

<b>1 chickens.h File Reference</b>	<b>1</b>
1.1 Detailed Description . . . . .	2
1.2 Macro Definition Documentation . . . . .	2
1.3 Typedef Documentation . . . . .	3
1.4 Enumeration Type Documentation . . . . .	4
1.5 Function Documentation . . . . .	5
<b>Index</b>	<b>9</b>

## 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**

<code>sensors</code>	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**

<code>c</code>	Pointer to the coop instance.
<code>chickens</code>	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**

<code>c</code>	Pointer to a pointer of type <code>coop_t</code> , 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**

<code>sensors</code>	Takes a pointer to a pointer of type <code>sensors_t</code> , 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**

<code>sensors</code>	Takes a pointer to a <code>sensors_t</code> .
<code>side</code>	The side to sense on.
<code>error</code>	Pointer to an <code>error_t</code> 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**

<code>sensors</code>	Takes a pointer to a <code>sensors_t</code> .
<code>side</code>	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**

<i>sensors</i>	Takes a pointer to a <a href="#">sensors_t</a> .
<i>coop</i>	Takes a pointer to a <a href="#">coop_t</a> .

**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**

<i>sensors</i>	Takes a pointer to a <a href="#">sensors_t</a> .
----------------	--

**Returns**

[error\\_t](#) Returns an [error\\_t](#) (OK or error code).

# Index

ABOVE  
    chickens.h, 5

add\_chicken  
    chickens.h, 5

AWAY  
    chickens.h, 5

chickens.h, 1  
    ABOVE, 5  
    add\_chicken, 5  
    AWAY, 5  
    coop\_t, 3  
    DETECTED, 4  
    EAGLE\_TIME, 2  
    EAST, 5  
    ERROR, 4  
    error, 4  
    error\_t, 3  
    FOX\_TIME, 2  
    free\_coop, 5  
    free\_sensors, 5  
    get\_chickens, 6  
    INIT\_CHICKENS, 2  
    init\_coop, 6  
    init\_sensors, 6  
    INVALID\_POSITION, 4  
    JITTER, 2  
    MALLOC, 4  
    MUTEX, 4  
    NORMAL, 4  
    NORTH, 5  
    NULL\_PTR, 4  
    OK, 4  
    sense, 7  
    sense\_result, 4  
    sense\_t, 3  
    sensors\_t, 3  
    side, 4  
    side\_t, 3  
    sound\_alarm, 7  
    SOUTH, 5  
    start\_hunt, 7  
    STEP\_TIME, 3  
    stop\_hunt, 8  
    TIMER\_CREATE, 4  
    TIMER\_DELETE, 4  
    TIMER\_SETTIME, 4

coop\_t  
    chickens.h, 3

DETECTED  
    chickens.h, 4

EAGLE\_TIME  
    chickens.h, 2

EAST

chickens.h, 5

ERROR  
    chickens.h, 4

error  
    chickens.h, 4

error\_t  
    chickens.h, 3

FOX\_TIME  
    chickens.h, 2

free\_coop  
    chickens.h, 5

free\_sensors  
    chickens.h, 5

get\_chickens  
    chickens.h, 6

INIT\_CHICKENS  
    chickens.h, 2

init\_coop  
    chickens.h, 6

init\_sensors  
    chickens.h, 6

INVALID\_POSITION  
    chickens.h, 4

JITTER  
    chickens.h, 2

MALLOC  
    chickens.h, 4

MUTEX  
    chickens.h, 4

NORMAL  
    chickens.h, 4

NORTH  
    chickens.h, 5

NULL\_PTR  
    chickens.h, 4

OK  
    chickens.h, 4

sense  
    chickens.h, 7

sense\_result  
    chickens.h, 4

sense\_t  
    chickens.h, 3

sensors\_t  
    chickens.h, 3

side  
    chickens.h, 4

side\_t  
    chickens.h, 3

sound\_alarm  
    chickens.h, [7](#)

SOUTH  
    chickens.h, [5](#)

start\_hunt  
    chickens.h, [7](#)

STEP\_TIME  
    chickens.h, [3](#)

stop\_hunt  
    chickens.h, [8](#)

TIMER\_CREATE  
    chickens.h, [4](#)

TIMER\_DELETE  
    chickens.h, [4](#)

TIMER\_SETTIME  
    chickens.h, [4](#)