate with index coordinateID.*
void	**setUpperLimit** (**CoordinateID** coordinate, **VxReal** limitPos, **VxReal** limitVel=0, **VxReal** restitution=0, **VxReal** stiffness=sDefaultStiffness, **VxReal** damping=sDefaultDamping) *Sets the configuration of the upper range limit on the controllable constraint coordinate with index coordinateID.*
void	**setLimitPositions** (**CoordinateID** coordinate, **VxReal** lower, **VxReal** upper) *Sets the lower and upper values of the range of a controllable coordinate with index coordinate.*
void	**setLimitPosition** (**CoordinateID** coordinate, **LimitEnum** index, **VxReal** limit) *Sets the lower or upper value of the range of a controllable coordinate with index coordinate to the value limit.*

VxReal	**getLimitPosition** (**CoordinateID** coordinate, **LimitEnum** index) const *Returns the value of the lower or upper range for a controllable coordinate with index coordinate.*
void	**setLimitVelocity** (**CoordinateID** coordinate, **LimitEnum** index, **VxReal** limitVelocity) *Sets the velocity of the lower or upper range limits, for the controllable coordinate with index coordinate.*
VxReal	**getLimitVelocity** (**CoordinateID** coordinate, **LimitEnum** index) const *Returns the velocity of the lower or upper range limit constraint of the controllable coordinate with index coordinate.*
void	**setLimitRestitution** (**CoordinateID** coordinate, **LimitEnum** index, **VxReal** restitution) *Sets the restitution property of the limit.*
VxReal	**getLimitRestitution** (**CoordinateID** coordinate, **LimitEnum** index) const *Returns limit restitution parameters.*
void	**setLimitMaximumForce** (**CoordinateID** coordinate, **LimitEnum** index, **VxReal** MaxForce) *Sets limit's spring max force.*
VxReal	**getLimitMaximumForce** (**CoordinateID** coordinate, **LimitEnum** index) const *Returns limit's spring max force.*
void	**setLimitStiffness** (**CoordinateID** coordinate, **LimitEnum** index, **VxReal** stiffness) *Sets the constraint stiffness on the range limit constraint.*
VxReal	**getLimitStiffness** (**CoordinateID** coordinate, **LimitEnum** index) const *Returns the constraint stiffness of the given range limit constraint for controllable coordinate with index coordinate.*
void	**setLimitDamping** (**CoordinateID** coordinate, **LimitEnum** index, **VxReal** damping) *Sets the constraint damping for the lower or upper range limit constraint on the controllable coordinate with index coordinate.*
VxReal	**getLimitDamping** (**CoordinateID** coordinate, **LimitEnum** index) const *Returns the constraint damping for the lower or upper range limit constraint on the controllable coordinate with index coordinate.*
void	**setLimitsActive** (**CoordinateID** coordinate, bool activate) *Call this method to enable or not the limit.*
bool	**getLimitsActive** (**CoordinateID** coordinate) const *Returns true if range limit constraints for controllable coordinate with index coordinate has been set and are active.*
bool	**isLimitExceeded** (**CoordinateID** coordinate) const *Returns true if the range limit for the controllable coordinate with index coordinate is activated, and the current coordinate value is outside the allowed bounds.*
bool	**isLimitReached** (**CoordinateID** coordinate, **LimitEnum** index, **VxReal** threshold=1.0e-10) const *Returns true if the current position for the controllable coordinate is close enough to the limit with index coordinate (and if the limit is activated)*

Contraint Equation Relaxation Methods

*Methods which act on individual fundamental constraint equations (specified by their **VxConstraint::ConstraintEquationID** index) in order to add compliance to a constraint.*

int	**getConstraintEquationCount** () const *Returns the number of basic constraint equations that are natively maintained by the constraint.*

void	**setRelaxationParameters** (**ConstraintEquationID** c, **VxReal** stiffness, **VxReal** damping, **VxReal** loss, bool enabled) *Sets the relaxation parameters of the basic constraint equation with index c.*
void	**getRelaxationParameters** (**ConstraintEquationID** c, **VxReal** *stiffness, **VxReal** *damping, **VxReal** *loss, bool *enabled) *Returns the relaxation paramters used for the basic constraint equation with index c.*
void	**setRelaxationStiffnessAndDamping** (**ConstraintEquationID** c, **VxReal** stiffness, **VxReal** damping) *Sets the value of stiffness and damping of basic constraint equations with index c.*
void	**setRelaxationStiffness** (**ConstraintEquationID** c, **VxReal** stiff) *Sets the value of stiffness for the basic constraint equation with index c.*
VxReal	**getRelaxationStiffness** (**ConstraintEquationID** c) const *Returns the value of constraint stiffness for the basic constraint equation with index c.*
void	**setRelaxationDamping** (**ConstraintEquationID** c, **VxReal** damp) *Sets the value of damping for the basic constraint equation with index c.*
VxReal	**getRelaxationDamping** (**ConstraintEquationID** c) const *Returns the value of constraint damping for the basic constraint equation with index c.*
void	**setRelaxationLoss** (**ConstraintEquationID** c, **VxReal** loss) *Sets the value of kinetic loss to loss for the basic constraint equation with index c.*
VxReal	**getRelaxationLoss** (**ConstraintEquationID** c) const *Returns the value of kinetic loss for the basic constraint equation with index c.*
void	**enableRelaxation** (**ConstraintEquationID** c, bool enable) *Enable/Disable relaxation of the basic constraint equation with index c.*
bool	**getRelaxationEnabled** (**ConstraintEquationID** c) const *Returns true if the basic constraint equation with index c has been relaxed.*
VxReal	**getConstraintEquationForce** (**ConstraintEquationID** c) const *Returns the force applied by the basic constraint with index c during last step.*
void	**setRelaxationMinMaxForce** (**ConstraintEquationID** c, **VxReal** minForce, **VxReal** maxForce) *Sets the minimum and maximum force the constraint equation may apply on the part, this applies only if the equation is relaxed.*
void	**getRelaxationMinMaxForce** (**ConstraintEquationID** c, **VxReal** *minForce, **VxReal** *maxForce) const *Gets the minimum and maximum force the constraint equation may apply on the part, this applies only if the equation is relaxed.*

Contraint friction Methods

Methods to control friction in the constraint.

The internal friction is only available for two parts constraints.

VxConstraintFriction *	**getCoordinateFriction** (**CoordinateID** coordinate) const *Returns friction relative to coordinate coordinate.*
VxReal	**getCoordinateFrictionForceLastFrame** (**CoordinateID** coordinate) const *Returns the force or torque added by the friction for the coordinate coordinate during last simulation step.*

bool	**getCoordinateFrictionAvailable** (**CoordinateID** coordinate) const
VxConstraintFriction *	**getConstraintEquationFriction** (**ConstraintEquationID** c) const *Returns friction relative to constraint equation c.*
VxReal	**getConstraintEquationFrictionForceLastFrame** (**ConstraintEquationID** c) const *Returns the amount of force or torque applied by the friction on the constraint equation during last simulation step.*
bool	**getConstraintEquationFrictionAvailable** (**ConstraintEquationID** c) const
VxConstraintFriction *	**getFriction** (int axis, bool linear) const
VxReal	**getFrictionForceLastFrame** (int axis, bool linear) const

Protected Member Functions

void	**setForcePartKinematicResponse** (**VxPart** *p) *Forcing one of the part in constraint to be considered as kinematic so that the other constraint parts will see it as if it had infinite mass.*
VxPart *	**getForcePartKinematicResponse** () *Return the part being forced kinematic,.*
void	**enableBodies** () *Forces the **VxPart** objects of a constraint to be in the enabled state, unless they are in the frozen state.*

Detailed Description

VxConstraint is the abstract base class for all constraints defined in Vortex.

Constraints impose restrictions on the motion of one or several parts in the case of kinematic constraints such as hinge or prismatic joints, or on the constraint forces themselves as in the case of the Coulomb friction model in the contact constraint, or drivers.

The constraint library of Vortex includes ideal (workless) positional kinematic constraints which restrict relative positions and orientations between a given list of **VxPart** objects such as hinge, prismatic, or ball joints for instance. Vortex also includes ideal velocity constraints restricting relative linear and angular velocity between a given list of parts, as is the case for the gear constraint for instance. Finally, Vortex provides non-ideal constraints which can do work on the system. By introducing relaxation for instance, any given constraint can be made non-ideal by becoming compliant. This applies to both positional and velocity constraints. Also, for some constraint types, the user can also set the maximum allowed constraint force which introduces dissipation and makes the given constraint non-ideal. All these features are useful in modelling real-life physical systems.

Each type of constraint provides one constructor which completely configures the constraint at creation time, and another one which creates the given constraint but without any configuration. The interface defined in the abstract base class allows specification of the constraint configuration including the specification of the **VxPart** objects involved, the geometric configuration, and the type of control applied on the constraint coordinates (specified by their **VxConstraint::CoordinateID** index). This is set using the **VxConstraint::setControl()** method. It is also possible to set the relaxation parameters for all individual fundamental constraint equations (specified by their **VxConstraint::CoordinateControlEnum** index).

Once a specific constraint object is created and configured, it must be added to the **VxUniverse** object and enabled to become active.

Constraint geometric configuration is specific to each constraint type but in general, the user must specify attachment coordinate systems by entering the location of the geometric center of a constraint as well as one or two axes as the case applies.

Each constraint type applies a number of individual fundamental constraint equations, and leaves a number of degrees of freedom. Vortex can then compute coordinates for the degrees of freedom which can be further controlled by the user by imposing range limits, locks, or drivers which simulate motors on them.

Constraints which offer a control interface for their degrees of freedom maintain dynamic state information and must be reset using the **VxConstraint::resetDynamics()** method if they are reconfigured or re-initialized.

The dynamical behavior of the individual fundamental constraints themselves can also be adjusted by the user as they can be relaxed individually with compliance (inverse stiffness) and damping parameters.

VxPart objects of a given constraint can be set to the nil pointer. In this case the corresponding attachment refers to the inertial coordinate system. The other non-nil constrained **VxPart** objects are then attached to the inertial frame by the given constraint.

Member Typedef Documentation

typedef int Vx::VxConstraint::ConstraintEquationID

Index of the constraint equation, the individual fundamental constraints forming the entire constraint.

They complement the controllable coordinates in each constraint, and correspond to the degrees of freedom which are always removed by the constraint. These equations can be relaxed, thus introducing compliance to the relevant "degree of freedom".

typedef int Vx::VxConstraint::CoordinateID

Controllable Coordinate Index.

Each subclassed contraint will have a set of CoordinateIDs defined as static variables. These are to be passed to functions such as **VxConstraint::setControl()**, **VxConstraint::setMotorParameters()**, **VxConstraint::setLockParameters()**, **VxConstraint::setLowerLimit()**, **VxConstraint::setUpperLimit()**, etc...

Member Enumeration Documentation

enum Vx::VxConstraint::CoordinateControlEnum

A controllable coordinate can be controlled in different ways.

See also:
setControl and **getControl**

Enumerator:

kControlFree
Coordinate is free.

kControlMotorized
Coordinate is motorized.

kControlLocked
Coordinate is locked.

enum Vx::VxConstraint::CoordinateTypeEnum

CoordinateTypeEnum describes the type of coordinated coordinate correspond to.

Enumerator:

kCoordinateLinear
Linear coordinate.

kCoordinateAngular
Angular coordinate.

kCoordinateDistance
Distance coordinate.

enum Vx::VxConstraint::Event

Enumerator:

EVENT_ADD_PART
A part has been added to the constraint.

EVENT_REMOVE_PART
A part has been removed from the constraint.

Reimplemented from **Vx::VxBase**.

enum Vx::VxConstraint::LimitEnum

Use *LimitEnum* enum with the limits interface.

Member Function Documentation

void VxConstraint::enable (bool *val*)

Enables or disables the constraint.

An argument value of "true" will enable the joint if possible, "false" will disable it and allow it's attached bodies to move freely. A constraint will be internally enabled if and only if there is at least one dynamic part added to it and all the added parts as well as the constraint itself as been added to the universe.

Constraints are enabled by default upon creation.

References **getMaxPartCount()**, **getPartCount()**, and **getUniverse()**.

Referenced by **Vx::VxPart::_applyCOMRelativeTM()**, **Vx::VxWheelTireModelSubscriber::handleWheelContacts()**, **Vx::VxWheelTireModelSubscriber::postStep()**, **Vx::VxStateConstraint::restoreState()**, **setPart()**, **Vx::VxAngular1Position3::~VxAngular1Position3()**, **Vx::VxAngular2Position3::~VxAngular2Position3()**, **Vx::VxAngular3::~VxAngular3()**, **Vx::VxBallAndSocket::~VxBallAndSocket()**, **Vx::VxCarWheel::~VxCarWheel()**, **Vx::VxConeConstraint::~VxConeConstraint()**, **Vx::VxContactGear::~VxContactGear()**, **Vx::VxCylindrical::~VxCylindrical()**, **Vx::VxDifferential::~VxDifferential()**, **Vx::VxDistanceJoint::~VxDistanceJoint()**, **Vx::VxDoubleWinch::~VxDoubleWinch()**, **Vx::VxGearRatio::~VxGearRatio()**, **Vx::VxLinear2::~VxLinear2()**, **Vx::VxLinear3::~VxLinear3()**, **Vx::VxMotorConstraint::~VxMotorConstraint()**, **Vx::VxRPRO::~VxRPRO()**, **Vx::VxScrewJoint::~VxScrewJoint()**, **Vx::VxSumDistance::~VxSumDistance()**, and **Vx::VxWinch::~VxWinch()**.

void VxConstraint::enableRelaxation (ConstraintEquationID *c*,
bool *enable*
)

Enable/Disable relaxation of the basic constraint equation with index *c*.

See also:
setRelaxationStiffnessAndDamping.

Referenced by **setRelaxationParameters()**.

VxAssemblyBase* Vx::VxConstraint::getAssembly ( ) const `[inline]`

Returns the assembly the constraint belong to.

Referenced by **Vx::VxAssemblyBase::_addConstraint()**, **Vx::VxAssemblyBase::_removeConstraint()**, **Vx::VxAssemblyBase::contains()**, and **Vx::VxStateConstraint::setBaseObject()**.

int VxConstraint::getConstraintEquationCount ( ) const

Returns the number of basic constraint equations that are natively maintained by the constraint.

Those equations can be relaxed using VxConstraint::setRelaxationXXX methods. Not to be confused with controllabled coordinates which can be limited, locked or motorized.

Referenced by **Vx::VxUniverse::verifyDynamics()**.

VxReal VxConstraint::getConstraintEquationForce (ConstraintEquationID *c*) const

Returns the force applied by the basic constraint with index *c* during last step.

Use isAngular(coordinate) to know if the returned value is a force or a torque.

See also:
VxConstraint::isAngular(CoordinateID coordinate).

VxConstraintFriction * VxConstraint::getConstraintEquationFriction (ConstraintEquationID *c*) const

Returns friction relative to constraint equation *c*.

Use the returned object to set the friction parameters. Null is return if friction is not available for this equation.

VxReal VxConstraint::getConstraintEquationFrictionForceLastFrame (ConstraintEquationID *c*) const

Returns the amount of force or torque applied by the friction on the constraint equation during last simulation step.

VxReal VxConstraint::getCoordinateForce (CoordinateID *coordinate*) const

Returns the force applied by the motor, lock and/or range limit constraint for the controllable coordinate with index *coordinate*, during last step.

Use isAngular(coordinate) to know if the returned value is a force or a torque.

See also:
VxConstraint::isAngular(CoordinateID coordinate).

VxConstraintFriction * VxConstraint::getCoordinateFriction (CoordinateID *coordinate*) const

Returns friction relative to coordinate *coordinate*.

Use the returned object to set the friction parameters. Null is return if friction is not available for this coordinate.

Referenced by setCoordinateReferencePartIndex().

VxReal VxConstraint::getCoordinateFrictionForceLastFrame (CoordinateID *coordinate*) const

Returns the force or torque added by the friction for the coordinate *coordinate* during last simulation step.

VxReal VxConstraint::getCoordinateInternalPosition (CoordinateID *coordinateID*) const

Returns the coordinate internal position.

See also:
setCoordinateInternalPosition.

References isCoordinateControllable().

Referenced by Vx::VxStateConstraint::loadState().

VxReal VxConstraint::getCoordinateOffset (CoordinateID *coordinateID*) const

Gives direct access to the coordinate offset between desired geometrical value of the position and the desired position.

See also:
setCoordinateCurrentPosition.

References isCoordinateControllable().

Referenced by Vx::VxStateConstraint::loadState().

int VxConstraint::getCoordinateReferencePartIndex (CoordinateID *coordinate*) const

A coordinate refers to a specific axis to compute its position.

This axis must be specified as being relative to a given part reference frame. The method returns the part's frame the coordinate is based on.

VxConstraint::CoordinateTypeEnum VxConstraint::getCoordinateType (CoordinateID *coordinate*) const

Returns an enumerated type describing whether the controllable coordinate with index *coordinate* is a position, a pure angle, or mixed type ((!) * kCoordinateDistance).

This is usefull if, for example, you want to know if the value returned by getPartForce is a force or a torque.

References **kCoordinateAngular**, **kCoordinateDistance**, and **kCoordinateLinear**.

VxReal VxConstraint::getCoordinateVelocity (CoordinateID *coordinate*) const

Returns the velocity of controllable coordinate with index *coordinate*.

Use isAngular(coordinate) to know if returned velocity is linear or angular.

See also:
VxConstraint::isAngular(CoordinateID coordinate).

Warning:
The returned velocity is valid only if the coordinate computation for the given degree of freedom is activated, or if the degree of freedom is locked or motorized

Referenced by **Vx::VxLockVelocityController::notifyEvent()**, and **Vx::VxMotorController::notifyEvent()**.

VxPart * VxConstraint::getForcePartKinematicResponse ( ) [protected]

Return the part being forced kinematic,.

See also:
setForcePartKinematicResponse().

VxReal VxConstraint::getLimitDamping (CoordinateID *coordinate*, LimitEnum *index*) const

Returns the constraint damping for the lower or upper range limit constraint on the controllable coordinate with index *coordinate*.

See also:

VxConstraint::setLimitDamping()

VxReal VxConstraint::getLimitMaximumForce (CoordinateID *coordinate*, LimitEnum *index*) const

Returns limit's spring max force.

VxReal VxConstraint::getLimitPosition (CoordinateID *coordinate*, LimitEnum *index*) const

Returns the value of the lower or upper range for a controllable coordinate with index *coordinate*.

index = VxConstraint::kLimitLower for the lower range limit; *index* = VxConstraint::kLimitUpper for the upper range limit.

Referenced by **Vx::VxStateConstraint::loadState()**, **Vx::VxUniverse::printContent()**, and **Vx::VxUniverse::verifyDynamics()**.

VxReal VxConstraint::getLimitRestitution (CoordinateID *coordinate*, LimitEnum *index*) const

Returns limit restitution parameters.

See also:

VxConstraint::setLimitRestitution

bool VxConstraint::getLimitsActive (CoordinateID *coordinate*) const

Returns true if range limit constraints for controllable coordinate with index *coordinate* has been set and are active.

See also:

Referenced by **Vx::VxStateConstraint::loadState()**, **Vx::VxUniverse::printContent()**, and **Vx::VxUniverse::verifyDynamics()**.

VxReal VxConstraint::getLimitVelocity (CoordinateID *coordinate*, LimitEnum *index*) const

Returns the velocity of the lower or upper range limit constraint of the controllable coordinate with index *coordinate*.

See also:
VxConstraint::LimitEnum.

Referenced by **Vx::VxStateConstraint::loadState()**.

VxReal VxConstraint::getLockDamping (CoordinateID *coordinate*) const

Returns the constraint damping on the lock constraint of controllable coordinate with index *coordinate*.

See also:
setLockStiffnessAndDamping

Referenced by **Vx::VxUniverse::printContent()**, and **setLockStiffness()**.

VxReal VxConstraint::getLockMaximumForce (CoordinateID *coordinate*) const

Returns the value of the maximum allowed force for the lock constraint on the controllable coordinate with index *coordinate*.

See also:
VxConstraint::setLockParameters()

Referenced by **Vx::VxUniverse::printContent()**, and **Vx::VxUniverse::verifyDynamics()**.

VxReal VxConstraint::getLockPosition (CoordinateID *coordinate*) const

Returns target coordinate for the lock constraint on the controllable coordinate with index *coordinate*.

Use **VxConstraint::getCoordinateCurrentPosition()** to find the actual coordinate of this controllable degree of freedom.

Referenced by **Vx::VxStateConstraint::loadState()**, **Vx::VxLockController::notifyEvent()**, **Vx::VxUniverse::printContent()**, and **Vx::VxUniverse::verifyDynamics()**.

VxReal VxConstraint::getLockStiffness (CoordinateID *coordinate*) const

Returns the constraint stiffness on the lock constraint of controllable coordinate with index *coordinate*.

See also:
setLockStiffnessAndDamping

Referenced by **Vx::VxUniverse::printContent()**, and **setLockDamping()**.

int Vx::VxConstraint::getMaxPartCount ( ) const `[inline]`

Returns the maximum number of **VxPart** objects that can be involved by this constraint.

Referenced by **Vx::VxWeakConstraintAnalyzer::acceptConstraint()**, **enable()**, **getPartAttachmentAxes()**, **getPartAttachmentAxesRel()**, **getPartAttachmentPosition()**, **getPartAttachmentPositionCOMRel()**, **getPartAttachmentPositionRel()**, **getPartCount()**, **Vx::VxStateConstraint::loadState()**, **Vx::VxUniverse::printContent()**, **Vx::VxStateConstraint::restoreState()**, **Vx::VxMotorConstraint::setCoordinateReferencePartIndex()**, **setCoordinateReferencePartIndex()**, **setForcePartKinematicResponseIndex()**, **setPart()**, **setPartAttachmentAxes()**, **setPartAttachmentAxesRel()**, **setPartAttachmentAxis()**, **setPartAttachmentAxisRel()**, **setPartAttachmentPosition()**, **setPartAttachmentPositionCOMRel()**, **setPartAttachmentPositionRel()**, **updateAttachmentFromPart()**, and **Vx::VxPart::~VxPart()**.

VxReal VxConstraint::getMotorDesiredVelocity (CoordinateID *coordinate*) const

Returns the desired velocity of the limited-force motor for the controllable coordinate with index *coordinate*.

See also:
VxConstraint::setMotorParameters()

Referenced by **Vx::VxStateConstraint::loadState()**, **Vx::VxMotorController::notifyEvent()**, **Vx::VxUniverse::printContent()**, and **Vx::VxUniverse::verifyDynamics()**.

VxReal VxConstraint::getMotorLoss (CoordinateID *coordinate*) const

Returns the value of kinetic loss of the limited-force motor for the controllable coordinate with index *coordinate*.

See also:
VxConstraint::setMotorParameters()

Referenced by **Vx::VxMotorController::notifyEvent()**.

VxReal VxConstraint::getMotorMaximumForce (CoordinateID *coordinate*) const

Returns the value of the maximum force (or torque) of the limited-force motor for the controllable coordinate with index *coordinate*.

See also:
VxConstraint::setMotorParameters()

Referenced by **Vx::VxUniverse::printContent()**, and **Vx::VxUniverse::verifyDynamics()**.

VxReal VxConstraint::getMotorMinimumForce (CoordinateID *coordinate*) const

Returns the value of the minimum force (or torque) of the limited-force motor for the controllable coordinate with index *coordinate*.

See also:
VxConstraint::setMotorParameters()

const VxPart* Vx::VxConstraint::getPart (const int *partIndex*) const `[inline]`

Returns the part referenced by *partIndex* that is constrained by this constraint.

This can return a NULL pointer.

VxPart* Vx::VxConstraint::getPart (const int *partIndex*) `[inline]`

Returns the part referenced by *partIndex* that is constrained by this constraint.

This can return a NULL pointer.

Referenced by **Vx::VxWeakConstraintAnalyzer::acceptConstraint()**, **Vx::VxCable::breakCable()**, **VxGraphics::VxVisualizer::displayConstraint()**, **getPartAttachmentAxesRel()**, **getPartAttachmentPositionRel()**, **Vx::VxSumDistance::insertPart()**, **Vx::VxUniverse::printContent()**, **Vx::VxCable::registerAttachmentConstraint()**, **Vx::VxSumDistance::resetDistance()**, **Vx::VxCable::resetPartCount()**, **Vx::VxSumDistance::retrievePart()**, **Vx::VxCable::setCableGeometryType()**, **setForcePartKinematicResponseIndex()**, **Vx::VxRPRO::setPartAttachmentAxes()**, **setPartAttachmentAxesRel()**, **setPartAttachmentAxisRel()**, **setPartAttachmentPositionRel()**, **Vx::VxUniverse::verifyUniverseContent()**, and **Vx::VxPart::~VxPart()**.

void VxConstraint::getPartAttachmentPositionCOMRel (int *partIndex*, VxReal3 *pos*) const

Returns the attachment postion of **VxPart** with index *partIndex* in the coordinate system of the **VxPart** center of mass.

Note:
that the part's com has the same orientation as the part.
that in case the part is merged with another part (cf. VxPart::setMergeParent) the returned position *pos* will be relative to the the center of mass of the entire compound of parts.

References **getMaxPartCount()**.

int VxConstraint::getPartCount ( ) const

Returns the number of non-nil **VxPart** objects currently added to the constraint.

References **getMaxPartCount()**.

Referenced by **VxGraphics::VxVisualizer::displayConstraint()**, and **enable()**.

VxReal VxConstraint::getRelaxationDamping (ConstraintEquationID *c*) const

Returns the value of constraint damping for the basic constraint equation with index *c*.

See also:
setRelaxationParameters.

bool VxConstraint::getRelaxationEnabled (ConstraintEquationID *c*) const

Returns true if the basic constraint equation with index *c* has been relaxed.

See also:
setRelaxationParameters.

Referenced by **getRelaxationParameters()**.

VxReal VxConstraint::getRelaxationLoss (ConstraintEquationID *c*) const

Returns the value of kinetic loss for the basic constraint equation with index *c*.

See also:
setRelaxationParameters.

```
void VxConstraint::getRelaxationMinMaxForce ( ConstraintEquationID c,
                                              VxReal *                minForce,
                                              VxReal *                maxForce
                                            )                         const
```

Gets the minimum and maximum force the constraint equation may apply on the part, this applies only if the equation is relaxed.

minForce < 0 && *maxForce* > 0.

void VxConstraint::getRelaxationParameters (ConstraintEquationID *c*,
VxReal * *stiffness*,
VxReal * *damping*,
VxReal * *loss*,
bool * *enabled*
)

Returns the relaxation paramters used for the basic constraint equation with index *c*.

See also:
setRelaxationParameters.

References **getRelaxationEnabled()**.

Referenced by **Vx::VxUniverse::verifyDynamics()**.

VxReal VxConstraint::getRelaxationStiffness (ConstraintEquationID *c*) const

Returns the value of constraint stiffness for the basic constraint equation with index *c*.

See also:
setRelaxationParameters.

VxUniverse* Vx::VxConstraint::getUniverse ( ) const `[inline]`

Returns the universe the constraint belong to.

Referenced by **enable()**, **Vx::VxWheelTireModelSubscriber::postStep()**, and **Vx::VxUniverse::printContent()**.

bool VxConstraint::getWeak ( )

returns true if the constraint is permanently weak.

Referenced by **Vx::VxWeakConstraintAnalyzer::acceptConstraint()**.

bool VxConstraint::getWeakOneFrame ( )

Returns true if the constraint is weak for the coming step only.

bool VxConstraint::isAngular (CoordinateID *coordinate*) const

Returns true if the controllable coordinate with index *coordinate* is of angular type.

If it is angular, "force" values correspond to torques, and velocities are in radians per second, for example.

bool Vx::VxConstraint::isCoordinateControllable (CoordinateID *coordinate*) const `[inline]`

Returns true if this constraint's coordinate with index *coordinate* is controllable, i.e.

if a lock, motor, or limits can be set on that coordinate.

Referenced by **getCoordinateCurrentPosition()**, **getCoordinateInternalPosition()**, **getCoordinateOffset()**, **recalculateCoordinateCurrentPosition()**, **setCoordinateCurrentPosition()**, **setCoordinateInternalPosition()**, and **setCoordinateOffset()**.

bool Vx::VxConstraint::isEnabled ( ) const `[inline]`

Tests whether a constraint is enabled or disabled.

See also:
VxConstraint::enable()

Referenced by **Vx::VxPart::_applyCOMRelativeTM()**, **VxGraphics::VxVisualizer::displayConstraint()**, **Vx::VxStateConstraint::loadState()**, **Vx::VxUniverse::printContent()**, **Vx::VxAttachmentPoint::setConstraint()**, and **Vx::VxUniverse::verifyDynamics()**.

bool Vx::VxConstraint::isEnabledInternal ( ) const `[inline]`

Tests whether a constraint is internally enabled or disabled.

See also:
VxConstraint::enable()

bool VxConstraint::isLocked (CoordinateID *coordinate*) const

Returns true if a lock constraint for controllable coordinate with index *coordinate* has been set and is active.

See also:
VxConstraint::setLockParameters()

Referenced by Vx::VxUniverse::printContent(), and Vx::VxUniverse::verifyDynamics().

bool VxConstraint::isMotorized (CoordinateID *coordinate*) const

Returns true if a limited-force motor constraint for controllable coordinate with index *coordinate* has been set and is active.

See also:
VxConstraint::setMotorParameters()

Referenced by Vx::VxUniverse::printContent(), and Vx::VxUniverse::verifyDynamics().

bool VxConstraint::isOfType (int *classType*) const

Returns true if the constraint type is of the specified class.

Referenced by VxGraphics::VxVisualizer::displayConstraint().

VxReal VxConstraint::recalculateCoordinateCurrentPosition (CoordinateID *coordinateID*)

Force coordinate position to be recalculated.

This is useful to get the coordinate position after moving a constrained part before doing a step. The new position is returned.

References isCoordinateControllable().

void VxConstraint::resetDynamics ( ) const

Resets all dynamical values associated with a constraint to their default states.

Referenced by Vx::VxStateConstraint::restoreKinematicState(), and Vx::VxStateConstraint::restoreState().

**bool VxConstraint::setControl (CoordinateID *coordinate*,
CoordinateControlEnum *control*
)**

Specifies the type of controller applied to the given controllable coordinate.

This method returns true if the previous control mode was different from *control*, and false otherwise.

References **enableBodies()**, **kControlFree**, **kControlLocked**, **kControlMotorized**, and **Vx::VX_MEDIUM_EPSILON**.

Referenced by **Vx::VxWheelTireModelSubscriber::handleWheelContacts()**, **Vx::VxLockVelocityController::notifyEvent()**, **Vx::VxLockController::notifyEvent()**, **Vx::VxMotorController::notifyEvent()**, **Vx::VxMotorForceController::notifyEvent()**, and **Vx::VxStateConstraint::restoreState()**.

void VxConstraint::setCoordinateCurrentPosition (CoordinateID *coordinateID*,
VxReal *newPosOffset*
)

Sets the the zero reference for the controllable degree of freedom with index *coordinateID*, so that the current coordinate value is *newPosOffset* after this method is invoked.

This does not change the actual position or orientation of any modelled object.

References **isCoordinateControllable()**.

void VxConstraint::setCoordinateInternalPosition (CoordinateID *coordinateID*,
VxReal *pos*
)

This will set the internal position of the coordinate.

This is only an initialisation call since the position will be overriden at next step. The utility of this method is to restore the appropriate winding number of an angular coordinate since there are no geometrical way of calculating this information.

References **isCoordinateControllable()**.

Referenced by **Vx::VxStateConstraint::restoreKinematicState()**, and **Vx::VxStateConstraint::restoreState()**.

void VxConstraint::setCoordinateOffset (CoordinateID *coordinateID*,
VxReal *offset*
)

Gives direct access to the coordinate offset between desired geometrical value of the position and the desired position.

See also:
setCoordinateCurrentPosition.

References **isCoordinateControllable()**.

Referenced by **Vx::VxCable::reset()**, **Vx::VxStateConstraint::restoreKinematicState()**, and **Vx::VxStateConstraint::restoreState()**.

void VxConstraint::setCoordinateReferencePartIndex (CoordinateID *coordinate*, int *partReferenceIndex*) [virtual]

Allows changing the reference part index.

Note:
this method only works for coordinate of linear type.
also, if friction is enabled on this coordinate, VxConstraintFriction::setPartRef will be automatically adapted.

Reimplemented in **Vx::VxMotorConstraint**.

References **getCoordinateFriction()**, **getMaxPartCount()**, and **Vx::VxConstraintFriction::setPartReferenceIndex()**.

void VxConstraint::setForcePartKinematicResponse (VxPart * *p*) [protected]

Forcing one of the part in constraint to be considered as kinematic so that the other constraint parts will see it as if it had infinite mass.

As a result the two constrained parts may not be in the same partition.

Referenced by **setForcePartKinematicResponseIndex()**, and **setPart()**.

void VxConstraint::setForcePartKinematicResponseIndex (int *partIndex*)

Sets index of part for which the constraint will consider as being kinematic so that no force will be added to it.

Use -1 for none, this is the default.

References **getMaxPartCount()**, **getPart()**, and **setForcePartKinematicResponse()**.

void VxConstraint::setLimitDamping (CoordinateID *coordinate*, LimitEnum *index*, VxReal *damping*)

Sets the constraint damping for the lower or upper range limit constraint on the controllable coordinate with index *coordinate*.

Damping is forced to be non-negative and defaults to 0.

See also:
VxUniverse::getCriticalDamping to get critical damping associated with a given stiffness.

```
void VxConstraint::setLimitMaximumForce ( CoordinateID  coordinate,
                                          LimitEnum     index,
                                          VxReal        MaxForce
                                        )
```

Sets limit's spring max force.

Default value is infinity.

```
void VxConstraint::setLimitPosition ( CoordinateID  coordinate,
                                      LimitEnum     index,
                                      VxReal        limit
                                    )
```

Sets the lower or upper value of the range of a controllable coordinate with index *coordinate* to the value *limit*.

index = VxConstraint::kLimitLower for the lower range limit; *index* = VxConstraint::kLimitUpper for the upper range limit.

References **enableBodies()**.

Referenced by **Vx::VxStateConstraint::restoreKinematicState()**, and **Vx::VxStateConstraint::restoreState()**.

```
void VxConstraint::setLimitRestitution ( CoordinateID  coordinate,
                                         LimitEnum     index,
                                         VxReal        restitution
                                       )
```

Sets the restitution property of the limit.

Restitution is forced to be in the range zero to one inclusive: the initial value is one. Beware of using restitution value of 1! Energy conservation in Vortex is not exact. With restitution equal to 1.0, it is possible to increase the energy of the system by a small amount each time a limit is crossed, leading to an explosion.

```
void VxConstraint::setLimitsActive ( CoordinateID  coordinate,
```

```
                                bool        activate
                              )
```

Call this method to enable or not the limit.

Activates or deactivates enforcement of the range limits on the controllable coordinate with index *coordinate*.

References **enableBodies()**.

Referenced by **Vx::VxCable::reset()**, and **Vx::VxStateConstraint::restoreState()**.

```
void VxConstraint::setLimitStiffness ( CoordinateID  coordinate,
                                       LimitEnum     index,
                                       VxReal        stiffness
                                     )
```

Sets the constraint stiffness on the range limit constraint.

Stiffness is forced to be greater than or equal to zero. The default value of the argument *stiffness* is VX_INFINITY.

See also:
VxUniverse::getCriticalDamping to get critical damping associated with a given stiffness.

References **enableBodies()**.

```
void VxConstraint::setLimitVelocity ( CoordinateID  coordinate,
                                      LimitEnum     index,
                                      VxReal        limitVel
                                    )
```

Sets the velocity of the lower or upper range limits, for the controllable coordinate with index *coordinate*.

(!)

See also:
VxConstraint::LimitEnum.

References **enableBodies()**, and **Vx::VX_MEDIUM_EPSILON**.

Referenced by **Vx::VxLimitMotorController::notifyEvent()**, and **Vx::VxStateConstraint::restoreState()**.

```
void VxConstraint::setLockDamping ( CoordinateID coordinate,
                                    VxReal       damping
                                  )
```

Sets the constraint damping on the lock constraint of controllable coordinate with index *coordinate*.

See also:
setLockStiffnessAndDamping

References getLockStiffness().

```
void VxConstraint::setLockMinimumAndMaximumForce ( CoordinateID coordinate,
                                                   VxReal       minForce,
                                                   VxReal       maxForce
                                                 )
```

Sets the minimum and maximum force that vortex will apply to enforce the position lock constraint on the controllable coordinate with index *coordinate*.

By default minForce == - maxForce.

Warning:
minForce must be lower then *maxForce*.

Referenced by Vx::VxCableProperties::setCompressionMaxForce(), and Vx::VxCableProperties::setElongationMaxForce().

```
void VxConstraint::setLockParameters ( CoordinateID coordinate,
                                       VxReal       lockValue,
                                       VxReal       maxForce,
                                       VxReal       stiffness = VX_INFINITY,
                                       VxReal       damping = 0,
                                       VxReal       velocity = 0
                                     )
```

Configures a lock controller for the controllable coordinate with index *coordinate*.

The lock controller allows both fixed and dynamic modes.

In fixed mode, the controlled coordinate will be constrained to the value of *lockValue*. In dynamic mode, the target position starts at *lockValue*

but moves with *velocity* with time. The dynamic mode is useful if you want to minimize the constraint slip observed when using the limited-force motor.

maxForce is the maximum force that vortex will applied to enforce the lock position. Defaults to VX_INFINITY.

stiffness and *damping* are the contraint stiffness and damping parameters which default to VX_INFINITY and 0, respectively.

See also:
VxConstraint::setMotorParameters()

References **enableBodies()**, and **Vx::VX_MEDIUM_EPSILON**.

void VxConstraint::setLockStiffness (CoordinateID *coordinate*,
VxReal *stiffness*
)

Sets the constraint stiffness on the lock constraint of controllable coordinate with index *coordinate*.

See also:
setLockStiffnessAndDamping

References **getLockDamping()**.

void VxConstraint::setLockStiffnessAndDamping (CoordinateID *coordinate*,
VxReal *stiffness*,
VxReal *damping*
)

Sets the constraint relaxation paramters of the lock constraint for the controllable coordinate with index *coordinate*.

Use this to get a soft lock constraint. Note that the force computed is still subject to be clamped by the maximum force value if it is finite.

See also:
VxConstraint::setLockMaximumForce.

void VxConstraint::setLowerLimit (CoordinateID *coordinateID*,
VxReal *limitPos*,
VxReal *limitVel* = 0,
VxReal *restitution* = 0,

```
                                VxReal      stiffness = sDefaultStiffness,
                                VxReal      damping = sDefaultDamping
                              )
```

Sets the configuration of the lower range limit on the controllable constraint coordinate with index *coordinateID*.

The value of *limitPos* is the coordinate value of the lower range limit.

The value of *limitVel* is the target velocity for this coordinate (!). Defaults to 0.

The value of *restitution* is the restitution parameter for the range limit; this is used when the given degree of freedom "collides" with the given range limit. Defaults to 0.

stiffness is the value of constraint stiffness for the range limit constraint when *limitPos* is reached or exceeded. Defaults to VX_INFINITY.

damping is the value of constraint damping for the range limit constraint when *limitPos* is reached or exceeded. Defaults to 0.

References **Vx::VX_MEDIUM_EPSILON**.

```
void VxConstraint::setMotorDesiredVelocity ( CoordinateID  coordinate,
                                             VxReal        desiredVelocity
                                           )
```

Sets the desired velocity of the limited-force motor for the controllable coordinate with index *coordinate*.

See also:
VxConstraint::setMotorParameters().

References **enableBodies()**.

Referenced by **Vx::VxWheelTireModelSubscriber::handleWheelContacts()**, **Vx::VxMotorForceController::notifyEvent()**, and **Vx::VxStateConstraint::restoreState()**.

```
void VxConstraint::setMotorLoss ( CoordinateID  coordinate,
                                  VxReal        loss
                                )
```

Sets the kinetic loss coefficient of the limited-force motor for the controllable coordinate with index *coordinate*.

The kinetic loss coefficient is the inverse of a viscous drag coefficient. A positive value of *loss* introduces slippage in the motor so that the controllable coordinate will ramp up exponentiall to the target value. The half-life of the ramp-up is proportional to the value of *loss*.

See also:
VxConstraint::setMotorParameters().

References enableBodies().

```
void VxConstraint::setMotorMaximumForce ( CoordinateID  coordinate,
                                          VxReal        maxForce
                                        )
```

Sets the maximum allowed force of the limited-force motor for the controllable coordinate with index *coordinate*.

See also:
VxConstraint::setMotorParameters().

References enableBodies().

```
void VxConstraint::setMotorMinimumAndMaximumForce ( CoordinateID  coordinate,
                                                    VxReal        minForce,
                                                    VxReal        maxForce
                                                  )
```

Sets the maximum allowed force of the limited-force motor for the controllable coordinate with index *coordinate*.

This method allows to specify a minimum and a maximum force on the motor, by default minForce == - maxForce.

Warning:
minForce must be lower then *maxForce*.

References enableBodies().

Referenced by Vx::VxWheelTireModelSubscriber::handleWheelContacts(), and Vx::VxMotorForceController::notifyEvent().

```
void VxConstraint::setMotorParameters ( CoordinateID  coordinate,
                                        VxReal        desiredVelocity,
                                        VxReal        maxForce,
                                        VxReal        loss = 0
                                      )
```

Sets the parameters for a limited-force motor on the controllable coordinate with index *coordinate*.

The limited-force motor constraint will attempt to achieve the desired velocity *desiredVelocity* for the given controllable coordinate, but only if the magnitude of the constraint force required to do so is less than the given *maxForce*.

maxForce must be non-negative. A value of zero deactivates the motor. Otherwise, the motor is activated after the method is invoked.

A positive value of *loss* makes the motor slip so that the actual velocity of the controllable coodinate ramps up exponnentially towards the desired value. The a half life of the ramp-up is proportional to *loss*.

In the case where you have both a joint limit and a motor, there can be chatter if the motor is driving against the limit. To prevent this, the desired velocity of the motor is reset to 0 in the case where the limit stiffness parameter is infinity, irrespective of the restitution parameters. This leads to stable collisions with joint limits. If the limit stiffness is not infinity, you will get chatter as the motor and the range limit dynamics are opposing each other.

When the value of *desiredVelocity* is 0, the motor acts as a brake.

This method does *not* wake attached **VxPart** objects: see VxPart::wakeDynamics(0).

References **enableBodies()**.

Referenced by **Vx::VxMotorController::notifyEvent()**.

void VxConstraint::setMotorStopAtLock (CoordinateID *coordinate*,
bool *b*
)

Force the coordinate control to switch from kControlMotorized to kControlLocked when constraint position meets lock position.

Note:
This feature requires lock velocity to be 0.

void VxConstraint::setName (const char * *name*) [virtual]

Sets the constraint name.

Reimplemented from **Vx::VxBase**.

References **Vx::VxBase::getName()**.

void VxConstraint::setPart (int *partIndex*,
VxPart * *part*
) [virtual]

Sets the **VxPart** with index *partIndex* to the given *part*.

If a **VxPart** has already been specified for the given index, it will be replaced by the new one.

Warning:
partIndex must greater or equal to zero and smaller than the value returned by **getMaxPartCount()**

Reimplemented in **Vx::VxContactGear**, **Vx::VxDifferential**, and **Vx::VxRPRO**.

References **enable()**, **EVENT_ADD_PART**, **EVENT_REMOVE_PART**, **getMaxPartCount()**, **Vx::VxBase::getName()**, and **setForcePartKinematicResponse()**.

Referenced by **Vx::VxSumDistance::insertPart()**, **Vx::VxSumDistance::retrievePart()**, **setPartAndAttachment()**, **setPartAndAttachmentRel()**, **Vx::VxGearRatio::VxGearRatio()**, **Vx::VxSumDistance::VxSumDistance()**, **Vx::VxWinch::VxWinch()**, and **Vx::VxPart::~VxPart()**.

```
void VxConstraint::setPartAttachmentAxes ( int             partIndex,
                                           const VxReal3   p,
                                           const VxReal3   s
                                         )                 [virtual]
```

Sets the primary, *p*, and secondary, *s*, constraint axes for **VxPart** with index *partIndex* in this constraint, in the world reference frame.

These two axes must be orthogonal.

Together, the vectors *p* and *s* define an orthonormal local attachment coordinate system for the given **VxPart**; the orthogonal complement of the two given vectors is computed internally. If the secondary axis, *s*, is set to {0,0,0}, Vortex will construct an orthonormal system by rotating the standard x,y, unit vectors so they become orthogonal to the given primary axis.

Warning:
Constraint axes must be configured properly before the stepping the **VxUniverse**.

Reimplemented in **Vx::VxCarWheel**, **Vx::VxRPRO**, and **Vx::VxWheelConstraint**.

References **getMaxPartCount()**.

Referenced by **Vx::VxWheelTireModelSubscriber::handleWheelContacts()**, **setPartAndAttachment()**, **setPartAttachmentAxis()**, **updateAttachmentFromPart()**, and **Vx::VxDoubleHinge::VxDoubleHinge()**.

```
void VxConstraint::setPartAttachmentAxis ( int             partIndex,
```

```
                                    const VxReal3  primary
                                )                  [virtual]
```

Sets the primary attachment axes in the **VxPart** with index *partIndex* in this constraint, in the world reference frame.

The secondary axes is computed automatically.

References **getMaxPartCount()**, **Vx::VxVector3::planeSpace()**, and **setPartAttachmentAxes()**.

```
void VxConstraint::setPartAttachmentAxisRel ( int        partIndex,
                                              const VxReal3  primary
                                            )                  [virtual]
```

Sets the primary attachment axes in the **VxPart** with index *partIndex* in this constraint, in the reference frame of the given **VxPart**.

The secondary axes is computed automatically.

References **getMaxPartCount()**, **Vx::VxPart::getMergeRoot()**, and **getPart()**.

Referenced by **Vx::VxWinch::VxWinch()**.

```
void VxConstraint::setPartAttachmentPosition ( int      partIndex,
                                               VxReal   x,
                                               VxReal   y,
                                               VxReal   z
                                             )          [virtual]
```

Sets the attachment point of the **VxPart** with index *partIndex* in the constraint.

See also:
VxConstraint::setPartAttachmentPosition(int partIndex, VxReal3 pos)

Reimplemented in **Vx::VxWheelConstraint**.

References **getMaxPartCount()**.

```
void VxConstraint::setPartAttachmentPosition ( int        partIndex,
                                               const VxReal3  pos
                                             )                  [virtual]
```

Sets the center of the attachment frame for the constrained **VxPart** with index *partIndex* in this constraint.

The center of the given attachment frame is expressed in world coordinates in *pos*. This is considered to be the current location of the center of the attachment. The method computes the corresponding body coordinates of *pos* and uses that value in future computations.

Warning:

The user must explicitly set the attachment positions of all **VxConstraint** objects before stepping a given **VxUniverse**. Attachment positions are set to the origin of each **VxPart** by default. In the case of a ball and socket joint for example, this would result in specifying that the center of two **VxPart** objects should coincide. If the **VxPart** objects also have a collision geometry attached to them, this leads to a conflicted configuration with unpredictable behavior.

References **getMaxPartCount()**.

Referenced by **Vx::VxWheelTireModelSubscriber::handleWheelContacts()**, **setPartAndAttachment()**, **updateAttachmentFromPart()**, and **Vx::VxDoubleHinge::VxDoubleHinge()**.

```
void VxConstraint::setPartAttachmentPositionCOMRel ( int      partIndex,
                                                     VxReal   x,
                                                     VxReal   y,
                                                     VxReal   z
                                                   )
```

Sets the attachment postion of **VxPart** with index *partIndex* in this constraint, directly in the coordinate system of the **VxPart** center of mass.

Use this method if you already know the location of the attachment point for the given **VxPart** relative to its center of mass.

Note:

that the part's com has the same orientation as the part.
that in case the part with the given index is merged with another part (cf. VxPart::setMergeParent) the center of mass corresponds actually to the the center of mass of the the complete part compound.

References **getMaxPartCount()**.

```
void VxConstraint::setPartAttachmentPositionCOMRel ( int              partIndex,
                                                     const VxReal3    pos
                                                   )
```

Sets the attachment postion of **VxPart** with index *partIndex* in this constraint, directly in the coordinate system of the **VxPart** center of mass.

Use this method if you already know the location of the attachment point for the given **VxPart** relative to its center of mass.

Note:
that the part's com has the same orientation as the part.
that in case the part is merged with another part (cf. VxPart::setMergeParent) the given position *pos* is set relative to the the center of mass of the entire compound of parts.

References **getMaxPartCount()**.

```
void VxConstraint::setPartAttachmentPositionRel ( int      partIndex,
                                                  VxReal   x,
                                                  VxReal   y,
                                                  VxReal   z
                                                )
```

Sets the attachment postion of **VxPart** with index *partIndex* in this constraint, directly in the coordinate system of the **VxPart**.

Use this method if you already know the location of the attachment point for the given **VxPart** in its own frame of reference.

References **setPartAttachmentPositionRel()**.

```
void VxConstraint::setPartAttachmentPositionRel ( int             partIndex,
                                                  const VxReal3   pos
                                                )
```

Sets the attachment postion of **VxPart** with index *partIndex* in this constraint, directly in the coordinate system of the **VxPart**.

Use this method if you already know the location of the attachment point for the given **VxPart** in its own frame of reference.

References **getMaxPartCount()**, **Vx::VxPart::getMergeRoot()**, and **getPart()**.

Referenced by **Vx::VxSumDistance::insertPart()**, **Vx::VxCable::reset()**, **Vx::VxStateConstraint::restoreKinematicState()**, **Vx::VxSumDistance::retrievePart()**, **setPartAndAttachmentRel()**, **setPartAttachmentPositionRel()**, **Vx::VxAngular3::VxAngular3()**, **Vx::VxCylindrical::VxCylindrical()**, **Vx::VxDifferential::VxDifferential()**, **Vx::VxGearRatio::VxGearRatio()**, **Vx::VxScrewJoint::VxScrewJoint()**, **Vx::VxSpring::VxSpring()**, **Vx::VxSumDistance::VxSumDistance()**, and **Vx::VxWinch::VxWinch()**.

```
void Vx::VxConstraint::setParts ( VxPart *  part1,
                                  VxPart *  part2
                                )          [inline]
```

Sets the first and the second part (index 0 and 1) attached to this constraint.

Referenced by Vx::VxWheelTireModelSubscriber::handleWheelContacts(), Vx::VxAngular3::VxAngular3(), Vx::VxBallAndSocket::VxBallAndSocket(), Vx::VxConeConstraint::VxConeConstraint(), Vx::VxContactGear::VxContactGear(), Vx::VxCylindrical::VxCylindrical(), Vx::VxDoubleHinge::VxDoubleHinge(), Vx::VxGearRatio::VxGearRatio(), Vx::VxHinge::VxHinge(), Vx::VxScrewJoint::VxScrewJoint(), Vx::VxSpring::VxSpring(), and Vx::VxWheelConstraint::VxWheelConstraint().

```
void VxConstraint::setRelaxationDamping ( ConstraintEquationID c,
                                          VxReal               newdamp
                                        )
```

Sets the value of *damping* for the basic constraint equation with index *c*.

See also:
setRelaxationParameters.

```
void VxConstraint::setRelaxationLoss ( ConstraintEquationID c,
                                       VxReal               newloss
                                     )
```

Sets the value of kinetic loss to *loss* for the basic constraint equation with index *c*.

See also:
setRelaxationParameters.

```
void VxConstraint::setRelaxationMinMaxForce ( ConstraintEquationID c,
                                              VxReal               minForce,
                                              VxReal               maxForce
                                            )
```

Sets the minimum and maximum force the constraint equation may apply on the part, this applies only if the equation is relaxed.

minForce < 0 && *maxForce* > 0. Default values are -VX_INFINITY and VX_INFINITY.

```
void VxConstraint::setRelaxationParameters ( ConstraintEquationID c,
                                             VxReal               stiffness,
                                             VxReal               damping,
                                             VxReal               loss,
                                             bool                 enabled
                                           )
```

Sets the relaxation parameters of the basic constraint equation with index *c*.

Basic constaint equation controlling relative position or orientation are relaxed using stiffness and damping parameters. Basic constraint equations controlling relative velocity or angular velocity are relaxed using kinetic loss parameter. Please refer to the specific constraint ConstraintEquation enum for information about a given constraint type.

See also:
VxUniverse::getCriticalDamping to get critical damping associated with a given stiffness.

References **enableRelaxation()**.

```
void VxConstraint::setRelaxationStiffness ( ConstraintEquationID  c,
                                            VxReal                newstiff
                                          )
```

Sets the value of *stiffness* for the basic constraint equation with index *c*.

See also:
setRelaxationParameters.

```
void VxConstraint::setRelaxationStiffnessAndDamping ( ConstraintEquationID  c,
                                                      VxReal                stiffness,
                                                      VxReal                damping
                                                    )
```

Sets the value of stiffness and damping of basic constraint equations with index *c*.

See also:
setRelaxationParameters.

```
void VxConstraint::setUpperLimit ( CoordinateID  coordinate,
                                   VxReal        limitPos,
                                   VxReal        limitVel = 0,
                                   VxReal        restitution = 0,
                                   VxReal        stiffness = sDefaultStiffness,
                                   VxReal        damping = sDefaultDamping
                                 )
```

Sets the configuration of the upper range limit on the controllable constraint coordinate with index *coordinateID*.

The value of *limitPos* is the coordinate value of the upper range limit.

The value of *limitVel* is the target velocity for this coordinate (!). Defaults to 0.

The value of *restitution* is the restitution parameter for the range limit; this is used when the given degree of freedom collides with the given range limit. Defaults to 0.

stiffness is the value of constraint stiffness for the range limit constraint when *limitPos* is reached or exceeded. Defaults to VX_INFINITY.

damping is the value of constraint damping for the range limit constraint when *limitPos* is reached or exceeded. Defaults to 0.

References **Vx::VX_MEDIUM_EPSILON**.

void VxConstraint::setWeak (bool *b*)

Indicates to the constraint solver that the constraint is permanently weak.

This forces the constraint to be secondary and allows the partitioner to segment the partitions onto smaller partitions to take advantage of solver multi-threading capability.

void VxConstraint::setWeakOneFrame ( )

Indicates to the constraint solver that the constraint is weak for the coming step.

This forces the constraint to be secondary and allows the partitioner to segment the partitions onto smaller partitions to take advantage of solver multi-threading capability.

Referenced by **Vx::VxWeakConstraintAnalyzer::acceptConstraint()**.

void VxConstraint::updateAttachmentFromPart (int *partIndex*)

Resets the attachment position and axes from given part.

Can be used when that part moved to make the constraint follow that part. Assumes valid attachments were set previously.

References **getMaxPartCount()**, **getPartAttachmentAxes()**, **getPartAttachmentPosition()**, **setPartAttachmentAxes()**, and **setPartAttachmentPosition()**.