Door Control

The Solar System can operate doors directly, using Max readers or other devices, or ESP8266 based WiFi devices.

However, the ESP8266 modules can also operate autonomously at various levels of integration, including entirely self contained.

The door control logic works over 6 different levels:

- Simply acting as NFC reader and inputs/outputs for the alarm system
- Managing a door *state* and operations such as lock, unlock, but not taking autonomous action to open doors. Also manages locking when closed. This is recommended rather than simply being inputs/outputs. This also tracks a number of error cases such as door being forced.
- Also managing door exit operation (i.e. exit button or ranger unlocking the door)
- Also managing door entry by use of any validated card/fob (using AES handshake). This only makes sense where any card is allowed to access all doors at all times.
- Managing entry using a validated card/fob and access control file on the card/fob which defines times of day by day of week, and allowed doors, and expiry.
- Also managing setting and unsetting “alarm” (door deadlock mode) which can causes an output to signal to alarm system, etc, that alarm should be set or not.

These various levels work well if the door control is mostly on-line to the alarm system. However, in difficult WiFi scenarios the controllers can *fall off* the WiFi for many seconds at a time, which is not a good user experience. By making all normal working autonomous the response is much better. However this means that cards have to be set up, and, if necessary, updated if permissions change. It also required proper use of blacklist settings and expiry controls to ensure lots cards to not have access.

NXP MIFARE DESFire NFC

WiFi devices with a PN532 reader attached can make use of high security access cards, the NXP MIFARE DESFire cards. This can work over WiFi or can even operate as a fake Max Reader sending a fob ID for NFC cards.

These need to be configured to work with the system. A top level AID for the system if defined automatically (random) as is a top level 128 bit AES key. These can be set manually, and can also be set per device, for example, if some doors warrant extra security. Each device looks for one AID only and uses one AES key. Note the AID is in transmission order (low yet first).

The card should allow the AID to be selected and authentication of key `0x01` with the specified AES key (i.e. application needs at least 2 keys). The files below need read access, and the log and counter files need write access too. The comms settings should be plain (for now, this may change later).

The following fields are defined - these are all optional, and are only used if present.

Not used in app	
`00`	User's name, this should be printable UTF-8 text. E.g. `Fred Bloggs`

Used for logging	
`01`	Log, a cyclic record file with record size of 10. Records hold 3 byte device chip-id followed by BCD coded YYYYMMDDHHMMSS UTC access time. Needs to be writable (allow credit 1).
`02`	Usage counter
`0A`	Fixed size access file - a backup file of 256 bytes with first byte saying how many bytes follow in the variable length access file (see below).

A command is included in the linux build (`cardissue`) which allows blank MIFARE DESFire EV1 cards to be configured, and updated, as needed to be used with the system.

This command also advises what fob ID would be seen by a Galaxy system using an RS485 bus to access the door controller as this is derived from the NFC card ID.

Access file

The access file is a single file that contains access control data. This would more logically be a series of files for different purposes, but by making a single file it reduces the number of card accesses. The file should exists in any fully autonomous control cases.

Logically the file is variable length, however DESFire cards do not lend themselves to such files. To read the file means checking the size first, and changing the file would mean deleting and re-creating the file, which uses up EEPROM. To address this the file format is a 256 byte *backup* file which has as the first byte a length of following data. This allows an initial block, say 16 bytes, to be read, and read more read only if needed, without first checking file size. It also allows the content to be overwritten as needed without wasting EEPROM.

The file then contains a number of tagged fields, each has one byte with top 4 bits as type and bottom 4 bits as length of following data. Typically either `0X` or `1X` are used, not both.

Access file tagged data	
A0	**Allow zero**: Card blocked, stop further checking and do not allow access.
AX	**Allow**: One or more 3 byte device IDs for doors that are allowed. If any 1X field exists the device has to be found in one of them to be allowed.
BX	**Bar**: One or more 3 byte device IDs for doors that are barred. If the device is found, then checking stops and access is not allowed.
FX	**From**: Times of day BCD coded as HHMM. This is the start time allowed. See below
2X	**To**: Times of day BCD coded as HHMM. This is the end time allowed. See below.
E1	**Expiry**: Folled by one byte number of days (1-255) to update expiry date when access allowed.
EX	**Expires**: BCD coded YYYYMMDDHHMMSS expiry, or truncated part thereof. Expiry is end of specific time, e.g. 201906 means end of Jun 2019.
0X	**Zero**: Padding, ignore.

Time ranges

From and to times are coded as pairs of bytes BCD coding HHMM.

- One time means a time for every day
- Two times means a time for Weekend, and a time for Weekdays
- Three times means a time for Sunday, Weekdays, and Saturday
- Seven times means time for each day Sunday to Saturday

The *from* time is from the beginning of the specified HHMM. The *to* time is up to the moment before the specified HHMM. A value of 2400 is valid. It is valid for only a *from* time time (so valid to end of day), or only a *to* time (so valid from start of day).

If *to* is before *from* then access is valid on, or after, the *from* time **or** before the *to* time, otherwise access is only valid on, or after, the *from* time **and** before the *to* time.

Using secure and insecure cards.

The system can use the following types of *fob ID* for identifying users.

- Max reader (125kHz proxy cards) - a simple numeric ID. This is considered insecure as it is easy to copy and fake (and the number is even printed on the fob).
- NFC card ID. This is a 4 or 7 byte value, so 8 or 14 hex characters. This is also considered insecure as it can be copied or faked
- MIFARE DESFire cards with a 4 or 7 byte ID, so 8 or 14 hex characters, having been validated using AES.

If any device is listed with `nfc` set then an `aid` and `aes` random code is created in the config automatically. If an `aid/aes` exists then the system assumes all IDs should be secure. An insecure ID will not have access unless the `insecure-` settings are defined for the user. In a system without `aid/aes` defined, all cards are assumed to be insecure, and the `insecure-` settings are not needed.

Using bank cards and other NFC cards

NFC cards have an ID, normally 4 or 7 bytes. This includes:-

- Bank cards
- MIFARE classic cards
- MIFARE DESFire cards
- Various fobs and devices
- Mobile phones
- Passports

Most of these provide an ID which is fixed and sufficiently unique to be used as an access control card. This is not true for all - e.g. phones and passports produce a random ID so cannot be used. Some cards, like Amex, use ID `00000000` which is not useful. MIFARE cards can be set to use a random ID. However, most bank cards can be used. These are, however, all considered to be insecure except the MIFARE DESFire when validated to AES encryption. Obviously you need to consider when convenience is more important and allow use of insecure cards (e.g. opening some doors, but not disarming alarms, etc).

MQTT messages relating to NFC cards

The AID and AES codes are sent (over MQTT over TLS) to the card as binary using a `command`, not a `setting`. They are held in memory and not written to flash/EEPROM, so have to be sent whenever the device is seen re-connecting to MQTT.

The reporting from the device for NFC cards is by use of an `event` of type `id`, or, for autonomous door control mode: `access` or `noaccess`. This is followed by the 8 or 14 character hex ID. If a card is held for a pre-defined period then an `event` of `held` is reported and then an `event` of `gone` is reported when the card is removed - these have the same card ID as the data. This applies to any NFC cards seen, and should be assumed to be an insecure ID. For `noaccess` the id is followed by a space and a text reason for no access.

If the card allows the AID to be selected then it is assumed to be a secure card, and is authenticated with AES.. The cards ID is then retrieved and validated. If this is successful then the ID reports is 14 bytes of HEX with a `+` suffix to indicate it is a secure ID being used. The log and usage count are updated after sending the ID (this can be changed to “before” using `nfccommit` setting if required).

An `error` message is sent if there is any problem - this can be after sending the ID if an issue with updating log, etc. If the card does not get authenticated (e.g. removed too quickly) its ID can be sent as insecure, so if some insecure operations are allowed this can allow faster working.

Autonomous Door Control

The SolarSystem normally provides full door control sequencing and monitoring, i.e. it knows to engage the lock when the door closes, and spot a door AJAR state if the lock engaged input does not indicate the lock has in fact engaged in the appropriate time, etc.

The door can be in one of a set of states, which depend on the state of inputs and outputs and timers. Typically each state also has an LED sequence shown on the NFC reader.

However the ESP code can also provide this door control, in which case the main software simply needs to send a message when needed to, for example, open the door. It leaves the ESP to sequence and time disengaging the lock, and waiting for the door open, and wait for door closed, and re-engage the lock, and wait for lock engaged input, etc. The ESP reports the appropriate states to the control system.

The main ways of working, set by the "`door`" setting, are :-

`door`	
0	If not set, no monitoring or control of door inputs/outputs
1	Lowest level state tracking, no automatic operation of exit button or access cards. It is expected that the control software tells it when to open doors.
2	Tracking and control, with exit button operation to allow the door to open (if not in `DEADLOCKED`) state. This allows fast response for exit buttons, but ensures security by checking all cards/fobs with the control system.
3	Tracking and control, exit button, and allow any secure card to open the door (again, this is not done if in `DEADLOCKED`) state. This provides fast response for cards as well, but means the control system has to step in to re-lock a door if an invalid card is used. This makes sense for some internal low security doors.
4	Fully autonomous control using fields on the MIFARE card to control access times, etc.
5	As 4, but also allows deadlock (hold fob) and un-deadlock.

Inputs / outputs

The operation replies on having `input` and `output` settings defined. It is valid for some of these not to be defined. Obviously some of these need to be configured to invert the operation to match the way of working that is expected.

input / output		
`input1`	`i-exit`	Exit button (1=exit pressed, 0=exit release)
`input2`	`i-open`	Door open (1=open, 0=closed)
`input3`	`i-unlock`	Lock not engaged (1=not engaged, 0=engaged)
`input4`	`i-undeadlock`	Deadlock not engaged (1=not engaged, 0=engaged)
`input8`	`i-exit`	Secondary exit button (1=exit pressed, 0=exit release)
`output1`	`o-unlock`	Unlock (1=unlock, 0=lock)
`output2`	`o-undeadlock`	Deadlock unlock (1=unlock, 0=lock)
`output3`	`o-beep`	Beeper

input / output		
output4	o-error	Active if door is in a fault or tamper state

Lock states

The system allows for two locks, a *main* and *deadlock*. The main lock is disengaged and engaged whenever the door is used. The deadlock is for when the alarm is set. The deadlock may not exist, or could simply be an output to tell an alarm system to be set, for example.

Each look has a state which is tracked. The state is based on observing two signals, an *unlock* output, and an *unlocked* input, which exist for each lock. The output controls the actual lock, and the input indicates the lock is engaged or not. Remember that inputs and outputs can be inverted so as to operate as *unlock* and *unlocked* as expected here.

- If there is not input or output defined, then we stat in the NOLOCK state.
- If there is only an input defined, we set the state to LOCKED or UNLOCKED based on the input.
- If the output state changes, then this indicates we are trying to lock or unlock. As such a timer is started and the state set to LOCKING or UNLOCKING accordingly. Note that the timer does not run when LOCKING (i.e. is continually restarted) if the door is open as it is assumed the lock cannot engage in that state.
- If the input state changes whilst the timer is running, and changes to a state that matches the output state, then the timer ends early.
- If the timer ends (including as above): If the input is not defined, or it matches the output we change state to LOCKED or UNLOCKED accordingly. Otherwise we change state to LOCKFAIL or UNLOCKFAIL accordingly.
- If the input state changes and the timer is not running, then we change state to FAULT or FORCED accordingly if the input does not match the output, otherwise LOCKED or UNLOCKED.

The lock state is then used to determine the door state, and state of faults and tampers.

Lock states	
NOLOCK	There is no input or output defined
LOCKED	Locked
UNLOCKED	Unlocked
LOCKING	Locking, the lock timer is running
UNLOCKING	Unlocking, the unlock timer is running
LOCKFAIL	The lock did not engage at the end of the lock timer
UNLOCKFAIL	The lock did not disengage at the end of the lock timer
FAULT	The lock changed to a locked state when we did not ask it to
FORCED	The lock changed to an unlocked state when we did not ask it to

Door states

The door can be in one of the following normal states, reported over MQTT.

Normal door states	
DEADLOCKED	This is the normal state when the alarm is set, it means the door is closed and the main and deadlock are engaged. i.e. alarm set. In this mode the system will not autonomously allow exit or secure cards to open the door unless mode 4 fully autonomous.
LOCKED	This is the normal, idle, door closed state where the door is closed and the main lock is engaged, but the deadlock is not engaged. i.e. alarm not set.
UNLOCKING	This is a state after LOCKED, or DEADLOCKED, where the lock or locks are disengaged, and we are waiting for the lock engaged input(s) to confirm the lock is no longer engaged.
UNLOCKED	This is a state after UNLOCKING where the lock is disengaged, and confirmed as such, but the door is not yet open. A timeout runs, at the end of which we engage the lock(s).
OPEN	This is a state where the door is open, and should be. A timeout runs, at the end of which we change to NOTCLOSED state.
PROPPED	This is a state where the door is open, and has been authorised to be propped open.
CLOSED	This is a state the same as unlocked after the door was open. A timeout runs, at the end of which we engage lock(s).
LOCKING	This is a state where the door is now closed, and the lock(s) are engaging. This also applies when changing from LOCKED to DEADLOCKED.

There are also exceptional states which come in to play after a timeout.

Exceptional door states	
NOTCLOSED	This is a state where the door is open, and has been for too long without authorisation. This follows from OPEN state. This can be changed to PROPPED.
AJAR	The door is closed, but the lock has not engaged, suggesting it is not fully closed and could be opened.

These states are based on a *door open* input, and the lock states.

If the door is open: The door can be in OPEN, NOTCLOSED, or PROPPED state.

- If not already in one of those states, the state changes to OPEN and a timer it set. If setting OPEN and any lock is in LOCKING state, we unlock that lock. i.e. opening whilst locking is allowed.
- If the timer expires in OPEN state the state changes to NOTCLOSED.
- If OPEN or NOTCLOSED and the door prop is authorised then the state changes to PROPPED.

If the door is closed: We test door states until one matches.

State	Main lock	Deadlock	Meaning
DEADLOCKED	NOLOCK LOCKED	LOCKED	The door is deadlocked.
LOCKED	LOCKED	NOLOCK UNLOCKED	The door is locked
UNLOCKING	UNLOCKING	-	The door is unlocking

State	Main lock	Deadlock	Meaning
	-	UNLOCKING	
LOCKING	LOCKING	-	The door is locking
	-	LOCKING	
AJAR	LOCKFAIL	-	The door has failed to lock, so probably is not closed properly.
	-	LOCKFAIL	
UNLOCKED	Otherwise		The door is unlocked, ready to be open. This state is if previous state was not OPEN, NOTCLOSED, or PROPPED.
CLOSED			The door is unlocked, having been closed.

Fault states

- Either lock is in a UNLOCKFAIL, or FAULT state.
- Exit or ranger input are stuck active

Tamper states

- Either lock is in FORCED state.
- Either lock is in LOCKED state and the door is open, i.e. door forced.

Settings

In addition to the main door setting, there are some timings that can be set.

Settings	
doorunlock	Time (ms) from lock disengage to lock engaged input confirming disengaged. This is used to indicate there is a problem unlocking (fault condition) if the lock engaged input is defined, or simply used as the time to allow in UNLOCKING state if not.
doorlock	Time (ms) from lock engaged to lock engaged input confirming engaged. This is used to indicate there is a problem locking (AJAR state) if the lock engaged input is defined, or simply used as the time to allow in LOCKING state if not.
doorprop	Time (ms) to allow door open before considered being NOTCLOSED state.
dooropen	Time (ms) to allow door to remain in UNLOCKED state after UNLOCKING state before door is locked as not having been opened.
doorclose	Time (ms) to allow door to remain in CLOSED state after closing the door before it is locked.
doorexit	Time (ms) to allow exit button press, beyond which its is assumed to be stuck somehow.
doorcycle	Time (ms) to cycle lock and beep in event of error (AJAR, NOTCLOSED, or forced)
doorbeep	If set to 1, then beep whilst waiting for door to open

Commands

The inputs report state changes as normal, and the door state changes are reported as a state message, but there are commands which can be sent to impact the door state:-

Commands	
lock	This aims to change state to LOCKED. If the door is CLOSED or UNLOCKED, it immediately changes to LOCKING state. This can also be used in DEADLOCKED state to move out of DEADLOCKED state, i.e. alarm unset.
deadlock	This aims to change state to DEADLOCKED. If the door is CLOSED or UNLOCKED it immediately changes to LOCKING state. This can be used in LOCKED state to move to DEADLOCKED state (i.e. alarm set). Deadlocking means the exit button and cards do not work autonomously unless in door mode 4 (fully autonomous). A flag is set so that once closed, both locks will engaged to moved to DEADLOCKED state.
unlock	This is the same as working the exit button, but works in DEADLOCKED state too. It disengages locks moving to UNLOCKING and CLOSED state to allow the door to be opened.
prop	If the door is in OPEN or NOTCLOSED state it changes to PROPPED state.
acess	This does nothing, just confirms an access. Used two send new access file content.

In effect there is a flag as well as the main door states, the *deadlock* flag is if deadlock command has been sent, this is cleared if any other command is sent. This flag means that when locking, both locks are engaged rather than just the main lock, leading to a DEADLOCKED rather than just LOCKED state.

These commands normally have no data part sent. If data is sent, it is new content of the 0A access file, starting with the length byte and followed by that many bytes. If the card is still connected, and the contents of the access file is different, then the reader attempts to write this to the card as the new access file. This would normally only make sense for a lock or access sent after reading a card.

Exit

The door control also tracks the exit button or ranger input and triggers unlock as appropriate.