# NBI数据接口V2

数据结构原始定义：

#define STRUCT\_VERSION 2 //传感器结构体协议版本号

#define SENSOR\_FIELD 20 //最多有多少个传感器数据域

#define SOCKETS 6 //有多少个485传感器插槽

#pragma pack(1) //数据密集对齐

typedef struct

{

uint8\_t structver; //结构体版本，应赋值为STRUCT\_VERSION

uint8\_t count; //传感器数据域数量

uint8\_t sockets[SOCKETS]; //接口所接的传感器标识

uint16\_t fields[SENSOR\_FIELD]; //传感器数据域

}sample\_t;

图表结构：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ver版本号 | 数据域数量 | 扩展接口标识\*6 | | 数据域\*MAX20 |
| 字段标识 | 字段数量 |
| 1byte,0x02 | 1byte | 5bit | 3bit | 2byte |

数据结构成员字段说明：

structver：数据接口版本号，长度1字节，此版本固定为0x02

count：数据域数量，等同fields实际存储的有效数据域数量

sockets：扩展接口标识，长度1字节，高5bit存储字段标识，低3bit存储该扩展接口的数据域数量

fields：实际存放的传感器数据域，根据：电量，光照，空气温度，空气湿度，485传感器（根据接口的传感器标识判断）的顺序存取

fields传感器数据域字段说明：

fields数据域使用无符号16位整形方式存储传感器数据，v2版本格式中根据电量，光照，空气温度，空气湿度，485传感器（根据接口的传感器标识判断）的顺序存储，但数据的实际类型因数据域类型不同而不同。fields数组中存储结构如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 偏移 | 0 | 1 | 2 | 3 | 4 | 5~MAX20 |
| 数据意义 | 电量 | 光照 | | 空气温度 | 空气湿度 | 根据sockets标识位决定 |
| 数据类型 | 无符号整形 | 无符号32位 | | 有符号1位小数 | 无符号1位小数 | 根据sockets标识位决定 |

其中，电量、光照、空气温度、空气湿度为固定字段，fields至少有5个有效数据，即count至少为5，否则整条数据无效。fields第5位至MAX20位存储的数据类型和数据意义根据sockets里面的标识位决定。

sockets传感器标识字段说明：

sockets传感器标识字段v2.0结构中总共由6个，6个标识位结构一致，其中第一个表示不带扩展盒直接连接主机上的扩展485设备，剩下的5个标识位标识连接在扩展盒上的扩展485设备。第一位和后5位不会同时存在有效数据，即一条数据中，设备或者没有连接任何扩展设备，或者只会出现直接连接在主机上的扩展传感器或扩展盒中的一种。

sockets传感器标识字段由高5bit字段标识和低3bit字段数量组成

1. 字段数量：

标识该标识位所标识的扩展接口上连接的传感器数据域数量，占低3bit，有效数据1~7。如果数据字段为0，表示该扩展接口上没有传感器数据，若该数据大于1，则表示该标识位上所标识的扩展口连接的传感器有多于一个传感器数据，传感器的字段标识顺延，如：sockets[0]为0x12，解析出字段标识为2，字段数量为2，则表示该标识位上包含编号2、3的两个传感器数据。

1. 字段标识：

字段标识占用高5bit，有效取值范围为1~31，若该字段为0，表示该标识位所标识的扩展口不包含扩展传感器，否则根据代表该标识位所标识的扩展接口上连接的传感器数据意义和类型。

字段编号和对应的意义、单位、数据类型

|  |  |  |  |
| --- | --- | --- | --- |
| **标识编号** | **说明** | **单位** | **数据类型和有效范围** |
| 1 | 光照度 | Lux | 0~376000，无符号双字2位小数 |
| 2 | 空气温度 | ℃ | -40~125，有符号1位小数 |
| 3 | 空气湿度 | % | 0~100，无符号1位小数 |
| 4 | 土壤温度 | ℃ | -30~70，有符号1位小数 |
| 5 | 土壤湿度 | % | 0~100，无符号1位小数 |
| 6 | 光和有效度 | μmol/m2•s | 0~2500，无符号整形 |
| 7 | 叶面温度 | ℃ | -30~70，有符号1位小数 |
| 8 | 叶面湿度 | % | 0~100，无符号2位小数 |
| 9 | 降雨量 | mm | 0~6553，有符号1位小数 |
| 10 | 风速 | m/s | 0~100，无符号一位小数 |
| 11 | 风向 | ° | 0~360，无符号一位小数 |
| 12 | 土壤EC值 | mS/cm | 0~20，无符号2位小数 |
| 13 | 土壤盐度 | ppm | 有符号1位小数小数 |
| 14 | 二氧化碳浓度 | ppm | 0~50000，无符号整形 |
| 15 | 大气压 | hPa | 0~1100，无符号整形 |
| 16 | 土壤PH值 | 无 | 0~14，有符号1位小数 |
| 17~31 | 保留 | 保留 | 保留 |

根据对应的标识编号，需要对数据进行相应的计算和赋值，如果时双字型数据，该数据占用2个数据域。如果数据是有符号数，需要将数据强制转换为有符号型，根据数据小数点位数，需要对原始数据分别除以10（1位小数）或除以100（2位小数）。

实际有效数据大小：

#define SAMPLE\_SIZE (samples.count\*2+SOCKETS+2)

数据解码过程：

1. 判断第一个字节数据结构版本是否为0x02，否则跳出
2. 判断数据域数量count是否大于或等于5且数据长度是否等于(count\*2+8)，否则跳出
3. 读取第一位数据域作为电量；读取第二、三位数据域，拼接为32位数并除以100.0作为光照；读取第四位数据域并强制转换为有符号16位数，除以10.0作为空气温度；读取第五位数据域并除以10.0作为空气湿度。
4. 读取sockets0~5位，拆分为数据域标识和数据域数量，根据数据域数量读取对应数量的下一个数据域（如读完空气湿度当前数据域为第五，则下一个读取的数据域为6），并根据数据域标识的数据类型进行计算。

数据解码PHP示例：

数据解码c语言示例：

enum FIELDS\_TYPE{

SENSOR\_BATTERY =0,

ILLUMINATION =1,

AIR\_TEMPERATURE =2,

AIR\_HUMIDITY =3,

SOIL\_TEMPERATURE =4,

SOIL\_HUMIDITY =5,

PHOTO\_RADIO =6,

LEAF\_TEMPERATURE =7,

LEAF\_HUMIDITY =8,

RAIN\_FALL =9,

WIND\_SPEED =10,

WIND\_DIRECTION =11,

SOIL\_EC =12,

SOIL\_SALINITY =13,

CO2 =14,

ATMOSPHERIC =15,

SOIL\_PH =16

};

typedef enum

{

U16=0,

S16=1,

U16D1=2,

S16D1=3,

U16D2=4,

S16D2=5,

U32D2=0xff

}DataType;

typedef struct

{

char\* fieldName;

DataType type;

}FieldInfo;

FieldInfo fieldInfo[]=

{

{"Battery",U16},

{"Illumination",U32D2},

{"AirTemperature",S16D1},

{"AirHumidity",U16D1},

{"SoilTemperature",S16D1},

{"SoilHumidity",U16D1},

{"Photosynthetic",U16},

{"LeafTemperature",S16D1},

{"LeafHumidity",U16D2},

{"RainFall",U16D1},

{"WindSpeed",U16D1},

{"WindDirection",U16D1},

{"SoilEleCond",U16D2},

{"SoilSalinity",U16D1},

{"Co2",U16},

{"Atmospheric",U16D1},

{"SoilPH",U16D2}

};

/\*

\* GetSensorSample: 获取传感器的一次采样结果字符串

\* str： 结果字符串指针，调用者需要自行分配释放内存

\* 返回值： 数据长度

\*/

uint16\_t GetSensorSample(char\* str)

{

uint8\_t i,index=0;

uint16\_t len;

uint8\_t battery;

uint32\_t lux;

uint16\_t airT;

int16\_t airH;

battery = samples.fields[index++];

lux = (((uint32\_t)(samples.fields[index++])<<16)&0xffff00);

lux += samples.fields[index++];

airT = samples.fields[index++];

airH = (int16\_t)samples.fields[index++];

len = sprintf(str,"%s@%s:json:{",event\_id,device\_id);

len += sprintf(str+len,"\"%s\":%d,",fieldInfo[0].fieldName,battery); //电量

len += sprintf(str+len,"\"%s\":%g,",fieldInfo[1].fieldName,lux/100.0); //光照

len += sprintf(str+len,"\"%s\":%g,",fieldInfo[2].fieldName,airT/10.0); //空气温度

len += sprintf(str+len,"\"%s\":%g,",fieldInfo[3].fieldName,airH/10.0); //空气湿度

for (i=0;i<SOCKETS;i++)

{

uint8\_t fieldType = (samples.sockets[i]&0xf8)>>3;

uint8\_t fieldCounts = samples.sockets[i]&0x07;

uint8\_t fieldIndex = 0;

if (!fieldCounts) //485接口没有接传感器，跳过

continue;

len += sprintf(str+len,"\"Socket%d\":{",i);

do

{

float fieldData = 0;

switch(fieldInfo[fieldType+fieldIndex].type)

{

case U16:

fieldData = (int16\_t)samples.fields[index++];

break;

case S16:

fieldData = samples.fields[index++];

break;

case U16D1:

fieldData = (int16\_t)samples.fields[index++]/10.0;

break;

case S16D1:

fieldData = samples.fields[index++]/10.0;

break;

case U16D2:

fieldData = (int16\_t)samples.fields[index++]/100.0;

break;

case S16D2:

fieldData = samples.fields[index++]/100.0;

break;

case U32D2:

fieldData = (((uint32\_t)(samples.fields[index++])<<16)&0xffff00);

fieldData +=samples.fields[index++];

break;

}

len += sprintf(str+len,"\"%s\":%g,",fieldInfo[fieldType+fieldIndex].fieldName,fieldData);

}while(++fieldIndex<fieldCounts&&index<=samples.count);

str[len-1] = '}';

str[len++] = ',';

}

str[len-1] = '}'; //json结尾为}

for (i=0;i<SAMPLE\_SIZE;i++)

{

DEBUG(3,"%02x",((uint8\_t\*)&samples)[i]);

}

DEBUG(3,"%s\r\n",str);

return len;

}