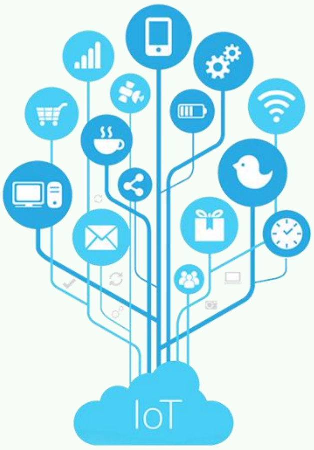
TP n°2 : Solutions

MASTER RÉSEAUX ET TÉLÉCOM



VERSION : 10 juin 2021

Florent NOLOT
UNIVERSITÉ DE REIMS CHAMPAGNE ARDENNE

Table des matières

I. Solutions

Pour ce tutoriel on utilise deux LaunchPads. Le LaunchPad CC1350 est utilisé pour l'émission de paquets (TX) et le LaunchPad CC1352 est utilisé pour la réception de paquets (RX).

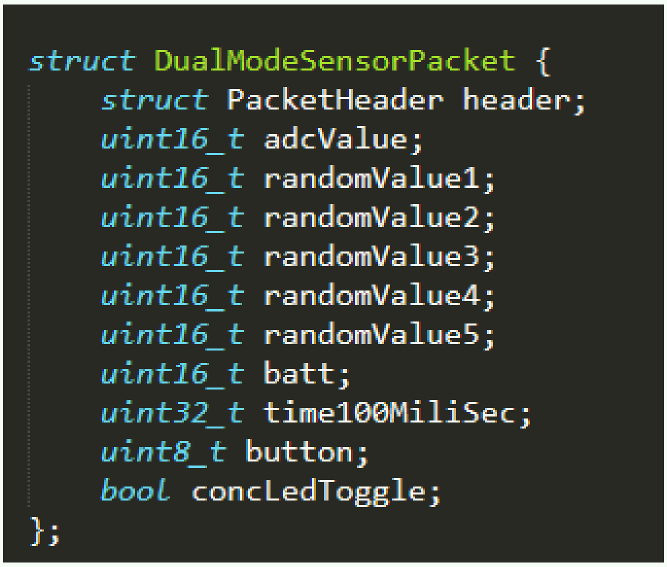
A. Exercice 1 – Transfert de données

L'objectif de cet exercice est de comprendre l'exemple importé précédemment. Modifiez le programme afin d'envoyer des données aléatoires de l'émetteur au récepteur, et afficher ces données grâce à l'émulateur série.

Astuce : Pour obtenir la valeur de voltage de la batterie, il faut diviser l'entier obtenu par 256.

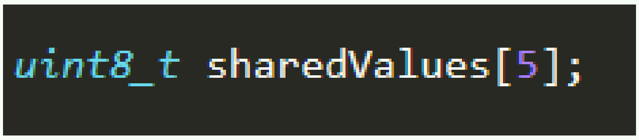
1. Programme d'émission – CC1352

Modification du fichier « RadioProtocol.h »



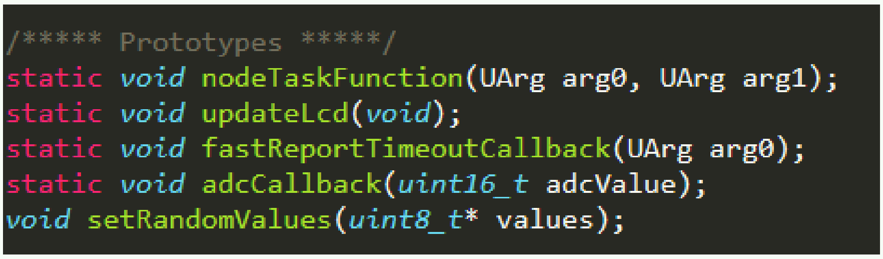
```
struct DualModeSensorPacket {
    struct PacketHeader header;
    uint16_t adcValue;
    uint16_t randomValue1;
    uint16_t randomValue2;
    uint16_t randomValue3;
    uint16_t randomValue4;
    uint16_t randomValue5;
    uint16_t batt;
    uint32_t time100MiliSec;
    uint8_t button;
    bool concLedToggle;
};
```

Modification du fichier « NodeTask.h »

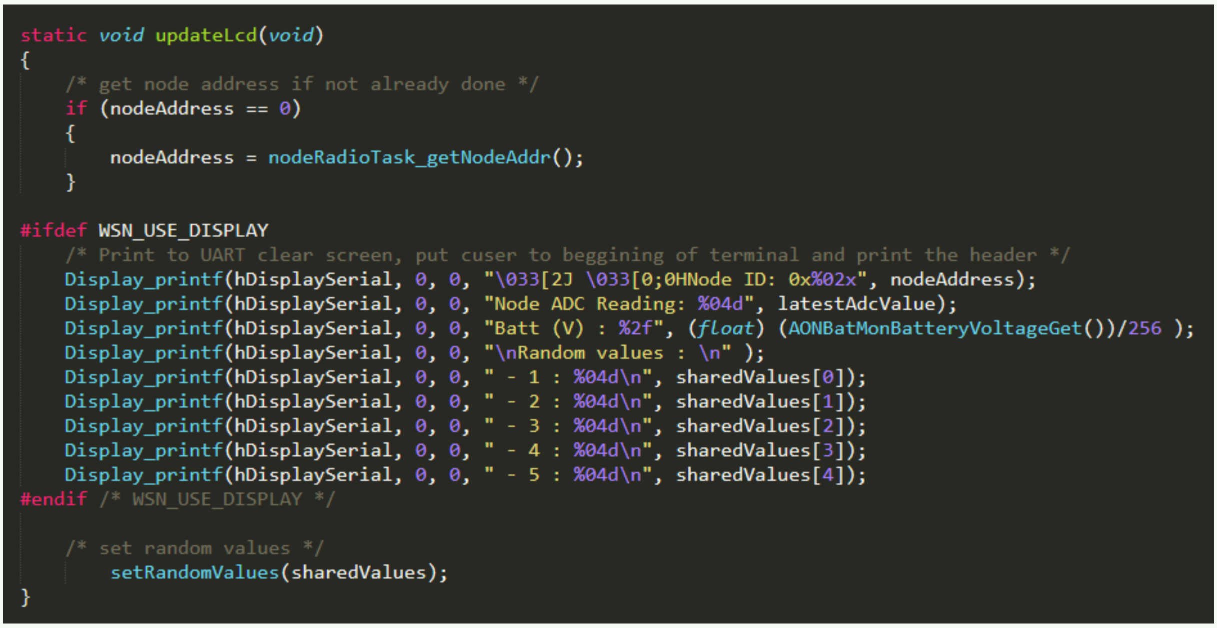


```
uint8_t sharedValues[5];
```

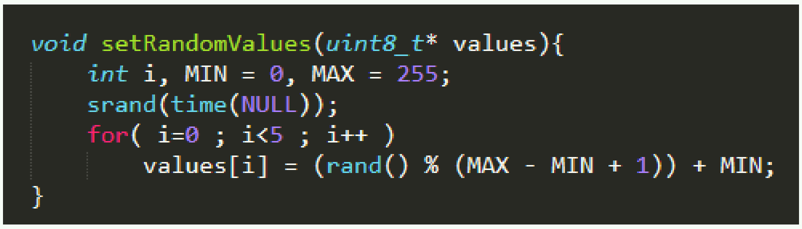
Modification du fichier « NodeTask.c »



```
/***** Prototypes *****/
static void nodeTaskFunction(UArg arg0, UArg arg1);
static void updateLcd(void);
static void fastReportTimeoutCallback(UArg arg0);
static void adcCallback(uint16_t adcValue);
void setRandomValues(uint8_t* values);
```

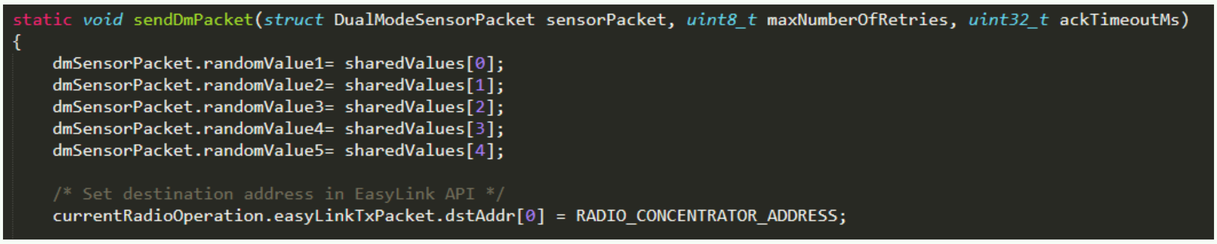


```
static void updateLcd(void)
{
    /* get node address if not already done */
    if (nodeAddress == 0)
    {
        nodeAddress = nodeRadioTask_getNodeAddr();
    }

#ifdef WSN_USE_DISPLAY
    /* Print to UART clear screen, put cuser to beggining of terminal and print the header */
    Display_printf(hDisplaySerial, 0, 0, "\033[2J \033[0;0HNode ID: 0x%02x", nodeAddress);
    Display_printf(hDisplaySerial, 0, 0, "Node ADC Reading: %04d", latestAdcValue);
    Display_printf(hDisplaySerial, 0, 0, "Batt (V) : %2f", (float) (AONBatMonBatteryVoltageGet())/256 );
    Display_printf(hDisplaySerial, 0, 0, "\nRandom values : \n" );
    Display_printf(hDisplaySerial, 0, 0, " - 1 : %04d\n", sharedValues[0]);
    Display_printf(hDisplaySerial, 0, 0, " - 2 : %04d\n", sharedValues[1]);
    Display_printf(hDisplaySerial, 0, 0, " - 3 : %04d\n", sharedValues[2]);
    Display_printf(hDisplaySerial, 0, 0, " - 4 : %04d\n", sharedValues[3]);
    Display_printf(hDisplaySerial, 0, 0, " - 5 : %04d\n", sharedValues[4]);
#endif /* WSN_USE_DISPLAY */

    /* set random values */
        setRandomValues(sharedValues);
}
```

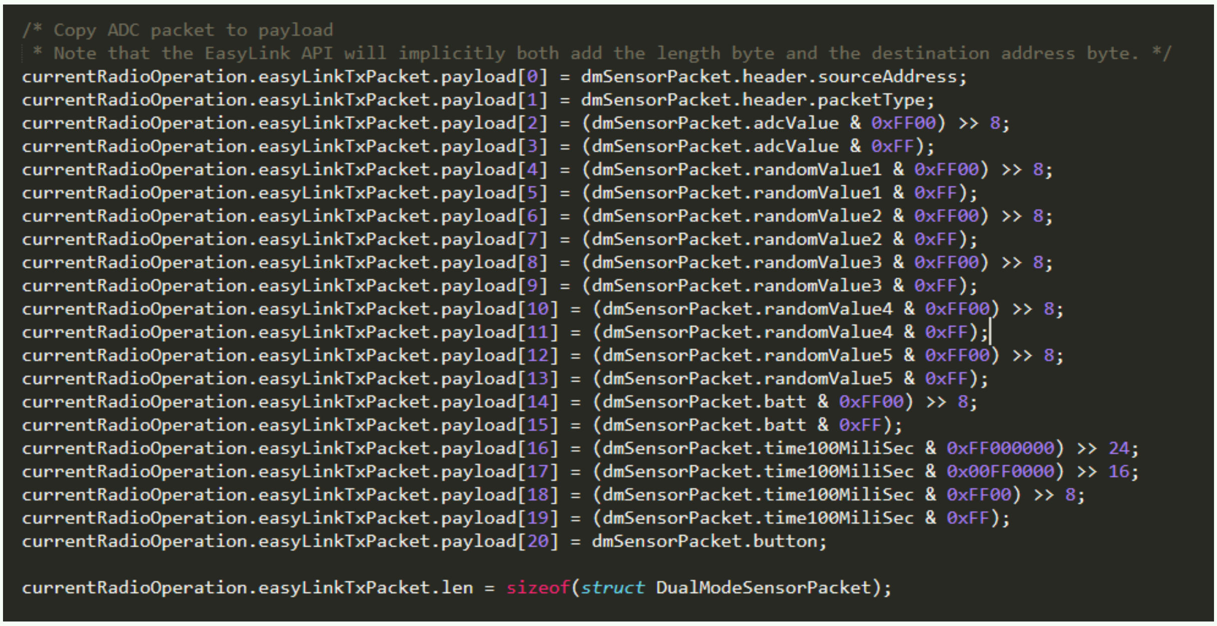


```
void setRandomValues(uint8_t* values){
    int i, MIN = 0, MAX = 255;
    srand(time(NULL));
    for( i=0 ; i<5 ; i++ )
        values[i] = (rand() % (MAX - MIN + 1)) + MIN;
}
```

Modification du fichier « NodeRadioTask.c »



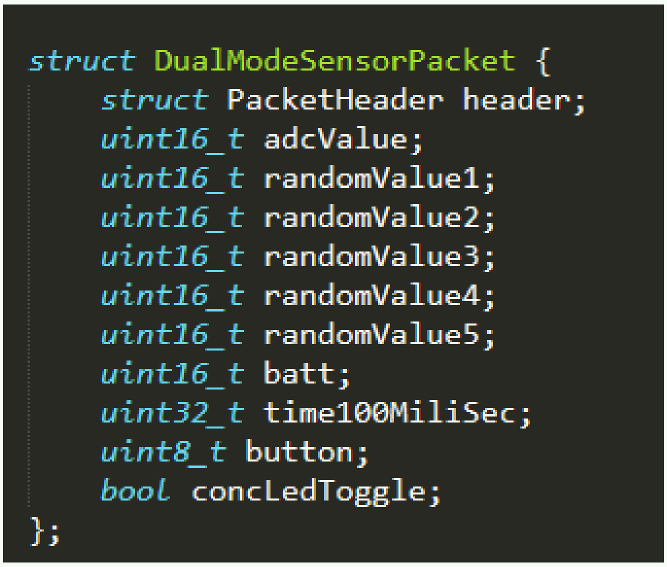
```
static void sendDmPacket(struct DualModeSensorPacket sensorPacket, uint8_t maxNumberOfRetries, uint32_t ackTimeoutMs)
{
    dmSensorPacket.randomValue1= sharedValues[0];
    dmSensorPacket.randomValue2= sharedValues[1];
    dmSensorPacket.randomValue3= sharedValues[2];
    dmSensorPacket.randomValue4= sharedValues[3];
    dmSensorPacket.randomValue5= sharedValues[4];

    /* Set destination address in EasyLink API */
    currentRadioOperation.easyLinkTxPacket.dstAddr[0] = RADIO_CONCENTRATOR_ADDRESS;
```



```
/* Copy ADC packet to payload
 * Note that the EasyLink API will implicitly both add the length byte and the destination address byte. */
currentRadioOperation.easyLinkTxPacket.payload[0] = dmSensorPacket.header.sourceAddress;
currentRadioOperation.easyLinkTxPacket.payload[1] = dmSensorPacket.header.packetType;
currentRadioOperation.easyLinkTxPacket.payload[2] = (dmSensorPacket.adcValue & 0xFF00) >> 8;
currentRadioOperation.easyLinkTxPacket.payload[3] = (dmSensorPacket.adcValue & 0xFF);
currentRadioOperation.easyLinkTxPacket.payload[4] = (dmSensorPacket.randomValue1 & 0xFF00) >> 8;
currentRadioOperation.easyLinkTxPacket.payload[5] = (dmSensorPacket.randomValue1 & 0xFF);
currentRadioOperation.easyLinkTxPacket.payload[6] = (dmSensorPacket.randomValue2 & 0xFF00) >> 8;
currentRadioOperation.easyLinkTxPacket.payload[7] = (dmSensorPacket.randomValue2 & 0xFF);
currentRadioOperation.easyLinkTxPacket.payload[8] = (dmSensorPacket.randomValue3 & 0xFF00) >> 8;
currentRadioOperation.easyLinkTxPacket.payload[9] = (dmSensorPacket.randomValue3 & 0xFF);
currentRadioOperation.easyLinkTxPacket.payload[10] = (dmSensorPacket.randomValue4 & 0xFF00) >> 8;
currentRadioOperation.easyLinkTxPacket.payload[11] = (dmSensorPacket.randomValue4 & 0xFF);
currentRadioOperation.easyLinkTxPacket.payload[12] = (dmSensorPacket.randomValue5 & 0xFF00) >> 8;
currentRadioOperation.easyLinkTxPacket.payload[13] = (dmSensorPacket.randomValue5 & 0xFF);
currentRadioOperation.easyLinkTxPacket.payload[14] = (dmSensorPacket.batt & 0xFF00) >> 8;
currentRadioOperation.easyLinkTxPacket.payload[15] = (dmSensorPacket.batt & 0xFF);
currentRadioOperation.easyLinkTxPacket.payload[16] = (dmSensorPacket.time100MiliSec & 0xFF000000) >> 24;
currentRadioOperation.easyLinkTxPacket.payload[17] = (dmSensorPacket.time100MiliSec & 0x00FF0000) >> 16;
currentRadioOperation.easyLinkTxPacket.payload[18] = (dmSensorPacket.time100MiliSec & 0xFF00) >> 8;
currentRadioOperation.easyLinkTxPacket.payload[19] = (dmSensorPacket.time100MiliSec & 0xFF);
currentRadioOperation.easyLinkTxPacket.payload[20] = dmSensorPacket.button;

currentRadioOperation.easyLinkTxPacket.len = sizeof(struct DualModeSensorPacket);
```

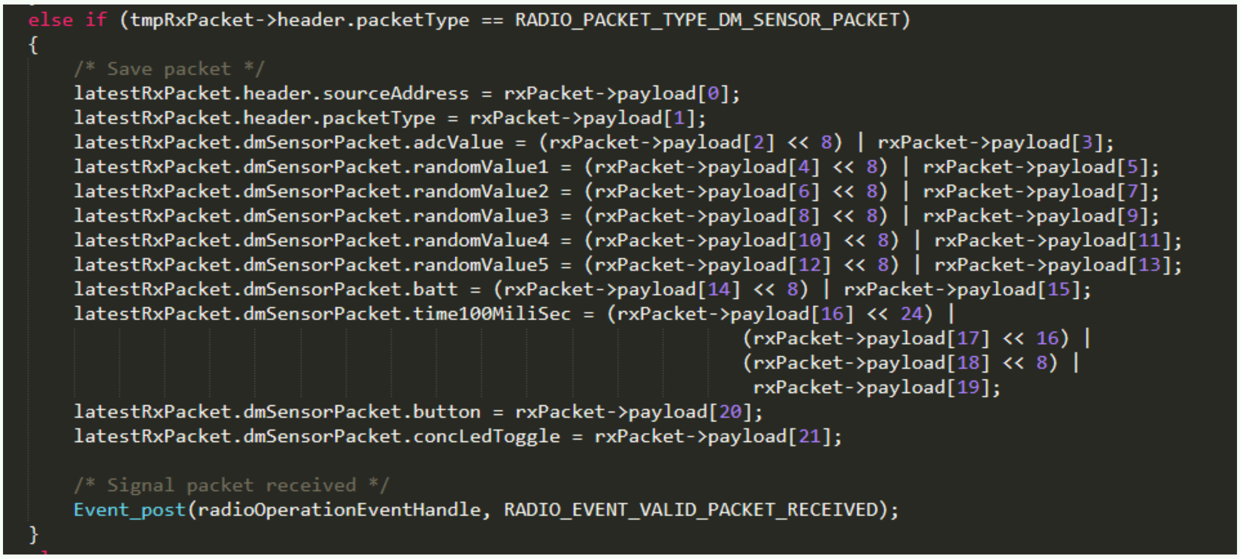
2. Programme de réception - CC1350

Modification du fichier « RadioProtocol.h »



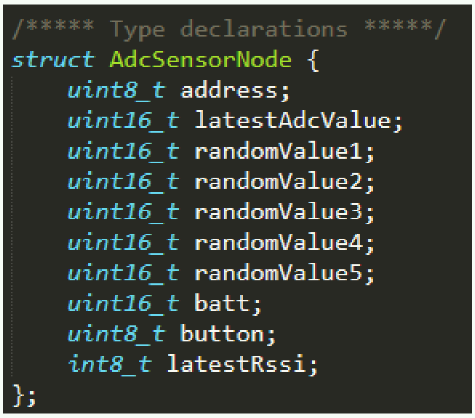
```
struct DualModeSensorPacket {
    struct PacketHeader header;
    uint16_t adcValue;
    uint16_t randomValue1;
    uint16_t randomValue2;
    uint16_t randomValue3;
    uint16_t randomValue4;
    uint16_t randomValue5;
    uint16_t batt;
    uint32_t time100MiliSec;
    uint8_t button;
    bool concLedToggle;
};
```

Modification du fichier « ConcentratorRadioTask.c »



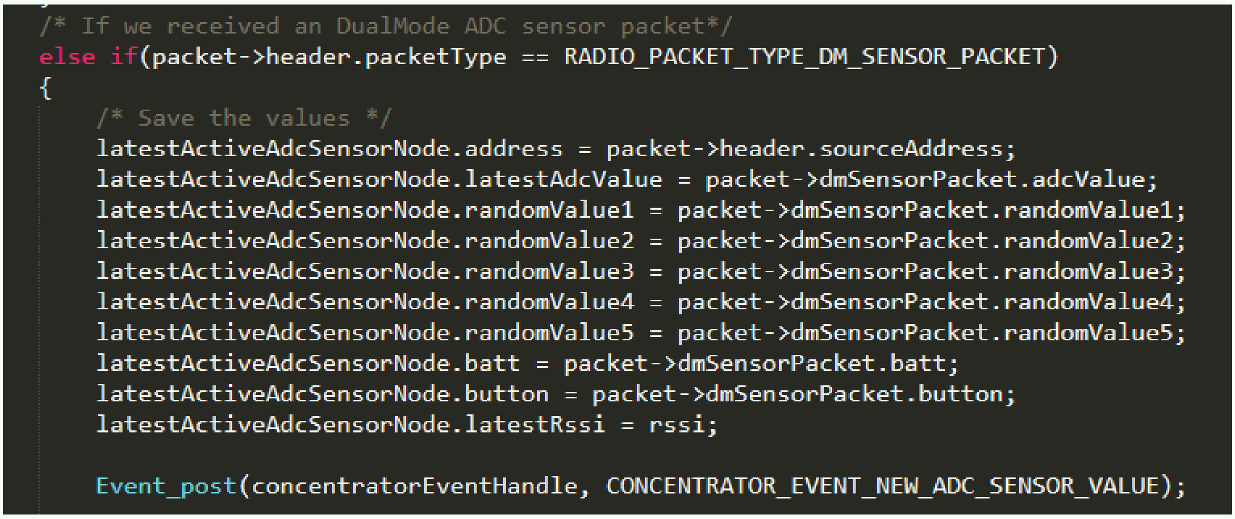
```c
else if (tmpRxPacket->header.packetType == RADIO_PACKET_TYPE_DM_SENSOR_PACKET)
{
    /* Save packet */
    latestRxPacket.header.sourceAddress = rxPacket->payload[0];
    latestRxPacket.header.packetType = rxPacket->payload[1];
    latestRxPacket.dmSensorPacket.adcValue = (rxPacket->payload[2] << 8) | rxPacket->payload[3];
    latestRxPacket.dmSensorPacket.randomValue1 = (rxPacket->payload[4] << 8) | rxPacket->payload[5];
    latestRxPacket.dmSensorPacket.randomValue2 = (rxPacket->payload[6] << 8) | rxPacket->payload[7];
    latestRxPacket.dmSensorPacket.randomValue3 = (rxPacket->payload[8] << 8) | rxPacket->payload[9];
    latestRxPacket.dmSensorPacket.randomValue4 = (rxPacket->payload[10] << 8) | rxPacket->payload[11];
    latestRxPacket.dmSensorPacket.randomValue5 = (rxPacket->payload[12] << 8) | rxPacket->payload[13];
    latestRxPacket.dmSensorPacket.batt = (rxPacket->payload[14] << 8) | rxPacket->payload[15];
    latestRxPacket.dmSensorPacket.time100MiliSec = (rxPacket->payload[16] << 24) |
                                                               (rxPacket->payload[17] << 16) |
                                                               (rxPacket->payload[18] << 8) |
                                                                rxPacket->payload[19];
    latestRxPacket.dmSensorPacket.button = rxPacket->payload[20];
    latestRxPacket.dmSensorPacket.concLedToggle = rxPacket->payload[21];

    /* Signal packet received */
    Event_post(radioOperationEventHandle, RADIO_EVENT_VALID_PACKET_RECEIVED);
}
```

Modification du fichier « ConcentratorTask.h»

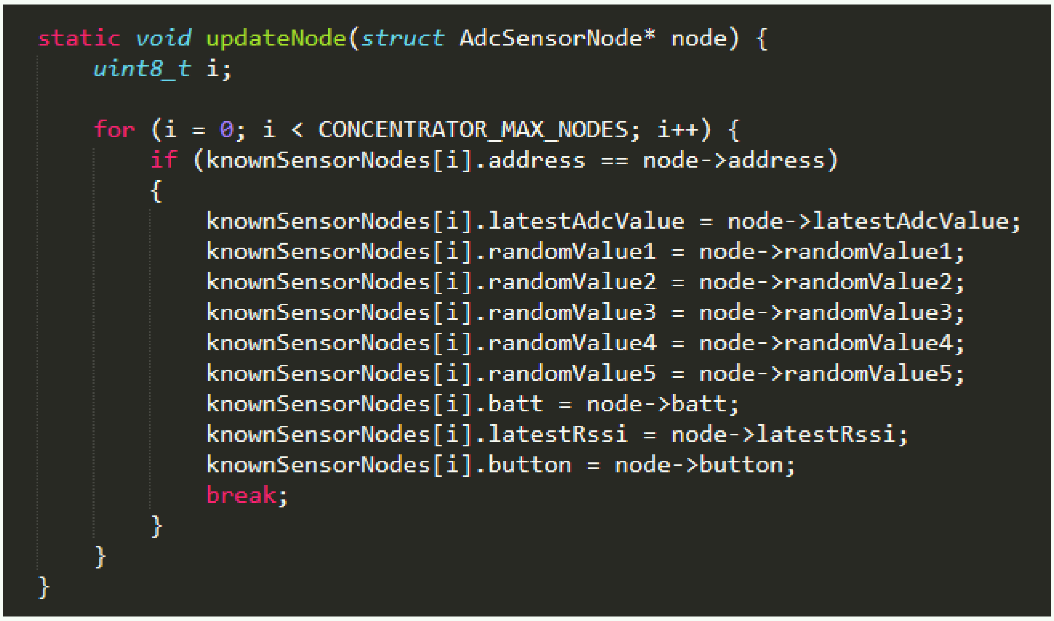


```c
/***** Type declarations *****/
struct AdcSensorNode {
    uint8_t address;
    uint16_t latestAdcValue;
    uint16_t randomValue1;
    uint16_t randomValue2;
    uint16_t randomValue3;
    uint16_t randomValue4;
    uint16_t randomValue5;
    uint16_t batt;
    uint8_t button;
    int8_t latestRssi;
};
```

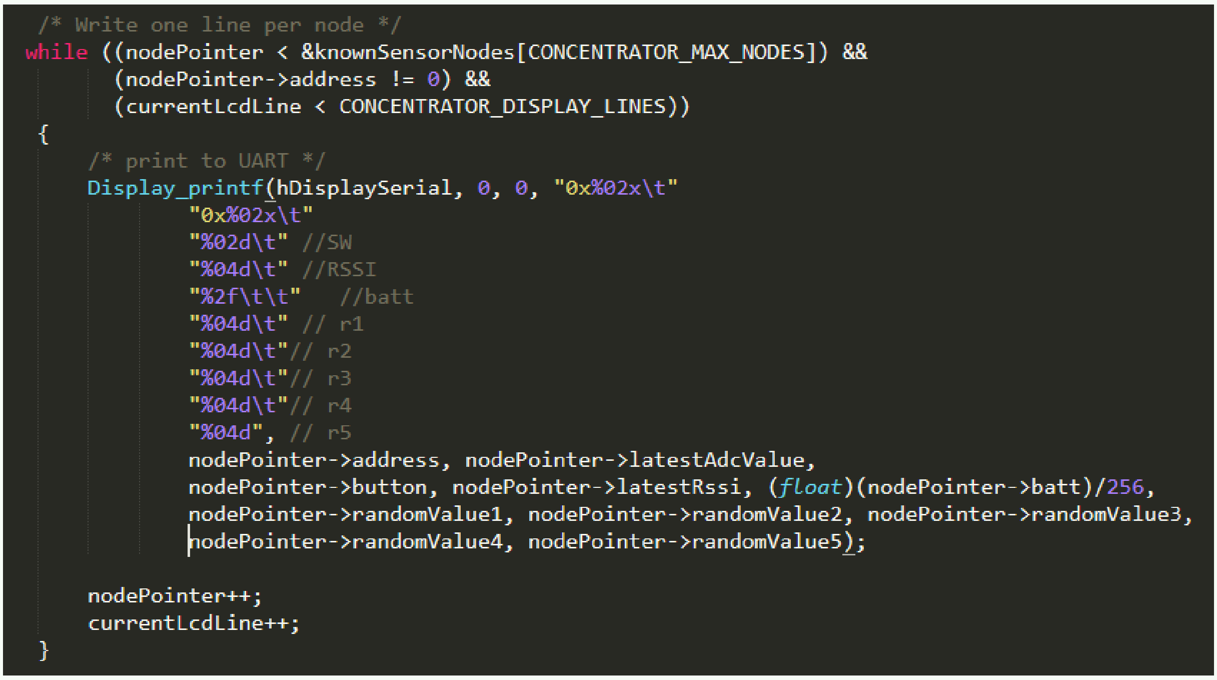
Modification du fichier « ConcentratorTask.c»



```c
/* If we received an DualMode ADC sensor packet*/
else if(packet->header.packetType == RADIO_PACKET_TYPE_DM_SENSOR_PACKET)
{
    /* Save the values */
    latestActiveAdcSensorNode.address = packet->header.sourceAddress;
    latestActiveAdcSensorNode.latestAdcValue = packet->dmSensorPacket.adcValue;
    latestActiveAdcSensorNode.randomValue1 = packet->dmSensorPacket.randomValue1;
    latestActiveAdcSensorNode.randomValue2 = packet->dmSensorPacket.randomValue2;
    latestActiveAdcSensorNode.randomValue3 = packet->dmSensorPacket.randomValue3;
    latestActiveAdcSensorNode.randomValue4 = packet->dmSensorPacket.randomValue4;
    latestActiveAdcSensorNode.randomValue5 = packet->dmSensorPacket.randomValue5;
    latestActiveAdcSensorNode.batt = packet->dmSensorPacket.batt;
    latestActiveAdcSensorNode.button = packet->dmSensorPacket.button;
    latestActiveAdcSensorNode.latestRssi = rssi;

    Event_post(concentratorEventHandle, CONCENTRATOR_EVENT_NEW_ADC_SENSOR_VALUE);
```



```
static void updateNode(struct AdcSensorNode* node) {
    uint8_t i;

    for (i = 0; i < CONCENTRATOR_MAX_NODES; i++) {
        if (knownSensorNodes[i].address == node->address)
        {
            knownSensorNodes[i].latestAdcValue = node->latestAdcValue;
            knownSensorNodes[i].randomValue1 = node->randomValue1;
            knownSensorNodes[i].randomValue2 = node->randomValue2;
            knownSensorNodes[i].randomValue3 = node->randomValue3;
            knownSensorNodes[i].randomValue4 = node->randomValue4;
            knownSensorNodes[i].randomValue5 = node->randomValue5;
            knownSensorNodes[i].batt = node->batt;
            knownSensorNodes[i].latestRssi = node->latestRssi;
            knownSensorNodes[i].button = node->button;
            break;
        }
    }
}
```



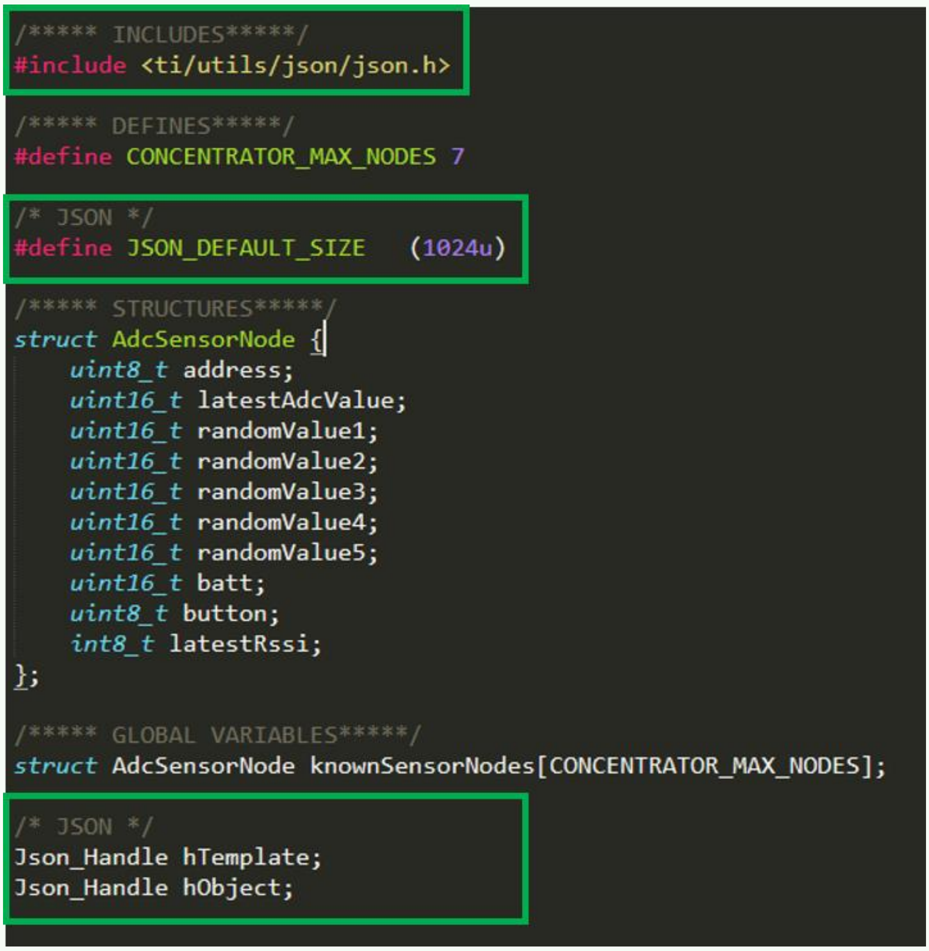
```
 /* Write one line per node */
while ((nodePointer < &knownSensorNodes[CONCENTRATOR_MAX_NODES]) &&
        (nodePointer->address != 0) &&
        (currentLcdLine < CONCENTRATOR_DISPLAY_LINES))
 {
     /* print to UART */
     Display_printf(hDisplaySerial, 0, 0, "0x%02x\t"
             "0x%02x\t"
             "%02d\t" //SW
             "%04d\t" //RSSI
             "%2f\t\t"   //batt
             "%04d\t" // r1
             "%04d\t"// r2
             "%04d\t"// r3
             "%04d\t"// r4
             "%04d", // r5
             nodePointer->address, nodePointer->latestAdcValue,
             nodePointer->button, nodePointer->latestRssi, (float)(nodePointer->batt)/256,
             nodePointer->randomValue1, nodePointer->randomValue2, nodePointer->randomValue3,
             nodePointer->randomValue4, nodePointer->randomValue5);

     nodePointer++;
     currentLcdLine++;
 }
```

B. Exercice 2 – Mise en forme des données

Modifiez le programme précédent en y incorporant une fonction permettant de formatter les données reçues par le concentrateur en utilisant l'API JSON. Le résultat peut être affiché grâce à l'émulateur de série.

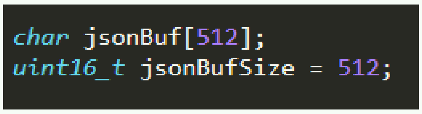
Les programmes d'émission et de réception sont identiques mise à part les fichiers ConcentratorTask

Modification du fichier « ConcentratorTask.h »

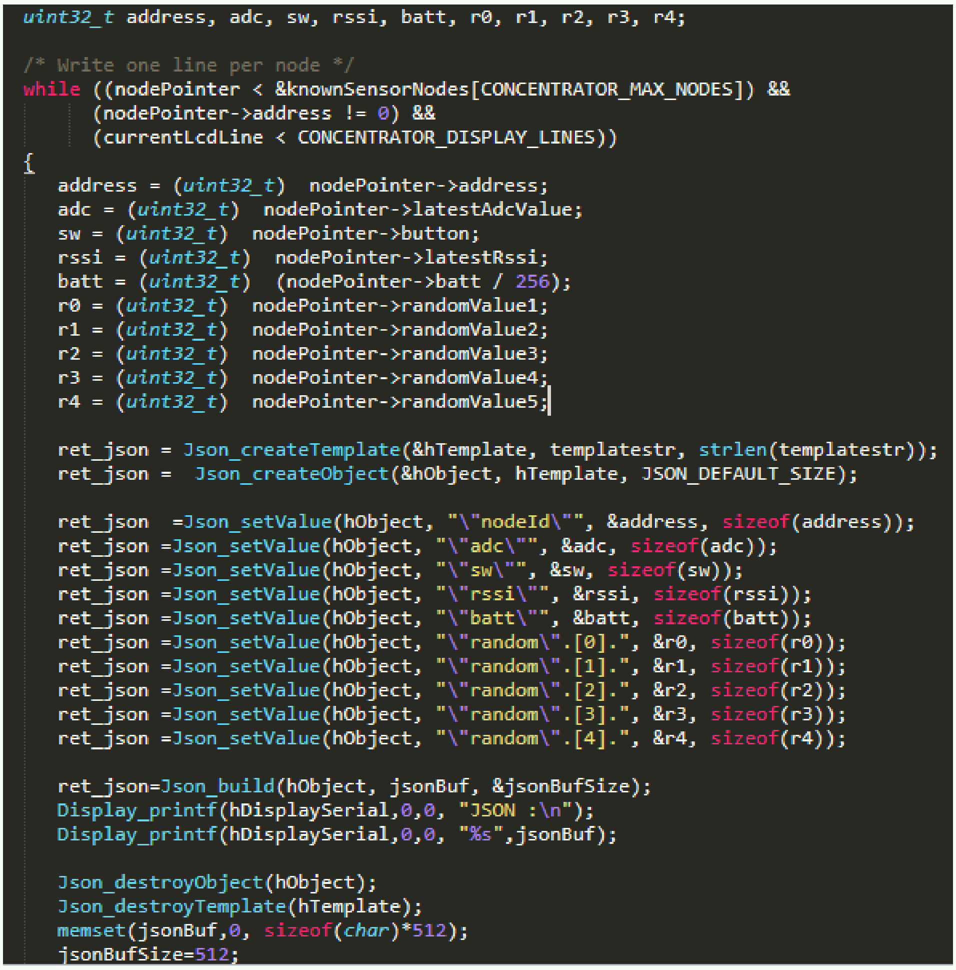


```
/***** INCLUDES*****/
#include <ti/utils/json/json.h>

/***** DEFINES*****/
#define CONCENTRATOR_MAX_NODES 7

/* JSON */
#define JSON_DEFAULT_SIZE   (1024u)

/***** STRUCTURES*****/
struct AdcSensorNode {
    uint8_t address;
    uint16_t latestAdcValue;
    uint16_t randomValue1;
    uint16_t randomValue2;
    uint16_t randomValue3;
    uint16_t randomValue4;
    uint16_t randomValue5;
    uint16_t batt;
    uint8_t button;
    int8_t latestRssi;
};

/***** GLOBAL VARIABLES*****/
struct AdcSensorNode knownSensorNodes[CONCENTRATOR_MAX_NODES];

/* JSON */
Json_Handle hTemplate;
Json_Handle hObject;
```

Modification du fichier « ConcentratorTask.c »



```
char jsonBuf[512];
uint16_t jsonBufSize = 512;
```



```
uint32_t address, adc, sw, rssi, batt, r0, r1, r2, r3, r4;

/* Write one line per node */
while ((nodePointer < &knownSensorNodes[CONCENTRATOR_MAX_NODES]) &&
       (nodePointer->address != 0) &&
       (currentLcdLine < CONCENTRATOR_DISPLAY_LINES))
{
   address = (uint32_t)  nodePointer->address;
   adc = (uint32_t)  nodePointer->latestAdcValue;
   sw = (uint32_t)  nodePointer->button;
   rssi = (uint32_t)  nodePointer->latestRssi;
   batt = (uint32_t)  (nodePointer->batt / 256);
   r0 = (uint32_t)  nodePointer->randomValue1;
   r1 = (uint32_t)  nodePointer->randomValue2;
   r2 = (uint32_t)  nodePointer->randomValue3;
   r3 = (uint32_t)  nodePointer->randomValue4;
   r4 = (uint32_t)  nodePointer->randomValue5;

   ret_json = Json_createTemplate(&hTemplate, templatestr, strlen(templatestr));
   ret_json =  Json_createObject(&hObject, hTemplate, JSON_DEFAULT_SIZE);

   ret_json  =Json_setValue(hObject, "\"nodeId\"", &address, sizeof(address));
   ret_json =Json_setValue(hObject, "\"adc\"", &adc, sizeof(adc));
   ret_json =Json_setValue(hObject, "\"sw\"", &sw, sizeof(sw));
   ret_json =Json_setValue(hObject, "\"rssi\"", &rssi, sizeof(rssi));
   ret_json =Json_setValue(hObject, "\"batt\"", &batt, sizeof(batt));
   ret_json =Json_setValue(hObject, "\"random\".[0].", &r0, sizeof(r0));
   ret_json =Json_setValue(hObject, "\"random\".[1].", &r1, sizeof(r1));
   ret_json =Json_setValue(hObject, "\"random\".[2].", &r2, sizeof(r2));
   ret_json =Json_setValue(hObject, "\"random\".[3].", &r3, sizeof(r3));
   ret_json =Json_setValue(hObject, "\"random\".[4].", &r4, sizeof(r4));

   ret_json=Json_build(hObject, jsonBuf, &jsonBufSize);
   Display_printf(hDisplaySerial,0,0, "JSON :\n");
   Display_printf(hDisplaySerial,0,0, "%s",jsonBuf);

   Json_destroyObject(hObject);
   Json_destroyTemplate(hTemplate);
   memset(jsonBuf,0, sizeof(char)*512);
   jsonBufSize=512;
```