;; Auto-generated. Do not edit!

(when (boundp 'gazebo\_msgs\_new::ApplyBodyWrench)

(if (not (find-package "GAZEBO\_MSGS\_NEW"))

(make-package "GAZEBO\_MSGS\_NEW"))

(shadow 'ApplyBodyWrench (find-package "GAZEBO\_MSGS\_NEW")))

(unless (find-package "GAZEBO\_MSGS\_NEW::APPLYBODYWRENCH")

(make-package "GAZEBO\_MSGS\_NEW::APPLYBODYWRENCH"))

(unless (find-package "GAZEBO\_MSGS\_NEW::APPLYBODYWRENCHREQUEST")

(make-package "GAZEBO\_MSGS\_NEW::APPLYBODYWRENCHREQUEST"))

(unless (find-package "GAZEBO\_MSGS\_NEW::APPLYBODYWRENCHRESPONSE")

(make-package "GAZEBO\_MSGS\_NEW::APPLYBODYWRENCHRESPONSE"))

(in-package "ROS")

(if (not (find-package "GEOMETRY\_MSGS"))

(ros::roseus-add-msgs "geometry\_msgs"))

(defclass gazebo\_msgs\_new::ApplyBodyWrenchRequest

:super ros::object

:slots (\_body\_name \_reference\_frame \_reference\_point \_wrench \_start\_time \_duration ))

(defmethod gazebo\_msgs\_new::ApplyBodyWrenchRequest

(:init

(&key

((:body\_name \_\_body\_name) "")

((:reference\_frame \_\_reference\_frame) "")

((:reference\_point \_\_reference\_point) (instance geometry\_msgs::Point :init))

((:wrench \_\_wrench) (instance geometry\_msgs::Wrench :init))

((:start\_time \_\_start\_time) (instance ros::time :init))

((:duration \_\_duration) (instance ros::time :init))

)

(send-super :init)

(setq \_body\_name (string \_\_body\_name))

(setq \_reference\_frame (string \_\_reference\_frame))

(setq \_reference\_point \_\_reference\_point)

(setq \_wrench \_\_wrench)

(setq \_start\_time \_\_start\_time)

(setq \_duration \_\_duration)

self)

(:body\_name

(&optional \_\_body\_name)

(if \_\_body\_name (setq \_body\_name \_\_body\_name)) \_body\_name)

(:reference\_frame

(&optional \_\_reference\_frame)

(if \_\_reference\_frame (setq \_reference\_frame \_\_reference\_frame)) \_reference\_frame)

(:reference\_point

(&rest \_\_reference\_point)

(if (keywordp (car \_\_reference\_point))

(send\* \_reference\_point \_\_reference\_point)

(progn

(if \_\_reference\_point (setq \_reference\_point (car \_\_reference\_point)))

\_reference\_point)))

(:wrench

(&rest \_\_wrench)

(if (keywordp (car \_\_wrench))

(send\* \_wrench \_\_wrench)

(progn

(if \_\_wrench (setq \_wrench (car \_\_wrench)))

\_wrench)))

(:start\_time

(&optional \_\_start\_time)

(if \_\_start\_time (setq \_start\_time \_\_start\_time)) \_start\_time)

(:duration

(&optional \_\_duration)

(if \_\_duration (setq \_duration \_\_duration)) \_duration)

(:serialization-length

()

(+

;; string \_body\_name

4 (length \_body\_name)

;; string \_reference\_frame

4 (length \_reference\_frame)

;; geometry\_msgs/Point \_reference\_point

(send \_reference\_point :serialization-length)

;; geometry\_msgs/Wrench \_wrench

(send \_wrench :serialization-length)

;; time \_start\_time

8

;; duration \_duration

8

))

(:serialize

(&optional strm)

(let ((s (if strm strm

(make-string-output-stream (send self :serialization-length)))))

;; string \_body\_name

(write-long (length \_body\_name) s) (princ \_body\_name s)

;; string \_reference\_frame

(write-long (length \_reference\_frame) s) (princ \_reference\_frame s)

;; geometry\_msgs/Point \_reference\_point

(send \_reference\_point :serialize s)

;; geometry\_msgs/Wrench \_wrench

(send \_wrench :serialize s)

;; time \_start\_time

(write-long (send \_start\_time :sec) s) (write-long (send \_start\_time :nsec) s)

;; duration \_duration

(write-long (send \_duration :sec) s) (write-long (send \_duration :nsec) s)

;;

(if (null strm) (get-output-stream-string s))))

(:deserialize

(buf &optional (ptr- 0))

;; string \_body\_name

(let (n) (setq n (sys::peek buf ptr- :integer)) (incf ptr- 4) (setq \_body\_name (subseq buf ptr- (+ ptr- n))) (incf ptr- n))

;; string \_reference\_frame

(let (n) (setq n (sys::peek buf ptr- :integer)) (incf ptr- 4) (setq \_reference\_frame (subseq buf ptr- (+ ptr- n))) (incf ptr- n))

;; geometry\_msgs/Point \_reference\_point

(send \_reference\_point :deserialize buf ptr-) (incf ptr- (send \_reference\_point :serialization-length))

;; geometry\_msgs/Wrench \_wrench

(send \_wrench :deserialize buf ptr-) (incf ptr- (send \_wrench :serialization-length))

;; time \_start\_time

(send \_start\_time :sec (sys::peek buf ptr- :integer)) (incf ptr- 4) (send \_start\_time :nsec (sys::peek buf ptr- :integer)) (incf ptr- 4)

;; duration \_duration

(send \_duration :sec (sys::peek buf ptr- :integer)) (incf ptr- 4) (send \_duration :nsec (sys::peek buf ptr- :integer)) (incf ptr- 4)

;;

self)

)

(defclass gazebo\_msgs\_new::ApplyBodyWrenchResponse

:super ros::object

:slots (\_success \_status\_message ))

(defmethod gazebo\_msgs\_new::ApplyBodyWrenchResponse

(:init

(&key

((:success \_\_success) nil)

((:status\_message \_\_status\_message) "")

)

(send-super :init)

(setq \_success \_\_success)

(setq \_status\_message (string \_\_status\_message))

self)

(:success

(&optional \_\_success)

(if \_\_success (setq \_success \_\_success)) \_success)

(:status\_message

(&optional \_\_status\_message)

(if \_\_status\_message (setq \_status\_message \_\_status\_message)) \_status\_message)

(:serialization-length

()

(+

;; bool \_success

1

;; string \_status\_message

4 (length \_status\_message)

))

(:serialize

(&optional strm)

(let ((s (if strm strm

(make-string-output-stream (send self :serialization-length)))))

;; bool \_success

(if \_success (write-byte -1 s) (write-byte 0 s))

;; string \_status\_message

(write-long (length \_status\_message) s) (princ \_status\_message s)

;;

(if (null strm) (get-output-stream-string s))))

(:deserialize

(buf &optional (ptr- 0))

;; bool \_success

(setq \_success (not (= 0 (sys::peek buf ptr- :char)))) (incf ptr- 1)

;; string \_status\_message

(let (n) (setq n (sys::peek buf ptr- :integer)) (incf ptr- 4) (setq \_status\_message (subseq buf ptr- (+ ptr- n))) (incf ptr- n))

;;

self)

)

(defclass gazebo\_msgs\_new::ApplyBodyWrench

:super ros::object

:slots ())

(setf (get gazebo\_msgs\_new::ApplyBodyWrench :md5sum-) "585b9f9618aa0581b207e2f2d90866bc")

(setf (get gazebo\_msgs\_new::ApplyBodyWrench :datatype-) "gazebo\_msgs\_new/ApplyBodyWrench")

(setf (get gazebo\_msgs\_new::ApplyBodyWrench :request) gazebo\_msgs\_new::ApplyBodyWrenchRequest)

(setf (get gazebo\_msgs\_new::ApplyBodyWrench :response) gazebo\_msgs\_new::ApplyBodyWrenchResponse)

(defmethod gazebo\_msgs\_new::ApplyBodyWrenchRequest

(:response () (instance gazebo\_msgs\_new::ApplyBodyWrenchResponse :init)))

(setf (get gazebo\_msgs\_new::ApplyBodyWrenchRequest :md5sum-) "585b9f9618aa0581b207e2f2d90866bc")

(setf (get gazebo\_msgs\_new::ApplyBodyWrenchRequest :datatype-) "gazebo\_msgs\_new/ApplyBodyWrenchRequest")

(setf (get gazebo\_msgs\_new::ApplyBodyWrenchRequest :definition-)

"# Apply Wrench to Gazebo Body.

# via the callback mechanism

# all Gazebo operations are made in world frame

string body\_name # Gazebo body to apply wrench (linear force and torque)

# wrench is applied in the gazebo world by default

# body names are prefixed by model name, e.g. pr2::base\_link

string reference\_frame # wrench is defined in the reference frame of this entity

# use inertial frame if left empty

# frame names are bodies prefixed by model name, e.g. pr2::base\_link

geometry\_msgs/Point reference\_point # wrench is defined at this location in the reference frame

geometry\_msgs/Wrench wrench # wrench applied to the origin of the body

time start\_time # (optional) wrench application start time (seconds)

# if start\_time is not specified, or

# start\_time < current time, start as soon as possible

duration duration # optional duration of wrench application time (seconds)

# if duration < 0, apply wrench continuously without end

# if duration = 0, do nothing

# if duration < step size, apply wrench

# for one step size

================================================================================

MSG: geometry\_msgs/Point

# This contains the position of a point in free space

float64 x

float64 y

float64 z

================================================================================

MSG: geometry\_msgs/Wrench

# This represents force in free space, separated into

# its linear and angular parts.

Vector3 force

Vector3 torque

================================================================================

MSG: geometry\_msgs/Vector3

# This represents a vector in free space.

# It is only meant to represent a direction. Therefore, it does not

# make sense to apply a translation to it (e.g., when applying a

# generic rigid transformation to a Vector3, tf2 will only apply the

# rotation). If you want your data to be translatable too, use the

# geometry\_msgs/Point message instead.

float64 x

float64 y

---

bool success # return true if set wrench successful

string status\_message # comments if available

")

(setf (get gazebo\_msgs\_new::ApplyBodyWrenchResponse :md5sum-) "585b9f9618aa0581b207e2f2d90866bc")

(setf (get gazebo\_msgs\_new::ApplyBodyWrenchResponse :datatype-) "gazebo\_msgs\_new/ApplyBodyWrenchResponse")

(setf (get gazebo\_msgs\_new::ApplyBodyWrenchResponse :definition-)

"# Apply Wrench to Gazebo Body.

# via the callback mechanism

# all Gazebo operations are made in world frame

string body\_name # Gazebo body to apply wrench (linear force and torque)

# wrench is applied in the gazebo world by default

# body names are prefixed by model name, e.g. pr2::base\_link

string reference\_frame # wrench is defined in the reference frame of this entity

# use inertial frame if left empty

# frame names are bodies prefixed by model name, e.g. pr2::base\_link

geometry\_msgs/Point reference\_point # wrench is defined at this location in the reference frame

geometry\_msgs/Wrench wrench # wrench applied to the origin of the body

time start\_time # (optional) wrench application start time (seconds)

# if start\_time is not specified, or

# start\_time < current time, start as soon as possible

duration duration # optional duration of wrench application time (seconds)

# if duration < 0, apply wrench continuously without end

# if duration = 0, do nothing

# if duration < step size, apply wrench

# for one step size

================================================================================

MSG: geometry\_msgs/Point

# This contains the position of a point in free space

float64 x

float64 y

float64 z

================================================================================

MSG: geometry\_msgs/Wrench

# This represents force in free space, separated into

# its linear and angular parts.

Vector3 force

Vector3 torque

================================================================================

MSG: geometry\_msgs/Vector3

# This represents a vector in free space.

# It is only meant to represent a direction. Therefore, it does not

# make sense to apply a translation to it (e.g., when applying a

# generic rigid transformation to a Vector3, tf2 will only apply the

# rotation). If you want your data to be translatable too, use the

# geometry\_msgs/Point message instead.

float64 x

float64 y

---

bool success # return true if set wrench successful

string status\_message # comments if available

")

(provide :gazebo\_msgs\_new/ApplyBodyWrench "585b9f9618aa0581b207e2f2d90866bc")