

hexConfig

Инструкция по использованию

Назначение

- Конфигурирование прошивки под конкретное использование, путем поиска структуры конфигурации и изменения в ней требуемых параметров;
- Расчет контрольной характеристики прошивки (CRC16) и добавление ее в конец файла;
- Учет версий прошивок по выбранным характеристикам в структуре конфигуриатора;
- Преобразование входных файлов (bin или hex) в файл формата hex с длиной строки 16 байт.

Использование

Версии программы для linux и windows идентичны по параметрам запуска.

Для запуска под windows необходимо наличие библиотек `cygssc_s-1.dll`, `cygpropt-0.dll` и `cygwin1.dll`.

Параметры запуска

Вызов справки: **hexConfig.exe -?**

Пример:

```
user@user:~/user/hex.configurator$ ./hexConfig.exe -?
```

Использование: **hexConfig [-? -pvhlbrf --PARAM1=VALUE1 --PARAM2=VALUE2**

--PARAMn=VALUEn] FILES

-i, --input	Имя входного файла (*.hex *.bin)
-o, --output	Имя выходного файла (*.hex)
-p, --params	Показать поля КОНФИГУРАТОРА (требуется входной файл)
-h, --phelp	Показать справку hexConfig для доступа к полям (требуется входной файл)
-l, --crc-l-e	Добавить CRC в конец файла (формат: little-endian)
-b, --crc-b-e	Добавить CRC в конец файла (формат: big-endian)
-c, --crc-alg	Кодировка алгоритма CRC. По умолчанию: CRC16.L.0x1021.0xFFFF
-r, --crc-replace	Заменить CRC файла. (размер файла не меняется)
-f, --crc-force	Добавить CRC в конец файла игнорируя что CRC равен 0!
-v, --verbose	Режим вывода информационных сообщений
-d, --out-dir	Директория размещения результата и доп. файлов
FILES	Файлы для копирования в папку размещения результата.

Опции помощи:

-?, --help	Вывод данной справки
--usage	Вывод справки в сокращенном виде

Опции

--input=имя файла

Данная опция задает имя входного файла. Форматы входных файлов: .hex (HEX-intel), .bin (двоичный файл). Для Hex-файлов поддерживаются следующие типы записей:

- 00 (Data Record);
- 01 (End of File Record);
- 04 (Extended Linear Address Record);
- 05 (Start Linear Address Record).

Если входной hex-файл содержит пропуски в областях памяти, они заполняются значением 0xFF.

Двоичный файл должен иметь расширение .bin иначе файл не будет обработан.

--output=имя файла

Опция задает имя выходного файла. Независимо от указанного расширения, выходной файл имеет hex-формат с длиной строки данных 16 байт.

--params

Опция позволяет вывести список параметров, указанных в конфигурационной структуре файла прошивки.

--crc-l-e

Опция задает формат размещения результата контрольной характеристики в формате little-endian. Например при расчетном CRC 0xE23C конец файла выглядит следующим образом:

```
:100B800075A00075813D020AA67BFF7A0979AD8EBA  
:080B9000258F2602058D3CE2D1  
:00000001FF
```

--crc-b-e

Опция задает формат размещения результата контрольной характеристики в формате big-endian. Например при расчетном CRC 0xE23C конец файла выглядит следующим образом:

```
:100B800075A00075813D020AA67BFF7A0979AD8EBA  
:080B9000258F2602058DE23CD1  
:00000001FF
```

--crc-alg=параметры алгоритма

Опция задает параметры алгоритма расчета контрольной характеристики.

Для алгоритма CRC16 определены три параметра: направление обработки бит, основание и начальное значение. В общем виде параметры алгоритма задаются так:

CRC16.DIR.BASE.INIT, где

DIR — направление обработки бит в байте. Может принимать значения «L» — что означает что обработка происходит от левого (старшего) бита к правому (младшему), и «R» — от правого (младшего) бита к левому. «R» — соответствует отраженным алгоритмам CRC.

BASE — основание полинома CRC16 с учетом направления обработки. Внимание: для DIR = «R» нужно указывать отраженное значение полинома (см. [wiki:Cyclic_redundancy_check](https://en.wikipedia.org/wiki/Cyclic_redundancy_check)). Значение BASE непосредственно используется в операции XOR в алгоритме расчета CRC16.

INIT — начальное значение регистра CRC.

В таблице приводятся различные алгоритмы CRC16 и их соответствие опции --crc-alg

№	Полином	Обозначение алгоритма	Значение опции --crc-alg
1	CRC-16-ANSI/IBM 0x8005	CRC16-BUYPASS	--crc-alg=CRC16.L.0x8005.0x0000
2		CRC16-MODBUS	--crc-alg=CRC16.R.0xA001.0xFFFF
3	CRC16-CCITT 0x1021	CRC16-CCITT-FFFF*	--crc-alg=CRC16.L.0x1021.0xFFFF
4		CRC16-XMODEM	--crc-alg=CRC16.L.0x1021.0x0000
5		CRC16-KERMIT	--crc-alg=CRC16.R.0x8408.0x0000

*Примечание: CRC16-CCITT-FFFF используется по умолчанию.

--crc-replace

Опция указывает hexConfig на необходимость сократить область расчета CRC на 2 байта и размещать результат вместо этих двух байт. Для примера, приведем результат расчета по бинарному файлу содержащему строку «123456789» с параметром **--crc-replace** и без него:

hexConfig.exe -i 1.bin -o 1.hex -b -r	hexConfig.exe -i 1.bin -o 1.hex -b
:0900000031323334353637 7718FC :00000001FF	:0B000000313233343536373839 29B13E :00000001FF
По умолчанию используется алгоритм: --crc-alg=CRC16.L.0x1021.0xFFFF	

Если попытаться добавить в конец файла, который уже содержит CRC, новый результат расчета (при этом если использовался один и тот же алгоритм расчета контрольной характеристики и соответствующее ему размещение результата), результат повторного расчета CRC будет равен нулю. Для корректного расчета контрольной характеристики файла, при первичном расчете CRC, опция **--crc-replace** должна отсутствовать, а при изменении полей конфигулятора, в файле уже содержащем CRC, опция **--crc-replace** должна быть установлена.

--out-dir=выражение (определяет путь назначения)

Изменение параметров в структуре конфигулятора фактически приводит к появлению новой версии прошивки, функционирование которой отличается каким-либо параметром от оригинальной прошивки. В этом случае может потребоваться сохранить файл прошивки под другим именем, или в отдельную директорию, имя которой, будет отражать особенность полученной прошивки.

Для этих целей предусмотрена опция --out-dir (-d), позволяющая задавать имя директории назначения с учетом значений полей конфигулятора. В общем виде значение этой опции можно представить как: --out-dir=«TEXT»+PARAMNAME1+«TEXT»+...+PARAMNAME_n

(Важно: Строка не должна содержать пробелов вне кавычек).

При работе программы значение этой опции будет преобразовано в имя директории, где вместо PARAMNAME_i будут их текстовые значения.

Если предполагается что путь назначения состоит из нескольких директорий, имена директорий должен разделять знак-разделитель «/» (как для linux, так и в версии под windows). Если директории назначения не существует – она будет создана.

Например: --out-dir=PARAMNAME1+«/»+PARAMNAME2+«/»+PARAMNAME3

В данном случае будет создана директория, имя которой задает PARAMNAME1, в ней директория, имя которой определяет PARAMNAME2, и уже в ней, директория, имя которой задает PARAMNAME3, в которую и будет размещен полученный файл.

Часто, бывает необходимо вместе с файлом прошивки положить какой-нибудь вспомогательный файл (например файл помощи, справки, или файл описания изменений прошивки). Для этих целей в hexConfig предусмотрены **FILES** – это перечень имен файлов, разделенных пробелами, которые нужно копировать, размещая вместе с полученным файлом прошивки.

Пример ниже демонстрирует, как на основе базовой прошивки, формируется прошивка со скоростью работы UART 9600 бод, при этом, полученная прошивка размещается в директорию, имя которой будет содержать версию прошивки и дату ее сборки компилятором, разделенных пробелом:

```
hexConfig.exe -i inputFile.hex -o outputFile.hex --BAUD=9600
--out-dir=VERSION+« »+DATE
```

В результате была создана директория «V1.101 Aug 05 2016» и в нее размещен файл outputFile.hex.

Для наглядности приведем дамп структуры конфигулятора в файле прошивке inputFile.hex (начало с адреса 0x1EF3):

00001EE0 56 31 2E 31	30 31 00 41	75 67 20 30	35 20 32 30 V1.101.Aug 05 20
00001EF0 31 36 00 43	4F 4E 46 49	47 55 52 41	54 4F 52 00 16.CONFIGURATOR.
00001F00 07 01 03 56	45 52 53 49	4F 4E 00 00	00 00 04 FF ...VERSION.....
00001F10 1E E0 00 44	41 54 45 00	00 00 00 00	00 00 04 FF ...DATE.....
00001F20 1E E7 00 46	4C 00 00 00	00 00 00 00	00 00 01 3F ...FL.....?
00001F30 C8 F5 C3 4D	48 5A 00 00	00 00 00 00	00 00 00 00 ...MHZ.....
00001F40 7A 12 00 42	41 55 44 00	00 00 00 00	00 00 00 00 z..BAUD.....
00001F50 01 C2 00 44	45 50 49 4E	00 00 00 00	00 00 00 00 ...DEPIN.....

--crc-verbose

Опция разрешает программе выводить на экран информационные сообщения о ходе выполнения программы.

FILES

Список дополнительных файлов, которые нужно скопировать в директорию назначения. Используется совместно с опцией --out-dir.

Внедрение структуры конфигулятора в программу

Для того что бы пользоваться возможностями hexConfig для изменения параметров прошивки, требуется внедрить в программу структурную константу формата **TConfigurator** (описана в «Configurator.h»):

```
typedef struct {
    char ConfiguratorID[13];
    unsigned char fieldCount;
    unsigned char fieldFormat;
    unsigned char sizeofPointer;
    TConfiguratorField Fields[CONF_FD_COUNT];
} TConfigurator;
```

,где TConfiguratorField определен как:

```
typedef struct {
    char fieldID[11];
    unsigned char fieldType;
    union
    {
        unsigned int    iValue;
        char *pChar;
        unsigned char   bValue;
        unsigned short  wValue;
        float            fValue;
        unsigned char   b[4];
    } Values;
} TConfiguratorField;
```

Разумеется, значения полей этой структуры должны использоваться программой для своей работы и определять ее поведение.

Внимание: Структуры должны быть упакованы (используйте `#pragma pack(push, 1)` / `#pragma pack(pop)` или атрибут `__attribute__((packed))` при определении типов структур). Так как разрядность типов `int` / `short` / `int` может отличаться от 32/16 соответственно, используйте соответствующие типы (например `long` вместо `int`, для 8-и разрядных архитектур).

Ниже приведен пример программы, демонстрирующий использование структуры конфигулятора.

```
// Пример1. Keil. ARM. Target: 1986BE91T (Cortex-M3)
#include <stdio.h>
#define CONF_FD_COUNT 7
#include "..\Configurator.h"

const TConfigurator Configurator = {CONFIGURATOR_KEY, CONF_FD_COUNT, CONF_FORMAT_LE,
CONF_SIZEOFPOINTER,
    { // C/C++: [C99 Mode]
        {"VERSION", CONF_FD_PCHAR, {pChar = "V1.101"}}, //0:
        {"DATE", CONF_FD_PCHAR, {pChar = __DATE__}}, //1:
        {"BAUD", CONF_FD_INT32, 115200}, //2:
        {"DE_PIN", CONF_FD_BYTE, {bValue = 10}}, //3:
        {"ADC_K0", CONF_FD_FLOAT, {fValue = 0}}, //4:
        {"ADC_K1", CONF_FD_FLOAT, {fValue = 0.0196}}, //5:
        {"PORT", CONF_FD_INT16, {wValue = 0x55AA}} //6:
    }
};

void InitUART(int baud)
{
    printf("baud=%d\n", baud);
}

int main (void)
{
    InitUART(Configurator.Fields[2].Values.iValue);
    printf("VERSION.... %s\n", Configurator.Fields[0].Values.pChar);
    printf("DATE..... %s\n", Configurator.Fields[1].Values.pChar);
    printf("BAUD..... %d\n", Configurator.Fields[2].Values.iValue);
    printf("DE_PIN.... %d\n", Configurator.Fields[3].Values.bValue);
    printf("ADC_K0..... %f\n", Configurator.Fields[4].Values.fValue);
    printf("ADC_K1..... %f\n", Configurator.Fields[5].Values.fValue);
    printf("PORT..... %x\n", Configurator.Fields[6].Values.wValue);
    while (1);
}

/*Для удобства доступа к значениям параметров может использоваться функции типа:
unsigned int getConfiguratorValue(const char * fieldName, unsigned int defaultValue)
{
    unsigned char i;
    for (i = 0; i < Configurator.fieldCount; i++)
        if (strcmp (Configurator.Fields[i].fieldID, fieldName) == 0)
            return Configurator.Fields[i].Values.iValue;
    return defaultValue;
}*/
```

Пояснения к примеру:

Перед подключением файла `"..\Configurator.h"` определяется символ `CONF_FD_COUNT`, который задает количество параметров в структуре конфигулятора. Далее объявляется структура конфигулятора (константа) и задаются значения параметров по умолчанию. В заголовке конфигулятора, важную роль играет поле `fieldFormat`, которое определяет формат хранения данных (`little-endian/big-endian`), оно требуется для корректной работы `hexConfig`. Этому полю

нужно присвоить значение соответствующее используемому формату: CONF_FORMAT_LE для little-endian и CONF_FORMAT_BE для big-endian. При описании параметров задается их имена (длина не более 10 символов), идентификатор типа:

Обозначение типа	Тип значения	Диапазон значений
CONF_FD_INT32	uint32_t	0..0xFFFFFFFF
CONF_FD_FLOAT	float	±3.4E38
CONF_FD_INT16	uint16_t	0..0xFFFF
CONF_FD_BYTE	unsigned char	0..0xFF
CONF_FD_PCHAR	char *	Не должен превышать разрядность 32 бит.

Для задания значения по умолчанию с учетом требуемого типа используется имя поля структуры. Если ваш компилятор не поддерживает такую инициализацию (например как keil51) в файле "Configurator.h" описаны отдельные контейнеры для каждого типа (TCField_i32, TCField_float, TCField_wInt, TCField_i8, TCField_pChar), которые можно использовать, как показано во втором примере.

При компиляции данного примера в полученном файле прошивки наблюдается структура конфигулятора следующего вида:

0000000FA0: 00 00 00 00 43 4F 4E 46	49 47 55 52 41 54 4F 52	CONFIGURATOR
0000000FB0: 00 07 00 04 56 45 52 53	49 4F 4E 00 00 00 00 04	• ◆VERSION ◆
0000000FC0: 30 10 00 08 44 41 54 45	00 00 00 00 00 00 00 04	0► □DATE ◆
0000000FD0: 38 10 00 08 42 41 55 44	00 00 00 00 00 00 00 00	8► □BAUD
0000000FE0: 00 C2 01 00 44 45 5F 50	49 4E 00 00 00 00 00 03	B⊙ DE_PIN ▼
0000000FF0: 0A 00 00 00 41 44 43 5F	4B 30 00 00 00 00 00 01	■ ADC_K0 ☺
0000001000: 00 00 00 00 41 44 43 5F	4B 31 00 00 00 00 00 01	ADC_K1 ☺
0000001010: 2E 90 A0 3C 50 4F 52 54	00 00 00 00 00 00 00 02	.? <PORT ●
0000001020: AA 55 00 00 3A 74 74 00	3A 74 74 00 3A 74 74 00	€U :tt :tt :tt
0000001030: 56 31 2E 31 30 31 00 00	41 75 67 20 31 31 20 32	V1.101 Aug 11 2
0000001040: 30 31 36 00 64 10 00 08	00 00 00 20 10 00 00 00	016 d► □ ►

// Пример2. Keil. x51. Target: Aduc845

```
#include <stdio.h>
```

```
#define CONF_FD_COUNT 7
```

```
#include "..\Configurator.h"
```

```
typedef struct {
```

```
    TConfiguratorHead Head;
```

```
    TCField_pChar VERSION;
```

```
    TCField_pChar DATE;
```

```
    TCField_i32 BAUD;
```

```
    TCField_i8 DE_PIN;
```

```
    TCField_float ADC_K0;
```

```
    TCField_float ADC_K1;
```

```
    TCField_wInt PORT;
```

```
} TMyConfigurator;
```

```
code const TMyConfigurator Configurator = {
```

```
    {CONFIGURATOR_KEY, CONF_FD_COUNT, CONF_FORMAT_BE, CONF_SIZEOFPOINTER},
```

```
    {"VERSION", CONF_FD_PCHAR, "V1.101"}, //0:
```

```
    {"DATE", CONF_FD_PCHAR, __DATE__}, //1:
```

```
    {"BAUD", CONF_FD_INT32, 115200}, //2:
```

```
    {"DE_PIN", CONF_FD_BYTE, 10}, //3:
```

```
    {"ADC_K0", CONF_FD_FLOAT, 0}, //4:
```

```
    {"ADC_K1", CONF_FD_FLOAT, 0.0196}, //5:
```

```
    {"PORT", CONF_FD_INT16, 0x55AA} //6:
```

```
};
```

```

void InitUART(int baud)
{
    printf("baud=%d\n", baud);
}

int main (void)
{
    InitUART(Configurator.BAUD.Values.iValue);
    printf("VERSION.... %s\n", Configurator.VERSION.Values.pChar);
    printf("DATE..... %s\n", Configurator.DATE.Values.pChar);
    printf("BAUD..... %d\n", Configurator.BAUD.Values.iValue);
    printf("DE_PIN..... %d\n", Configurator.DE_PIN.Values.bValue);
    printf("ADC_K0..... %f\n", Configurator.ADC_K0.Values.fValue);
    printf("ADC_K1..... %f\n", Configurator.ADC_K1.Values.fValue);
    printf("PORT..... %x\n", Configurator.PORT.Values.wValue);
    while (1);
}

```

Пояснения к примеру:

В данном примере компилятор не поддерживает стандарт C99, поэтому для инициализации значений по умолчанию (с учетом требуемого типа) пришлось объявить свой тип **TMyConfigurator**, в который входит служебный заголовок (тип TconfiguratorHead) и требуемые параметры. В данном примере использование отдельных типизированных контейнеров для каждого параметра положительно повлияло на читаемость кода при использовании параметров.

При компиляции получаем структуру конфигулятора следующего вида:

00001EE0 56 31 2E 31	30 31 00 41	75 67 20 30	35 20 32 30 V1.101.Aug 05 20
00001EF0 31 36 00 43	4F 4E 46 49	47 55 52 41	54 4F 52 00 16. CONFIGURATOR.
00001F00 07 01 03 56	45 52 53 49	4F 4E 00 00	00 00 04 FF ...VERSION.....
00001F10 1E E0 00 44	41 54 45 00	00 00 00 00	00 00 04 FF ...DATE.....
...			

Примеры команд

В качестве входного файла возьмем файл прошивки полученный при компиляции примера №1.

Просмотр списка параметров конфигулятора в файле h.hex

```
D:\...ev\hex.configurator>hexConfig.exe -i h.hex -p

Входной файл : 'h.hex' -> Файл результата : 'out.hex'
Вход> CONFIGURATOR [Кол-во полей: 7]. Формат хранения : little-endian
VERSION - STRING [значение: V1.101]
DATE - STRING [значение: Aug 11 2016]
BAUD - INT32 [значение: 115200]
DE_PIN - BYTE [значение: 10]
ADC_K0 - FLOAT [значение: 0.000000]
ADC_K1 - FLOAT [значение: 0.019600]
PORT - INT16 [значение: 21930]
```

Вызов справки для доступа к параметрам

```
D:\...ev\hex.configurator>hexConfig.exe -i h.hex -h
Входной файл : 'h.hex' -> Файл результата : 'out.hex'
Usage: hexConfig [-] [--VERSION=STRING] [--DATE=STRING] [--BAUD=INT]
           [--DE_PIN=INT] [--ADC_K0=FLOAT] [--ADC_K1=FLOAT] [--PORT=INT] ...
```

Изменение параметра BAUD и защита файла CRC16 (режим вывода сообщений)

```
D:\...\hex.configurator>hexConfig.exe -i h.hex -o hout.hex -b -v --BAUD=9600
Входной файл : 'h.hex' -> Файл результата : 'hout.hex'

Вход> CONFIGURATOR [Кол-во полей: 7]. Формат хранения : little-endian
VERSION - STRING [значение: V1.101]
DATE - STRING [значение: Aug 11 2016]
BAUD - INT32 [значение: 115200]
DE_PIN - BYTE [значение: 10]
ADC_K0 - FLOAT [значение: 0.000000]
ADC_K1 - FLOAT [значение: 0.019600]
PORT - INT16 [значение: 21930]

Кол-во измененных полей: 1
Выход> CONFIGURATOR [Кол-во полей: 7]. Формат хранения : little-endian
VERSION - STRING [значение: V1.101]
DATE - STRING [значение: Aug 11 2016]
BAUD - INT32 [значение: 9600]
DE_PIN - BYTE [значение: 10]
ADC_K0 - FLOAT [значение: 0.000000]
ADC_K1 - FLOAT [значение: 0.019600]
PORT - INT16 [значение: 21930]

Результат вычисления CRC: 0x6866 (алгоритм: CRC16.L.0x1021.0xFFFF)
Размер файла: 8788
Запись файла 'hout.hex'...
```

Посмотрим как разместилось CRC

h.hex	hout.hex
:042250000000000008A	:062250000000000006866BA
:040000050800000C12E	:040000050800000C12E
:00000001FF	:00000001FF

Попытка повторно защитить файл контрольной характеристикой

```
D:\...ev\hex.configurator>hexConfig.exe -i hout.hex -o hout1.hex -b
Входной файл : 'hout.hex' -> Файл результата : 'hout1.hex'
```

Результат вычисления CRC равен 0! Файл уже содержит CRC код. (используйте -r)
Для принудительной записи 0 в конец файла используйте --crc-force или -f

Сохранение данной версии файла прошивки в директории

```
D:\>hexConfig.exe -i h.hex -o hout0.hex --out-dir=VERSION+" "+DATE makefile -v
Входной файл : 'h.hex' -> Файл результата : 'hout0.hex'
```

```
Вход> CONFIGURATOR [Кол-во полей: 7]. Формат хранения : little-endian
VERSION - STRING [значение: V1.101]
DATE - STRING [значение: Aug 11 2016]
BAUD - INT32 [значение: 115200]
DE_PIN - BYTE [значение: 10]
ADC_K0 - FLOAT [значение: 0.000000]
ADC_K1 - FLOAT [значение: 0.019600]
PORT - INT16 [значение: 21930]
```

Кол-во измененных полей: 0

```
Выход> CONFIGURATOR [Кол-во полей: 7]. Формат хранения : little-endian
VERSION - STRING [значение: V1.101]
DATE - STRING [значение: Aug 11 2016]
BAUD - INT32 [значение: 115200]
DE_PIN - BYTE [значение: 10]
ADC_K0 - FLOAT [значение: 0.000000]
ADC_K1 - FLOAT [значение: 0.019600]
PORT - INT16 [значение: 21930]
```

Скопированно файлов: 1

Запись файла 'V1.101 Aug 11 2016/hout0.hex'...

Результат:

```
D:\....configurator\V1.101 Aug 11 2016>dir
hout0.hex  makefile
```

Приложение. Файл Configurator.h

```
/*
 * File:    Configurator.h
 * Author:  Dunaev
 *
 */

#ifndef CONFIGURATOR_H
#define CONFIGURATOR_H

#pragma pack(push, 1)

#define CONF_FD_INT32 0
#define CONF_FD_FLOAT 1
#define CONF_FD_INT16 2
#define CONF_FD_BYTE 3
#define CONF_FD_PCHAR 4
// #define CONF_FD_PDOUBLE 5 (unsupported)

#define CONF_FORMAT_LE 0
#define CONF_FORMAT_BE 1
#define CONF_SIZEOFPOINTER sizeof(char *)

#define CONFIGURATOR_KEY "CONFIGURATOR"

typedef struct {
    char fieldID[11];
    unsigned char fieldType;
    union
    {
        unsigned int iValue;
        char *pChar;
        unsigned char bValue;
        unsigned short wValue;
        float fValue;
        unsigned char b[4];
    } Values;
} TConfiguratorField;

#ifndef CONF_FD_COUNT
#define CONF_FD_COUNT 10
#endif

typedef struct {
    char ConfiguratorID[13];
    unsigned char fieldCount;
    unsigned char fieldFormat;
    unsigned char sizeOfPointer;
    TConfiguratorField Fields[CONF_FD_COUNT];
} TConfigurator;

//-----
typedef struct {
    char fieldID[11];
    unsigned char fieldType;
    union
    {
        long iValue;
```

```

        unsigned char b[4];
    } Values;
} TCField_i32;

typedef struct {
    char fieldID[11];
    unsigned char fieldType;
    union
    {
        unsigned char bValue;
        unsigned char b[4];
    } Values;
} TCField_i8;

typedef struct {
    char fieldID[11];
    unsigned char fieldType;
    union
    {
        float fValue;
        unsigned char b[4];
    } Values;
} TCField_float;

typedef struct {
    char fieldID[11];
    unsigned char fieldType;
    union
    {
        char *pChar;
        unsigned char b[4];
    } Values;
} TCField_pChar;

typedef struct {
    char fieldID[11];
    unsigned char fieldType;
    union
    {
        unsigned short wValue;
        unsigned char b[4];
    } Values;
} TCField_wInt;

typedef struct {
    char ConfiguratorID[13];
    unsigned char fieldCount;
    unsigned char fieldFormat;
    unsigned char sizeofPointer;
} TConfiguratorHead;

#pragma pack(pop)
#endif      /* CONFIGURATOR_H */

```