AskSin library  
by [trilu@gmx.de](mailto:trilu@gmx.de)

Framework for developing own HomeMatic® network compatible devices based on the Arduino platform.

Creative commons license

http://creativecommons.org/licenses/by-nc-sa/3.0/de/

# Intend & License

Intend of this Library is to simplify the development of new devices for an existing HomeMatic® network.

All protocol related communication should be handled in a class module; device related communication (e.g. status message sending) is handled by function calls; incoming messages (e.g. Switch, Remote events) are handled as events and forwarded to user specific modules.

The library should be as compatible as possible and support all 3 types of different device types. (SWITCH, SENSOR, ACTUATOR). Power management features and also battery measurement is included, to support development of battery powered devices.

Base for this library is the CC1101 communication chip from TI and the Atmel AVR328. A user ready module is available under the label of Panstamp®.

The library is Arduino compatible and should be used only via the Arduino Framework. Never the less, all hardware related functions/macros are consolidated on one place (HAL.h and HAL.cpp). Therefor it should be easy to port the library to a new platform.

As the library is a pure hobby for me, you will get some limited support at the FHEM forum in the respective area. The library is still at beta status and will probably never leave.

AskSin library is provided under creative commons license

http://creativecommons.org/licenses/by-nc-sa/3.0/de/

You may not use the material for commercial purposes. You must give appropriate credit, provide a link to the license, and indicate if changes were made. If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

HomeMatic is a registered Trademark of eQ-3

# Index

[Intend & License 2](#_Toc403036528)

[Index 3](#_Toc403036529)

[Register definition of a client device 3](#_Toc403036530)

[AskSin library structure 3](#_Toc403036531)

[How to get started 3](#_Toc403036532)

[HomeMatic communication overview 3](#_Toc403036533)

[Configuration – Pairing 3](#_Toc403036534)

[Configuration – Peering 3](#_Toc403036535)

[Different HomeMatic message types 3](#_Toc403036536)

[HomeMatic message structure 3](#_Toc403036537)

[DEVICE\_INFO message 3](#_Toc403036538)

[CONFIG\_PEER\_ADD message 3](#_Toc403036539)

[CONFIG\_PEER\_REMOVE message 3](#_Toc403036540)

[CONFIG\_PEER\_LIST\_REQ message 3](#_Toc403036541)

[CONFIG\_PARAM\_REQ message 3](#_Toc403036542)

[CONFIG\_START message 3](#_Toc403036543)

[CONFIG\_END message 3](#_Toc403036544)

[CONFIG\_WRITE\_INDEX message 3](#_Toc403036545)

[CONFIG\_SERIAL\_REQ message 3](#_Toc403036546)

[PAIR\_SERIAL message 3](#_Toc403036547)

[CONFIG\_STATUS\_REQUEST message 3](#_Toc403036548)

[ACK message 3](#_Toc403036549)

[ACK\_STATUS message 3](#_Toc403036550)

[INFO\_SERIAL message 3](#_Toc403036551)

[INFO\_PEER\_LIST message 3](#_Toc403036552)

[INFO\_PARAM\_RESPONSE\_PAIRS message 3](#_Toc403036553)

[INFO\_PARAM\_RESPONSE\_SEQ message 3](#_Toc403036554)

[INFO\_PARAMETER\_CHANGE message 3](#_Toc403036555)

[INFO\_ACTUATOR\_STATUS message 3](#_Toc403036556)

[Missing examples 3](#_Toc403036557)

[Missing examples 3](#_Toc403036558)

## Register definition of a client device

A HomeMatic client device has at a minimum 2 channels. Channel 0 as the main device (Pair), Channel 1 to x for interaction and peer handling.

Channel 0

Channel 1

**Principle**

List 0

List 1

**Register content**

List 3 or 4

Device specific information

(e.g. pair address)

Channel specific information

(e.g. long key press time)

Peer specific information

(e.g. need burst, need AES)

List 0 and List 1 are unique per channel.

List 3 and List 4 depending on the peer address.

This would mean - as many possible peer addresses the device has, as many List 3 or List 4 the device will have. Remember this by designing your device, every List 3 or 4 will need the respective space in Eeprom.

Let‘s assume your device will be an Actuator with 3 channels;

Your List 3 for each of these channels needs 15 bytes and you want to provide 6 peers per channel.

You will need 342 bytes in Eeprom by doing following calculation:

3 Channels x 6 Peers per Channel x 15 bytes for List 3 = 270 bytes

4 bytes per peer for storing the peer address = 72 bytes

# AskSin library structure

CC1101

(CC)

Send module (SN)

Receive module

(RV)

DB module (EE)

User Channel module

User Channel module

User Channel module

Registrar (RG)

Channel and List oriented (byte)

AskSin module (AS)

Register.h

Flash

Eeprom

Bit field oriented

HAL

Config key

Led

CC1101

CPU power management

**ATmega328P and related hardware**

The library is based on a class structure. Each logical function block is within a separate class, but aligned to the main class AskSin (AS).

In mainly all subclasses you will find a class pointer to the main class.

AskSin class (AS) is responsible for the complete HomeMatic message handling. Incoming message are handled within the “recvMessage” function. Outgoing messages are generated by the respective “send\*” functions.

DB module (EE) is responsible for flash and Eeprom handling. Structure is given by declarations in Register.h. EE module handles peer and register database. Declarations in Register.h are crucial for proper device operations.

Send module (SN) handles the send buffer but also the send queue. Send queue is designed as a poll function. It analyzes messages, encodes, sends out the bytes in send buffer and waits until an ACK is given or runs into a timeout.

Receive module (RV) handles the receive buffer but does also the first analyzes of received messages. Checks if it is a valid sender, checks if messages are addressed to the own device address and checks the intent of the message. Further analyze regarding the content is then done in AS module.

CC1101 (CC) module is responsible for the communication with the transceiver. Hardware related function as SPI interface is defined in HAL.h, but communication is driven via the CC module. Power management functions are also incorporated in the CC module.

Registrar (RG) is responsible for the communication with the user modules, like Relay class or Dimmer class. User modules get registered within this class and respective list1, list3/4’s are automatically forwarded to the modules while settings had changed or a peer message was received. All respective HomeMatic messages are also forwarded to the user module.

HAL.h/.cpp holds all hardware related information, several macros/functions are important for AskSin. Some not (like dbg), but if you want to port the AskSin framework to a different hardware, I would strongly recommend to start here. While HAL.h is part of the library, all settings outlined here are valid for the different example sketches. Please do not change settings in the HAL, if there is a need to change pins or ports, do it in the user sketch file.

# How to get started

# HomeMatic communication overview

HomeMatic protocol is a bidirectional protocol with the advantage of a more stable communication due to getting ACK‘s on successful transitions.

Master

Sensor

Actuator

Actuator

Switch

Pair

Peer

There are two types of different communication.

* Master <-> Device communication as pair
* Device <-> Device communication as peer

Pair communication is for client configuration, reading status messages and some simple control of clients. Also peer setup is done via Pair communication.

Main advantage of HomeMatic protocol is the peer communication. By setting up a proper interconnection between peers, a full home automation is possible, without having a master in place.

## Configuration – Pairing

Master

Client

DEVICE\_INFO

ACK

CONFIG\_START

ACK

CONFIG\_WRITE\_INDEX

ACK

CONFIG\_END

ACK

Send device info to master

Enable Eeprom to store new settings

Send new settings

Write to Eeprom

Flow above shows a typical configuration session for a client device. By pressing the configuration button on the client device, a message containing device information's will send to the master.

By this message the master will recognize if there is a new device in the network and will send the configuration enable flag. Based on this flag, client device will enable the respective list and wait for the configuration settings.

With CONFIG\_WRITE\_INDEX the master transmits the respective settings to the write enabled list. If it is a pairing string, there will be at least the address of the master written to the client device. To store the provided settings the master will send a CONFIG\_END string to the client.

## Configuration – Peering

Master

Remote

DEVICE\_INFO

ACK

CONFIG\_START

ACK

CONFIG\_WRITE\_INDEX

ACK

CONFIG\_END

ACK

Send device info to master

Enable Eeprom to store new settings

Send new settings

Write to Eeprom

Actuator

to be reworked

# Different HomeMatic message types

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte 03** | **Byte 10** | **Byte 11** | **Description** |
| **00** |  |  | DEVICE\_INFO |
| **01** |  | 01 | CONFIG\_PEER\_ADD |
| **01** |  | 02 | CONFIG\_PEER\_REMOVE |
| **01** |  | 03 | CONFIG\_PEER\_LIST\_REQ |
| **01** |  | 04 | CONFIG\_PARAM\_REQ |
| **01** |  | 05 | CONFIG\_START |
| **01** |  | 06 | CONFIG\_END |
| **01** |  | 08 | CONFIG\_WRITE\_INDEX |
| **01** |  | 09 | CONFIG\_SERIAL\_REQ |
| **01** |  | 0A | PAIR\_SERIAL |
| **01** |  | 0E | CONFIG\_STATUS\_REQUEST |
| **02** | 00 |  | ACK |
| **02** | 01 |  | ACK\_STATUS |
| **02** | 02 |  | ACK2 |
| **02** | 04 |  | ACK-PROC (AES-CHALLANGE) |
| **02** | 80 |  | NACK |
| **02** | 84 |  | NACK\_TARGET\_INVALID |
| **02** |  |  | ACK\_NACK\_UNKNOWN |
| **02** |  |  | REQUEST\_AES |
| **03** |  |  | AES\_RESPONSE |
| **04** | 00 |  | TO-HMLAN\_SEND\_AES\_CODE |
| **04** |  |  | TO-ACTOR\_SEND\_AES\_CODE |
| **10** | 00 |  | INFO\_SERIAL |
| **10** | 01 |  | INFO\_PEER\_LIST |
| **10** | 02 |  | INFO\_PARAM\_RESPONSE\_PAIRS |
| **10** | 03 |  | INFO\_PARAM\_RESPONSE\_SEQ |
| **10** | 04 |  | INFO\_PARAMETER\_CHANGE |
| **10** | 06 |  | INFO\_ACTUATOR\_STATUS |
| **11** | 02 |  | SET |
| **11** | 03 |  | STOP\_CHANGE |
| **11** | 04 | 00 | RESET |
| **11** | 80 |  | LED |
| **11** | 81 | 00 | LEDALL |
| **11** | 81 |  | LEVEL |
| **11** | 82 |  | SLEEPMODE |
| **11** | CA |  | ENTER\_BOOTLOADER |
| **12** |  |  | HAVE\_DATA |
| **3E** |  |  | SWITCH |
| **3F** |  |  | TIMESTAMP |
| **40** |  |  | REMOTE |
| **41** |  |  | SENSOR\_EVENT / CONDITIONAL\_SWITCH\_COMMAND |
| **53** |  |  | SENSOR\_DATA |
| **58** |  |  | CLIMATE\_EVENT |
| **5E** |  |  | POWER\_EVENT\_CYCLIC |
| **5F** |  |  | POWER\_EVENT |
| **70** |  |  | WEATHER\_EVENT |

## HomeMatic message structure

**Body - used for all message types**

01 02 03 04 05 06 07

1A 00 A2 00 3F A6 5C 00 00 00 10 80 02 50 53 30 30 30 30 30 30 30 31 9F 04 01 01

1. **Message length indicator**

Reflect the size of the message without the indicator

**02 Message counter**

One Byte counter, increased by every message sent. When sending an ACK the message counter from the received message has to be used.

**03 Communication bit field**

80 Message to master or message could be repeated

40 Repeated message

20 BiDi (ACK requested)

10 Burst

8 unknown

4 Broadcast

2 Wake me up

1 Wakeup

**04 Message type**

See table on next page

**05 Source Address**

3 Byte length, identification of own device

**06 Target Address**

3 Byte length, 00 00 00 means broadcast

**07 Payload; max size ~20 bytes**

## DEVICE\_INFO message

**01 02 03 04 05**

1A 00 A2 00 3F A6 5C 00 00 00 **10 80 02 50 53 30 30 30 30 30 30 30 31 9F 04 01 01**

**Payload**

1. Firmware version

1 Byte, seems to be not important

1. Model id

2 Bytes, as HM starts with 0, own devices should start with FF

1. Serial number

Byte ASCII, must be unique

1. Sub type id

1 Byte, not needed for FHEM, it's something like a group ID

1. Device info  
   3 Byte, describes device, includes amount of channels

**Description**

Device info message will be send from slave to master after pressing configuration button of device. This message starts typical the pairing sequence. Message could be the answer to a PAIR\_SERIAL request from master.

## CONFIG\_PEER\_ADD message

**01 02 03 04**

10 55 A0 01 63 19 63 1E 7A AD **03** 01 **1F A6 5C 06 05**

**Payload**

1. Channel

1 Byte, channel on which the peer should be added

1. Peer id

3 Bytes, device address of the peer

1. Peer channel A

1 Byte, first channel of peer device

1. Peer channel B

1 Byte, second channel of peer device, if not needed then a 0 is send

**Description**

Config peer add message is send from master to client. Intent is to make a peer connection between two client devices, e.g. a remote and an actor.

If two buttons are set, one for on, the other for off, then peer channel A and b is filled with the respective remote button channels. If you have configured a toggle button, peer channel b is filled with a 00. This means, no need for storing the second peer id in the peer database.

Valid answer is an ACK.

## CONFIG\_PEER\_REMOVE message

**01 02 03 04**

10 55 A0 01 63 19 63 1E 7A AD **03** 02 **1F A6 5C 06 05**

**Payload**

1. Channel  
   1 Byte, where the peer should be added
2. Peer id  
   3 Bytes, device address of the peer
3. Peer channel A  
   1 Byte, first channel of peer device
4. Peer channel B  
   1 Byte, second channel of peer device, if not needed then a 0 is send

**Description**

Config peer remove message is send from master to client. Intent is to remove a peer connection between two client devices.

Valid answer is ACK.

## CONFIG\_PEER\_LIST\_REQ message

**01**

0B 05 A0 01 63 19 63 1E 7A AD **01** 03

**Payload**

1. Channel  
   1 Byte, from which the peer list is requested

**Description**

Config peer list request message will be send from master to slave to get the complete peer list of the respective channel.

Valid answer is an INFO\_PEER\_LIST message.

## CONFIG\_PARAM\_REQ message

**01 02 03 04**

10 04 A0 01 63 19 63 1E 7A AD **01** 04 **00 00 00 00 01**

**Payload**

1. Channel  
   1 Byte, from which the peer list is requested
2. Peer id  
   3 Byte, peer id of the requested list. If a List0/1 is requested   
   peer id is 00 00 00
3. Peer channel  
   1 Byte, indicates the respective channel of peer device
4. List  
   1 Byte, select the number of the list, e.g. List0, List1, List3…

**Description**

Config param request message will be send from master to slave to get the respective List for the required channel and peer.

Valid answer:  
INFO\_PARAM\_RESPONSE\_PAIRS

INFO\_PARAM\_RESPONSE\_SEQ

## CONFIG\_START message

**01 02 03 04**

10 01 A0 01 63 19 63 1E 7A AD **00** 05 **00 00 00 00 00**

**Payload**

1. Channel  
   1 Byte, from which the peer list is requested
2. Peer id  
   3 Byte, peer id of the requested list. If a List0/1 is requested   
   peer id is 00 00 00
3. Peer channel  
   1 Byte, indicates the respective channel of peer device
4. List  
   1 Byte, select the number of the list, e.g. List0, List1, List3…

**Description**

Config start message will be send from master to slave to open the respective List for write access. Next message has to be a CONFIG\_WRITE\_INDEX message with the fitting channel.

For List0 and List1 peer id has to be 00 00 00.

Valid answer is an ACK

## CONFIG\_END message

**01**

0B 01 A0 01 63 19 63 1E 7A AD **00** 06

**Payload**

1. Channel  
   1 Byte, from which the peer list is requested

**Description**

Config end message will be send from master to slave to close the respective channel for write access.

For List0 and List1 peer id has to be 00 00 00.

Valid answer is an ACK.

## CONFIG\_WRITE\_INDEX message

**01 02 03**

13 02 A0 01 63 19 63 1E 7A AD **00** 08 **02 01 0A 63 0B 19 0C 63**

**Payload**

1. Channel  
   1 Byte, from which the peer list is requested
2. Length of data bytes
3. Data  
   Register address: byte value, register address: byte value, etc

**Description**

Config write index message will be send from master to slave to update a list which was opened by CONFIG\_START message.

Valid answer is an ACK

## CONFIG\_SERIAL\_REQ message

0B 48 A0 01 63 19 63 1E 7A AD 00 09

**Payload**

No payload

**Description**

Config serial request message will be send from master to slave to get the serial number of client device.

Valid answer is an INFO\_SERIAL message

## PAIR\_SERIAL message

**01**

15 48 A0 01 63 19 63 1E 7A AD 00 0A **4A 45 51 30 37 33 31 39 30 35**

**Payload**

1. Serial number  
   10 Byte, serial number of client device

**Description**

Pair serial message will be send from master to slave to start the pairing sequence from the master. Target address and serial has to fit to the client device.

Valid answer is a DEVICE\_INFO message.

## CONFIG\_STATUS\_REQUEST message

**01**

0B 30 A0 01 63 19 63 2F B7 4A **01** 0E

**Payload**

1. Channel  
   1 Byte, channel of the client device

**Description**

Config status request message will be send from master to slave to get the current status of the respective channel from the client device.

Valid answer is an INFO\_ACTUATOR\_STATUS message.

## ACK message

0A 04 80 02 1F A6 5C 63 19 63 00

**Payload**

No payload

**Description**

ACK is the standard answer to most messages which are not requesting some information. An ACK is requested in communication by the 3 byte, bit 6 of the initial message. See „HomeMatic message structure“

Standard timeout is 300ms for most HM devices.

Message counter byte (byte 2) has to be the same, as the initial message was.

## ACK\_STATUS message

**01 02 03 04**

0E 12 80 02 1F B7 4A 63 19 63 01 **01 00 00 3B**

**Payload**

1. Channel  
   1 Byte, reflects the channel the message is send from
2. Status  
   1 Bytes, most HM devices are using 0 for off and 200 for on. If status is provided in %, a multiplier of 2 is used
3. Combined  
   1 Byte, ‘UP’ 0x10, ‘DOWN’ 0x20, LOWBAT ‘0x80’
4. RSSI  
   1 Byte, indicates signal strength of master or peer device, depending on addressed device

**Description**

ACK\_STATUS is an enhanced version of ACK which contains some status information of client device.

ACK\_STATUS is send as answer to (no completely sure)

|  |  |  |  |
| --- | --- | --- | --- |
| **Byte 03** | **Byte 10** | **Byte 11** | **Description** |
| **11** | 02 |  | SET |
| **3E** |  |  | SWITCH |
| **40** |  |  | REMOTE |
| **41** |  |  | SENSOR\_EVENT |

## INFO\_SERIAL message

**01**

14 48 80 10 1E 7A AD 63 19 63 00 **4A 45 51 30 37 33 31 39 30 35**

**Payload**

1. Serial number  
   10 Byte, serial number of device, has to be unique

**Description**

Info serial message is the appropriate answer from a client to master for a CONFIG\_SERIAL\_REQ.

## INFO\_PEER\_LIST message

**01 02 03 04 05**

1A 05 A0 10 1E 7A AD 63 19 63 **01 1F A6 5C 02 1F A6 5C 01 11 22 33 02 11 22 33 01**

**Payload**

1. Channel  
   1 Byte, reflects the channel the peer-list comes from
2. Peer 1  
   4 Byte, first peer in peer database
3. Peer 2  
   4 Byte, second peer in peer database
4. Peer 3  
   4 Byte, third peer in peer database
5. Peer 4  
   4 Byte, fourth peer in peer database

**Description**

Info peer list message is the answer from slave device to master while received a CONFIG\_PEER\_LIST\_REQ. Payload of this message is 16 bytes. Message has to be terminated with 00 00 00 00.

If more than 3 peers are to send, the message is send multiple time with different payload reflecting the whole peer database of the respective channel.

Valid answer from master is an ACK

## INFO\_PARAM\_RESPONSE\_PAIRS message

**01**

1A 31 A0 10 1E 7A AD 63 19 63 02 **30 06 32 50 34 4B 35 50 56 00 57 24 58 01 59 00**

**Payload**

1. Data  
   max 16 Byte, contains register address and content of respective List

**Description**

Info param response pairs is the answer from a client to master as answer to a CONFIG\_PARAM\_REQ. This is again a multi string transmission which has to be terminated by 00 00 at the end.

Valid answer from master is an ACK.

// config param req

10 31 A0 01 63 19 63 1E 7A AD 02 04 00 00 00 00 01

// info param response pairs  
1A 31 A0 10 1E 7A AD 63 19 63 02 30 06 32 50 34 4B 35 50 56 00 57 24 58 01 59 00

// ACK from master  
0A 31 80 02 63 19 63 1E 7A AD 00

// info param response pairs  
0C 32 A0 10 1E 7A AD 63 19 63 02 00 00

// ACK from master

0A 32 80 02 63 19 63 1E 7A AD 00

## INFO\_PARAM\_RESPONSE\_SEQ message

**01 02 03 04 05**

1A 00 A2 00 3F A6 5C 00 00 00 **10 80 02 50 53 30 30 30 30 30 30 30 31 9F 04 01 01**

**Payload**

1. Firmware version

1 Byte, seems to be not important

1. Model id

2 Bytes, as HM starts with 0, own devices should start with FF

To be updated

1. Serial number

Byte ASCII, must be unique

1. Sub type id

1 Byte, not needed for FHEM, it's something like a group ID

1. Device info  
   3 Byte, describes device, includes amount of channels

**Description**

Device info message will be send from slave to master after pressing configuration button of device. This message starts typical the pairing sequence. Message could be the answer to a PAIR\_SERIAL request from master.

## INFO\_PARAMETER\_CHANGE message

**Payload**

**Description**

## INFO\_ACTUATOR\_STATUS message

**01 02 03 04**

06 12 A0 10 3F A6 5C 63 19 63 06 **01 00 00 3B**

**Payload**

1. Channel  
   1 Byte, reflects the channel the message is send from
2. Status  
   1 Byte, most HM devices are using 0 for off and 200 for on. If status  
   is provided in %, a multiplier of 2 is used
3. Flag  
   1 Byte, flags to be specified
4. RSSI  
   1 Byte, indicates signal strength of master or peer device, depending  
   on addressed device

**Description**

Info actuator status message is the answer from a client to master on a CONFIG\_STATUS\_REQUEST.

Valid answer from master to client is an ACK.

## POWER\_EVENT, POWER\_EVENT\_ CYCLIC message

**01 02 03 04 05 06 07 08 09 0A 0B**

14 12 A0 5E 3F A6 5C 63 19 83 17 BD 00 17 85 01 82 09 38 FF POWER\_EVENT

14 12 A0 5F 3F A6 5C 63 19 83 17 BD 00 17 85 01 82 09 38 FF POWER\_EVENT\_CYCLIC

**Payload**

1. Boot, Energycounter
2. ???
3. ???
4. Power byte 1
5. Power byte 2
6. Power byte 3
7. Current Byte 1
8. Current Byte 2
9. Voltage Byte 1
10. Voltage Byte 2
11. Frequency Byte  
    Signed byte (Frequence difference to 50 hz)

**Description**

Info actuator status message is the answer from a client to master on a CONFIG\_STATUS\_REQUEST.

Valid answer from master to client is an ACK.

## Missing examples

|  |
| --- |
| ACK2 |
| ACK-PROC |
| NACK |
| NACK\_TARGET\_INVALID |
| ACK\_NACK\_UNKNOWN |
| REQUEST\_AES |
| AES\_REPLY |
| TO-HMLAN\_SEND\_AES\_CODE |
| TO-ACTOR\_SEND\_AES\_CODE |
| ENTER\_BOOTLOADER |

Nachrichten die per AES-Signierung gesichert sind:

INFO\_ACTUATOR\_STATUS

SET

CONFIG\_START

CONFIG\_WRITE\_INDEX

CONFIG\_END

CONFIG\_PEER\_ADD

REMOTE

Nachrichten die per AES-Signierung NICHT gesichert sind:

CONFIG\_PARAM\_REQ

INFO\_PARAM\_RESPONSE\_SEQ