1 chickens.h File Reference

API for a real-time chicken coop protection system simulation.

Macros

- #define INIT_CHICKENS 5

  Number of initial chickens in the coop.
- #define FOX_TIME 4000ULL

  Period of the fox (in ms). It hunts every FOX_TIME.
- #define EAGLE_TIME (FOX_TIME/2)

  Period of the eagle (in ms). It hunts every EAGLE_TIME.
- #define STEP_TIME (FOX_TIME/8)

  Time of a "step" (in ms). It takes STEP_TIME to use sense or sound_alarm on a given side.
- #define JITTER 100ULL

  Small variation in delays (in ms). The sense and sound_alarm functions take a time STEP_TIME - JITTER to account for external delays from the OS.

Typedefs

- typedef enum error error_t

  Value indicating an error.
- typedef enum side side_t

  Typedef for the different sides available.
- typedef enum sense_result sense_t

  Result of the sensor.
- typedef struct coop coop_t

  Place where the chickens roam free (Opaque structure).
- typedef struct sensors sensors_t

  The sensor system itself (Opaque structure).

Enumerations

- enum error {
  OK = 0 , INVALID_POSITION = 1 , NULL_PTR = 2 , MALLOC = 3 ,
  TIMER_SETTIME = 4 , TIMER_CREATE = 5 , TIMER_DELETE = 6 , MUTEX = 7 }

  Possible results and errors returned by functions.
- enum side {
  AWAY = 0 , NORTH = 1 , SOUTH = 2 , EAST = 3 ,
  ABOVE = 4 }

  The different sides of the coop. Note that there is no WEST since there is a wall on the west side.
- enum sense_result { DETECTED = 0 , NORMAL = 1 , ERROR = 2 }

  Possible results of the sensor.

Functions

- error_t init_sensors (sensors_t **sensors)
  Initialize the sensor using dynamic memory allocation.
- error_t free_sensors (sensors_t *sensors)
  Frees the dynamically allocated resources.
- error_t start_hunt (sensors_t *sensors, coop_t *coop)
  Starts the two threats (eagle and fox).
- error_t stop_hunt (sensors_t *sensors)
  Stops the two threats (eagle and fox).
- sense_t sense (sensors_t *sensors, side_t side, error_t *error)
  Senses on the given side if a threat is present.
- error_t sound_alarm (sensors_t *sensors, side_t side)
  Sounds the alarm on the given side.
- error_t init_coop (coop_t **c)
  Initializes the coop.
- error_t free_coop (coop_t *c)
  Frees the resources of the coop.
- error_t get_chickens (coop_t *c, int *chickens)
  Writes the number of chickens remaining in the pointer.
- error_t add_chicken (coop_t *c)
  Adds one chicken to the coop.

1.1 Detailed Description

API for a real-time chicken coop protection system simulation.

This header defines constants, data types, and functions used to simulate a chicken coop threatened by a fox and an eagle, and protected by a sensor/alarm system.

1.2 Macro Definition Documentation

EAGLE_TIME

```
#define EAGLE_TIME (FOX_TIME/2)
```

Period of the eagle (in ms). It hunts every EAGLE_TIME.

FOX_TIME

```
#define FOX_TIME 4000ULL
```

Period of the fox (in ms). It hunts every FOX_TIME.

INIT_CHICKENS

```
#define INIT_CHICKENS 5
```

Number of initial chickens in the coop.

JITTER

```
#define JITTER 100ULL
```

Small variation in delays (in ms). The sense and sound_alarm functions take a time STEP_TIME - JITTER to account for external delays from the OS.

STEP_TIME

```
#define STEP_TIME (FOX_TIME/8)
```

Time of a "step" (in ms). It takes STEP_TIME to use sense or sound_alarm on a given side.

1.3 Typedef Documentation

coop_t

```
typedef struct coop coop_t
```

Place where the chickens roam free (Opaque structure).

error_t

```
typedef enum error error_t
```

Value indicating an error.

See also

enum error

sense_t

```
typedef enum sense_result sense_t
```

Result of the sensor.

See also

enum sense_result

sensors_t

```
typedef struct sensors sensors_t
```

The sensor system itself (Opaque structure).

side_t

```
typedef enum side side_t
```

Typedef for the different sides available.

See also

enum side

1.4 Enumeration Type Documentation

error

```
enum error
```

Possible results and errors returned by functions.

Enumerator

OK	No error, normal result.
INVALID_POSITION	Code errors, probably an error in the code. The side given as input was invalid (e.g., out of range)
NULL_PTR	An argument was a NULL pointer.
MALLOC	System errors, could come from the OS. Malloc error, lack of memory
TIMER_SETTIME	Error when setting the time of a timer.
TIMER_CREATE	Error when creating a timer.
TIMER_DELETE	Error when deleting a timer.
MUTEX	Mutex error. Concurrency problem. (the mutex could not be created, destroyed, locked, unlocked, etc.)

sense_result

```
enum sense_result
```

Possible results of the sensor.

Enumerator

DETECTED	Threat detected.
NORMAL	Nothing detected, all normal.
ERROR	An error occurred (malloc, mutex, ...)

side

```
enum side
```

The different sides of the coop. Note that there is no WEST since there is a wall on the west side.

Enumerator

AWAY	Away from the coop, not stealing chicken.
NORTH	North of the coop.
SOUTH	South of the coop.
EAST	East of the coop.
ABOVE	Above the coop, the eagle is flying.

1.5 Function Documentation

add_chicken()

```
error_t add_chicken (
            coop_t * c)
```

Adds one chicken to the coop.

Adds one chicken to the coop if the number of chickens is lower than INIT_CHICKEN. Uses a mutex to be thread-safe.

Parameters

c	Pointer to the coop instance.

Returns

error_t Returns an error_t (OK or error code).

free_coop()

```
error_t free_coop (
            coop_t * c)
```

Frees the resources of the coop.

Parameters

c	Pointer to the coop instance.

Returns

error_t Returns an error_t (OK or error code).

free_sensors()

```
error_t free_sensors (
            sensors_t * sensors)
```

Frees the dynamically allocated resources.

Parameters

sensors	Takes a pointer to a sensor.

Returns

error_t Returns an error_t (OK or error code).

get_chickens()

```
error_t get_chickens (
            coop_t * c,
            int * chickens)
```

Writes the number of chickens remaining in the pointer.

If the pointer is NULL, does nothing and returns NULL_POINTER. Uses a mutex to be thread-safe.

Parameters

c	Pointer to the coop instance.
chickens	Pointer to an integer output parameter where the count will be stored.

Returns

error_t Returns an error_t (OK or error code).

init_coop()

```
error_t init_coop (
            coop_t ** c)
```

Initializes the coop.

Parameters

c	Pointer to a pointer of type coop_t, which will be set.

Returns

error_t Returns an error_t (OK or error code).

init_sensors()

```
error_t init_sensors (
            sensors_t ** sensors)
```

Initialize the sensor using dynamic memory allocation.

Sensor must be freed after using free_sensors.

Parameters

sensors	Takes a pointer to a pointer of type sensors_t, which will be set.

Returns

error_t Returns an error_t (OK or error code).

sense()

```
sense_t sense (
            sensors_t * sensors,
            side_t side,
            error_t * error)
```

Senses on the given side if a threat is present.

The pointer to error_t can be NULL. If it is *not NULL*, it will be set in case of error to give more information. Uses a mutex to be thread-safe. Takes STEP_TIME ms.

Parameters

sensors	Takes a pointer to a sensors_t.
side	The side to sense on.
error	Pointer to an error_t output parameter (can be NULL).

Returns

sense_t Returns a sense_t (DETECTED, NORMAL, or ERROR).

sound_alarm()

```
error_t sound_alarm (
            sensors_t * sensors,
            side_t side)
```

Sounds the alarm on the given side.

If a threat was on that side, it is chased away and won't steal a chicken until its next period. Takes a pointer to a sensors_t and a side_t. Uses a mutex to be thread-safe. Takes STEP_TIME ms.

Parameters

sensors	Takes a pointer to a sensors_t.
side	The side to sound the alarm on.

Returns

error_t Returns an error_t (OK or error code).

start_hunt()

```
error_t start_hunt (
            sensors_t * sensors,
            coop_t * coop)
```

Starts the two threats (eagle and fox).

Important: If the number of chickens reaches zero, the code calls the exit function, causing the program to stop.

Parameters

sensors	Takes a pointer to a sensors_t.
coop	Takes a pointer to a coop_t.

Returns

error_t Returns an error_t (OK or error code).

stop_hunt()

```
error_t stop_hunt (
            sensors_t * sensors)
```

Stops the two threats (eagle and fox).

Parameters

sensors	Takes a pointer to a sensors_t.

Returns

error_t Returns an error_t (OK or error code).

Index