;; Auto-generated. Do not edit!

(when (boundp 'robotnik\_msgs\_new::SafetyModuleStatus)

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

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

(shadow 'SafetyModuleStatus (find-package "ROBOTNIK\_MSGS\_NEW")))

(unless (find-package "ROBOTNIK\_MSGS\_NEW::SAFETYMODULESTATUS")

(make-package "ROBOTNIK\_MSGS\_NEW::SAFETYMODULESTATUS"))

(in-package "ROS")

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

(intern "\*SAFE\*" (find-package "ROBOTNIK\_MSGS\_NEW::SAFETYMODULESTATUS"))

(shadow '\*SAFE\* (find-package "ROBOTNIK\_MSGS\_NEW::SAFETYMODULESTATUS"))

(defconstant robotnik\_msgs\_new::SafetyModuleStatus::\*SAFE\* "safe")

(intern "\*OVERRIDABLE\*" (find-package "ROBOTNIK\_MSGS\_NEW::SAFETYMODULESTATUS"))

(shadow '\*OVERRIDABLE\* (find-package "ROBOTNIK\_MSGS\_NEW::SAFETYMODULESTATUS"))

(defconstant robotnik\_msgs\_new::SafetyModuleStatus::\*OVERRIDABLE\* "overridable")

(intern "\*EMERGENCY\*" (find-package "ROBOTNIK\_MSGS\_NEW::SAFETYMODULESTATUS"))

(shadow '\*EMERGENCY\* (find-package "ROBOTNIK\_MSGS\_NEW::SAFETYMODULESTATUS"))

(defconstant robotnik\_msgs\_new::SafetyModuleStatus::\*EMERGENCY\* "emergency")

(intern "\*LASER\_MUTE\*" (find-package "ROBOTNIK\_MSGS\_NEW::SAFETYMODULESTATUS"))

(shadow '\*LASER\_MUTE\* (find-package "ROBOTNIK\_MSGS\_NEW::SAFETYMODULESTATUS"))

(defconstant robotnik\_msgs\_new::SafetyModuleStatus::\*LASER\_MUTE\* "laser\_mute")

(defclass robotnik\_msgs\_new::SafetyModuleStatus

:super ros::object

:slots (\_safety\_mode \_charging \_emergency\_stop \_safety\_stop \_safety\_overrided \_lasers\_on\_standby \_current\_speed \_speed\_at\_safety\_stop \_lasers\_mode \_lasers\_status ))

(defmethod robotnik\_msgs\_new::SafetyModuleStatus

(:init

(&key

((:safety\_mode \_\_safety\_mode) "")

((:charging \_\_charging) nil)

((:emergency\_stop \_\_emergency\_stop) nil)

((:safety\_stop \_\_safety\_stop) nil)

((:safety\_overrided \_\_safety\_overrided) nil)

((:lasers\_on\_standby \_\_lasers\_on\_standby) nil)

((:current\_speed \_\_current\_speed) 0.0)

((:speed\_at\_safety\_stop \_\_speed\_at\_safety\_stop) 0.0)

((:lasers\_mode \_\_lasers\_mode) (instance robotnik\_msgs\_new::LaserMode :init))

((:lasers\_status \_\_lasers\_status) (let (r) (dotimes (i 0) (push (instance robotnik\_msgs\_new::LaserStatus :init) r)) r))

)

(send-super :init)

(setq \_safety\_mode (string \_\_safety\_mode))

(setq \_charging \_\_charging)

(setq \_emergency\_stop \_\_emergency\_stop)

(setq \_safety\_stop \_\_safety\_stop)

(setq \_safety\_overrided \_\_safety\_overrided)

(setq \_lasers\_on\_standby \_\_lasers\_on\_standby)

(setq \_current\_speed (float \_\_current\_speed))

(setq \_speed\_at\_safety\_stop (float \_\_speed\_at\_safety\_stop))

(setq \_lasers\_mode \_\_lasers\_mode)

(setq \_lasers\_status \_\_lasers\_status)

self)

(:safety\_mode

(&optional \_\_safety\_mode)

(if \_\_safety\_mode (setq \_safety\_mode \_\_safety\_mode)) \_safety\_mode)

(:charging

(&optional \_\_charging)

(if \_\_charging (setq \_charging \_\_charging)) \_charging)

(:emergency\_stop

(&optional \_\_emergency\_stop)

(if \_\_emergency\_stop (setq \_emergency\_stop \_\_emergency\_stop)) \_emergency\_stop)

(:safety\_stop

(&optional \_\_safety\_stop)

(if \_\_safety\_stop (setq \_safety\_stop \_\_safety\_stop)) \_safety\_stop)

(:safety\_overrided

(&optional \_\_safety\_overrided)

(if \_\_safety\_overrided (setq \_safety\_overrided \_\_safety\_overrided)) \_safety\_overrided)

(:lasers\_on\_standby

(&optional \_\_lasers\_on\_standby)

(if \_\_lasers\_on\_standby (setq \_lasers\_on\_standby \_\_lasers\_on\_standby)) \_lasers\_on\_standby)

(:current\_speed

(&optional \_\_current\_speed)

(if \_\_current\_speed (setq \_current\_speed \_\_current\_speed)) \_current\_speed)

(:speed\_at\_safety\_stop

(&optional \_\_speed\_at\_safety\_stop)

(if \_\_speed\_at\_safety\_stop (setq \_speed\_at\_safety\_stop \_\_speed\_at\_safety\_stop)) \_speed\_at\_safety\_stop)

(:lasers\_mode

(&rest \_\_lasers\_mode)

(if (keywordp (car \_\_lasers\_mode))

(send\* \_lasers\_mode \_\_lasers\_mode)

(progn

(if \_\_lasers\_mode (setq \_lasers\_mode (car \_\_lasers\_mode)))

\_lasers\_mode)))

(:lasers\_status

(&rest \_\_lasers\_status)

(if (keywordp (car \_\_lasers\_status))

(send\* \_lasers\_status \_\_lasers\_status)

(progn

(if \_\_lasers\_status (setq \_lasers\_status (car \_\_lasers\_status)))

\_lasers\_status)))

(:serialization-length

()

(+

;; string \_safety\_mode

4 (length \_safety\_mode)

;; bool \_charging

1

;; bool \_emergency\_stop

1

;; bool \_safety\_stop

1

;; bool \_safety\_overrided

1

;; bool \_lasers\_on\_standby

1

;; float64 \_current\_speed

8

;; float64 \_speed\_at\_safety\_stop

8

;; robotnik\_msgs\_new/LaserMode \_lasers\_mode

(send \_lasers\_mode :serialization-length)

;; robotnik\_msgs\_new/LaserStatus[] \_lasers\_status

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

))

(:serialize

(&optional strm)

(let ((s (if strm strm

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

;; string \_safety\_mode

(write-long (length \_safety\_mode) s) (princ \_safety\_mode s)

;; bool \_charging

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

;; bool \_emergency\_stop

(if \_emergency\_stop (write-byte -1 s) (write-byte 0 s))

;; bool \_safety\_stop

(if \_safety\_stop (write-byte -1 s) (write-byte 0 s))

;; bool \_safety\_overrided

(if \_safety\_overrided (write-byte -1 s) (write-byte 0 s))

;; bool \_lasers\_on\_standby

(if \_lasers\_on\_standby (write-byte -1 s) (write-byte 0 s))

;; float64 \_current\_speed

(sys::poke \_current\_speed (send s :buffer) (send s :count) :double) (incf (stream-count s) 8)

;; float64 \_speed\_at\_safety\_stop

(sys::poke \_speed\_at\_safety\_stop (send s :buffer) (send s :count) :double) (incf (stream-count s) 8)

;; robotnik\_msgs\_new/LaserMode \_lasers\_mode

(send \_lasers\_mode :serialize s)

;; robotnik\_msgs\_new/LaserStatus[] \_lasers\_status

(write-long (length \_lasers\_status) s)

(dolist (elem \_lasers\_status)

(send elem :serialize s)

)

;;

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

(:deserialize

(buf &optional (ptr- 0))

;; string \_safety\_mode

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

;; bool \_charging

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

;; bool \_emergency\_stop

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

;; bool \_safety\_stop

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

;; bool \_safety\_overrided

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

;; bool \_lasers\_on\_standby

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

;; float64 \_current\_speed

(setq \_current\_speed (sys::peek buf ptr- :double)) (incf ptr- 8)

;; float64 \_speed\_at\_safety\_stop

(setq \_speed\_at\_safety\_stop (sys::peek buf ptr- :double)) (incf ptr- 8)

;; robotnik\_msgs\_new/LaserMode \_lasers\_mode

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

;; robotnik\_msgs\_new/LaserStatus[] \_lasers\_status

(let (n)

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

(setq \_lasers\_status (let (r) (dotimes (i n) (push (instance robotnik\_msgs\_new::LaserStatus :init) r)) r))

(dolist (elem- \_lasers\_status)

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

))

;;

self)

)

(setf (get robotnik\_msgs\_new::SafetyModuleStatus :md5sum-) "3cc2546ca0e4e9e66c091e756e7d22d7")

(setf (get robotnik\_msgs\_new::SafetyModuleStatus :datatype-) "robotnik\_msgs\_new/SafetyModuleStatus")

(setf (get robotnik\_msgs\_new::SafetyModuleStatus :definition-)

"# robot safety mode

string SAFE=safe

string OVERRIDABLE=overridable

string EMERGENCY=emergency

string LASER\_MUTE=laser\_mute

string safety\_mode

bool charging

bool emergency\_stop # if e-stop is pressed

bool safety\_stop # if system is stopped due to safety system

bool safety\_overrided # if safety system is overrided

bool lasers\_on\_standby # if lasers don't have power

float64 current\_speed # current speed measured by safety system

float64 speed\_at\_safety\_stop # speed measured at last safety stop by safety system

robotnik\_msgs\_new/LaserMode lasers\_mode

robotnik\_msgs\_new/LaserStatus[] lasers\_status

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

MSG: robotnik\_msgs\_new/LaserMode

string STANDARD=standard

string DOCKING\_STATION=docking\_station

string CART=cart

string CART\_DOCKING\_CART=cart\_docking\_cart

string DOCKING\_CART=docking\_cart

string INVALID=invalid

string name

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

MSG: robotnik\_msgs\_new/LaserStatus

string name

bool detecting\_obstacles

bool contaminated

bool free\_warning

# one input per each warning zone.

# first areas are closer to the robot (i.e. more restrictive)

bool[] warning\_zones

")

(provide :robotnik\_msgs\_new/SafetyModuleStatus "3cc2546ca0e4e9e66c091e756e7d22d7")