;; Auto-generated. Do not edit!

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

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

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

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

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

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

(in-package "ROS")

;;//! \htmlinclude WorldState.msg.html

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

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

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

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

(defclass gazebo\_msgs\_new::WorldState

:super ros::object

:slots (\_header \_name \_pose \_twist \_wrench ))

(defmethod gazebo\_msgs\_new::WorldState

(:init

(&key

((:header \_\_header) (instance std\_msgs::Header :init))

((:name \_\_name) (let (r) (dotimes (i 0) (push "" r)) r))

((:pose \_\_pose) (let (r) (dotimes (i 0) (push (instance geometry\_msgs::Pose :init) r)) r))

((:twist \_\_twist) (let (r) (dotimes (i 0) (push (instance geometry\_msgs::Twist :init) r)) r))

((:wrench \_\_wrench) (let (r) (dotimes (i 0) (push (instance geometry\_msgs::Wrench :init) r)) r))

)

(send-super :init)

(setq \_header \_\_header)

(setq \_name \_\_name)

(setq \_pose \_\_pose)

(setq \_twist \_\_twist)

(setq \_wrench \_\_wrench)

self)

(:header

(&rest \_\_header)

(if (keywordp (car \_\_header))

(send\* \_header \_\_header)

(progn

(if \_\_header (setq \_header (car \_\_header)))

\_header)))

(:name

(&optional \_\_name)

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

(:pose

(&rest \_\_pose)

(if (keywordp (car \_\_pose))

(send\* \_pose \_\_pose)

(progn

(if \_\_pose (setq \_pose (car \_\_pose)))

\_pose)))

(:twist

(&rest \_\_twist)

(if (keywordp (car \_\_twist))

(send\* \_twist \_\_twist)

(progn

(if \_\_twist (setq \_twist (car \_\_twist)))

\_twist)))

(:wrench

(&rest \_\_wrench)

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

(send\* \_wrench \_\_wrench)

(progn

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

\_wrench)))

(:serialization-length

()

(+

;; std\_msgs/Header \_header

(send \_header :serialization-length)

;; string[] \_name

(apply #'+ (mapcar #'(lambda (x) (+ 4 (length x))) \_name)) 4

;; geometry\_msgs/Pose[] \_pose

(apply #'+ (send-all \_pose :serialization-length)) 4

;; geometry\_msgs/Twist[] \_twist

(apply #'+ (send-all \_twist :serialization-length)) 4

;; geometry\_msgs/Wrench[] \_wrench

(apply #'+ (send-all \_wrench :serialization-length)) 4

))

(:serialize

(&optional strm)

(let ((s (if strm strm

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

;; std\_msgs/Header \_header

(send \_header :serialize s)

;; string[] \_name

(write-long (length \_name) s)

(dolist (elem \_name)

(write-long (length elem) s) (princ elem s)

)

;; geometry\_msgs/Pose[] \_pose

(write-long (length \_pose) s)

(dolist (elem \_pose)

(send elem :serialize s)

)

;; geometry\_msgs/Twist[] \_twist

(write-long (length \_twist) s)

(dolist (elem \_twist)

(send elem :serialize s)

)

;; geometry\_msgs/Wrench[] \_wrench

(write-long (length \_wrench) s)

(dolist (elem \_wrench)

(send elem :serialize s)

)

;;

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

(:deserialize

(buf &optional (ptr- 0))

;; std\_msgs/Header \_header

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

;; string[] \_name

(let (n)

(setq n (sys::peek buf ptr- :integer)) (incf ptr- 4)

(setq \_name (make-list n))

(dotimes (i n)

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

))

;; geometry\_msgs/Pose[] \_pose

(let (n)

(setq n (sys::peek buf ptr- :integer)) (incf ptr- 4)

(setq \_pose (let (r) (dotimes (i n) (push (instance geometry\_msgs::Pose :init) r)) r))

(dolist (elem- \_pose)

(send elem- :deserialize buf ptr-) (incf ptr- (send elem- :serialization-length))

))

;; geometry\_msgs/Twist[] \_twist

(let (n)

(setq n (sys::peek buf ptr- :integer)) (incf ptr- 4)

(setq \_twist (let (r) (dotimes (i n) (push (instance geometry\_msgs::Twist :init) r)) r))

(dolist (elem- \_twist)

(send elem- :deserialize buf ptr-) (incf ptr- (send elem- :serialization-length))

))

;; geometry\_msgs/Wrench[] \_wrench

(let (n)

(setq n (sys::peek buf ptr- :integer)) (incf ptr- 4)

(setq \_wrench (let (r) (dotimes (i n) (push (instance geometry\_msgs::Wrench :init) r)) r))

(dolist (elem- \_wrench)

(send elem- :deserialize buf ptr-) (incf ptr- (send elem- :serialization-length))

))

;;

self)

)

(setf (get gazebo\_msgs\_new::WorldState :md5sum-) "de1a9de3ab7ba97ac0e9ec01a4eb481e")

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

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

"# This is a message that holds data necessary to reconstruct a snapshot of the world

#

# = Approach to Message Passing =

# The state of the world is defined by either

# 1. Inertial Model pose, twist

# \* kinematic data - connectivity graph from Model to each Link

# \* joint angles

# \* joint velocities

# \* Applied forces - Body wrench

# \* relative transform from Body to each collision Geom

# Or

# 2. Inertial (absolute) Body pose, twist, wrench

# \* relative transform from Body to each collision Geom - constant, so not sent over wire

# \* back compute from canonical body info to get Model pose and twist.

#

# Chooing (2.) because it matches most physics engines out there

# and is simpler.

#

# = Future =

# Consider impacts on using reduced coordinates / graph (parent/child links) approach

# constraint and physics solvers.

#

# = Application =

# This message is used to do the following:

# \* reconstruct the world and objects for sensor generation

# \* stop / start simulation - need pose, twist, wrench of each body

# \* collision detection - need pose of each collision geometry. velocity/acceleration if

#

# = Assumptions =

# Assuming that each (physics) processor node locally already has

# \* collision information - Trimesh for Geoms, etc

# \* relative transforms from Body to Geom - this is assumed to be fixed, do not send oved wire

# \* inertial information - does not vary in time

# \* visual information - does not vary in time

#

Header header

string[] name

geometry\_msgs/Pose[] pose

geometry\_msgs/Twist[] twist

geometry\_msgs/Wrench[] wrench

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

MSG: std\_msgs/Header

# Standard metadata for higher-level stamped data types.

# This is generally used to communicate timestamped data

# in a particular coordinate frame.

#

# sequence ID: consecutively increasing ID

uint32 seq

#Two-integer timestamp that is expressed as:

# \* stamp.sec: seconds (stamp\_secs) since epoch (in Python the variable is called 'secs')

# \* stamp.nsec: nanoseconds since stamp\_secs (in Python the variable is called 'nsecs')

# time-handling sugar is provided by the client library

time stamp

#Frame this data is associated with

string frame\_id

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

MSG: geometry\_msgs/Pose

# A representation of pose in free space, composed of position and orientation.

Point position

Quaternion orientation

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

MSG: geometry\_msgs/Point

# This contains the position of a point in free space

float64 x

float64 y

float64 z

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

MSG: geometry\_msgs/Quaternion

# This represents an orientation in free space in quaternion form.

float64 x

float64 y

float64 z

float64 w

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

MSG: geometry\_msgs/Twist

# This expresses velocity in free space broken into its linear and angular parts.

Vector3 linear

Vector3 angular

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

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

float64 z

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

MSG: geometry\_msgs/Wrench

# This represents force in free space, separated into

# its linear and angular parts.

Vector3 force

Vector3 torque

")

(provide :gazebo\_msgs\_new/WorldState "de1a9de3ab7ba97ac0e9ec01a4eb481e")