Konzept ESP32-sensact

* 125kbits
* ExtendedID mit 29bits, 5bit Namespace, 24bit namespace-spezifisch
  + Namespace 0b00000: Events. 16bit EmitterID, 8 bit EventType
* Nur Events, keine Commands. Prinzip: Alle dürfen in den CAN-Bus was hineinrufen, aber keiner darf dem anderen vorschreiben, was er zu tun hat
  + Problem: Ohne die ESP32 umzuprogrammieren, ist dann auch kein verändertes Verhalten PC-seitig einfach zu implementieren
  + EventID besteht aus EmitterID (16bit, wer sendet es weg) und EventType (8bit, was ist dort passiert).
  + Ein Event kann außerdem noch bis zu 8 Byte Nutzdaten transportieren
  + EmitterID=0 bedeutet: Event kommt von der Management-Ebene
  + EventType=0 bedeutet: Unspezifiziertes Event vom Sender
  + EventType 0xFF bedeuet: Dem Empfänger ist es egal, welches konkrete Event gesendet wird. Der Empfänger reagiert damit auf alle Events gleichermaßen
* Event Emitter sind
  + „ButtonSensor“ mit Press, Hold (wenn sicher ist, dass es kein ShortPress ist), ReleaseShort (auch wenn danach ein DoubleClick kommt), ReleaseLong, DoubleClick (DoubleClick hat zuvor einen ReleaseShort, aber dann kein Press und ReleaseShort mehr) (Prinzip: Events werden zum frühest möglichen Zeitpunkt gesendet)
  + RotarySensor mit TurnLeft, TurnRight, Press
  + PWMLightingActor mit OnTriggerHeartbeat
* BlindsActor kennen eine Liste ihrer „Up“-ButtonSensors und ihrer „Down“-ButtonSensors. SingleControl-Buttons gibt es da nicht
* PWMLightingActor kennen die Liste der Ups, der Downs, der SingleControls und der RotarySensors. Ein derartiger Actor ist für die kompletten LEDs eines Raumbereichs zuständig (nicht: jeder Spot hat eine eigene App)
* BinaryActor kennt Toggles, Ons, Offs, Heartbeats
* SoundActor hat eine Map EventId🡪Song-Index

## Task-Konzept

* Sensor-Task holt sich die Messwerte der I2C und 1Wire-Sensoren
* PLC-Task agiert im 50ms-Zyklus