EER34 – API Elemon para stack LoRaWan de SAMR34

# Introducción

Este documento es una versión inicial, incompleta, provisoria y preliminar.

El objetivo de esta API es crear un ambiente amigable para desarrollar aplicaciones bajo AtmeStudio para el SAMR34, usando el stack LoRaWan de Microchip, basado en las aplicaciones de demo, pero con una API más sencilla, con prestaciones similares a los comandos del RN2903, y con una estructura simple de la aplicación, similar a la de Arduino.

# Funciones de la API

Funciones de inicialización:

void EES34\_reset(void);

void EER34\_init(void);

void EER34\_tickStart(int t);

void EER34\_tickStop(void);

int EER34\_sleep(uint32\_t time);

Funciones de seteos de parametros:

int EER34\_setDevEui(uint8\_t \*devEui);

int EER34\_setAppEui(uint8\_t \*appEui);

int EER34\_setAppKey(uint8\_t \*appKey);

int EER34\_setDevAddr(uint32\_t \*devAddr);

int EER34\_setAppSKey(uint8\_t \*appSKey);

int EER34\_setNwkSKey(uint8\_t \*nwkSKey);

int EER34\_setBand(IsmBand\_t band, int subBand);

Funciones de operación LoRaWan:

int EER34\_joinOTAA(void);

int EER34\_joinABP(void);

int EER34\_tx(EER34\_txMode\_t mode, int port, uint8\_t \*data, int len);

Callbacks a la aplicación:

void EER34\_statusCallback(EER34\_status\_t sts, StackRetStatus\_t LoraSts);

void EER34\_rxDataCallback(int port, uint8\_t \*data, int len);

void EER34\_tickCallback(void);

void EES34\_enterLowPower(void);

void EES34\_exitLowPower(void);

Funciones de la aplicación:

void EES34\_appInit(void);

void EES34\_appTask(void);

# Proyecto de Ejemplo

El proyecto de ejemplo hace el join OTAA y transmite cada 5 segundos un paquete fijo con confirmación, imprimiendo mensajes explicando las acciones que va realizando, y cada tres paquetes enviados se pone en modo de bajo consumo por 15 segundos.

Usa las calbacks y el timer de tick.

Está todo armado, compila y funciona bien.

Todos los archivos fuente .c y .h de la aplicación se pueden poner en: ProjectFolder\src\EER34\App\

El archivo principal de la aplicación es app.c, y en el ejemplo es el único archivo fuente de la aplicación. En principio no hay un app.h porque todas las funciones de app.c que son parte de la API están declaradas en eer34.h.

Las funciones de calback que son opcionales están declaradas weak en eer34.c, de manera que en app.c solo hace falta que poner las que use la aplicación.

# Como se Creó el Proyecto de Ejemplo

Partiendo del proyecto APPS\_ED\_DEMO\_SER\_NWK\_PROVISION1.atsln se creó una carpeta nueva colgada de Src, que se llame EER34. Al crear la carpeta en el proyecto el AtmelStudio crea un subdirectorio, y ahí se pusieron los archivos EER34.c y .h, que hay que agregarlos a esa carpeta del proyecto.

Se agregó el include de eer34.h al comienzo de main.c y se agrego el path correspondiente en el proyecto.

En main.c se reemplazaron las líneas:

SwTimerCreate(&demoTimerId);

SwTimerCreate(&lTimerId);

mote\_demo\_init();

por:

EER34\_init();

Y un poco antes se reemplazó la línea:

print\_reset\_causes();

por:

EES34\_reset();

Y se borro en el main la función print\_reset\_causes();

Se creó otra carpeta de proyecto colgada de EER34 llamada App, para poner los archivos .c y .h de la aplicación. El archivo app.c es el principal, donde este el task de la aplicación, la inicialización, callbacks, etc.

Se eliminaron los archivos enddevice\_Demo.c y enddevice\_cert.c.

En app.c se crearon las funciones de inicialización task y callbacks necesarias para la aplicación de ejemplo.