Библиотека доступа к БД (DBAccessLib) для платформы MCU-X/Y

ФБ чтения/записи подындексов через массив

ФБ для чтения из БД через массив

Входные данные (Inputs)

Имя переменной	Тип данных	Описание
`Key`	*STRING*	Ключ к индексу в БД
`Subi_Start`	*USINT*	Начальный подындекс
`Subi_Count`	*USINT*	Количество подындексов и элементов массива
`Arrays_Start`	*USINT*	Начальный элемент массива **данных и ошибок**. С этого элемента массива будут находиться прочитанные данные и записаны ошибки доступа

- Значения `Subi_Start` , `Subi_Count` и `Array_Start` должны находиться в диапазоне от 1 до 255
- Значения (`Subi_Start` + `Subi_Count`) и (`Array_Start` + `Subi_Count`) не должны превышать 255

Выходные данные (Outputs)

Имя переменной	Тип данных	Описание
`Data_Array`	*ARRAY[1..255] OF <TYPE>*	Массив данных (255 элементов). Данные будут находиться начиная с `Array_Start` элемента
`Error`	*UDINT*	Код ошибки доступа
`Error_Array`	*ARRAY[1..255] OF UDINT*	Массив с ошибками доступа

ФБ для записи в БД через массив

Входные данные (Inputs)

Имя переменной	Тип данных	Описание
`Key`	*STRING*	Ключ к индексу в БД
`Subi_Start`	*USINT*	Начальный подындекс.
`Subi_Count`	*USINT*	Количество подындексов

Имя переменной	Тип данных	Описание
`Arrays_Start`	*USINT*	Начальный элемент массива **данных и ошибок**. С этого элемента массивов будут записаны данные и находиться коды ошибок
`Data_Array`	*ARRAY[1..255] OF <TYPE>*	Массив данных (255 элементов) для записи в БД. Данные будут записаны с `Array_Start` элемента

- Значения `Subi_Start` , `Subi_Count` и `Array_Start` должны находиться в диапазоне от 1 до 255
- Значения (`Subi_Start` + `Subi_Count`) и (`Array_Start` + `Subi_Count`) не должны превышать 255

Выходные данные (Outputs)

Имя переменной	Тип данных	Описание
`Error`	*UDINT*	Код ошибки доступа
`Error_Array`	*ARRAY[1..255] OF UDINT*	Массив с ошибками доступа

Параметр `Arrays_Start` позволяет управлять индексацией элементов массива данными без привязки к начальному подындексу (`Subi_Start`). Например, при чтении 10 подындексов начиная с 20-го и задав `Arrays_Start` равный 1, получим что данные и ошибки будут находиться с 1-го элемента массива.

Обработка ошибок

Раздел в разработке

Список ФБ

Функции чтения и записи строк ("STRING) не поддерживаются

Set функции (запись массива данных)

- `setDBEntriesBOOL`
- `setDBEntriesSINT`
- `setDBEntriesINT`
- `setDBEntriesDINT`
- `setDBEntriesLINT`
- `setDBEntriesUSINT`
- `setDBEntriesUINT`
- `setDBEntriesUDINT`
- `setDBEntriesULINT`
- `setDBEntriesREAL`
- `setDBEntriesLREAL`
- `setDBEntriesBYTE` - для доступа к подындексам типа USINT
- `setDBEntriesWORD` - для доступа к подындексам типа UINT
- `setDBEntriesDWORD` - для доступа к подындексам типа UDINT

- `setDBEntriesLWORD` - для доступа к подындексам типа ULINT

Get функции (чтение массива данных)

- `getDBEntriesBOOL`
- `getDBEntriesSINT`
- `getDBEntriesINT`
- `getDBEntriesDINT`
- `getDBEntriesLINT`
- `getDBEntriesUSINT`
- `getDBEntriesUINT`
- `getDBEntriesUDINT`
- `getDBEntriesULINT`
- `getDBEntriesREAL`
- `getDBEntriesLREAL`
- `getDBEntriesBYTE` - для доступа к подындексам типа USINT
- `getDBEntriesWORD` - для доступа к подындексам типа UINT
- `getDBEntriesDWORD` - для доступа к подындексам типа UDINT
- `getDBEntriesLWORD` - для доступа к подындексам типа ULINT

Пример

```
(* Запись в индекс "key_set" "count" начиная с "start" подындекса *)
start := 3; (* Пишем начиная с 3 подындекса *)
count := 4; (* Пишем всего 4 подындекса *)

(* Заполняем массив для записи. Смещение в массиве соответствует смещению подындекса *)
setDBEntriesBOOL.data_array[ start ] = TRUE;
...
setDBEntriesBOOL.data_array[ start + i ] = TRUE;

(* Вызываем функцию *)
setDBEntriesBOOL(key := index_key, subi_start := start, subi_count := count, array_start := start);
(* Есть ошибки доступа? *)
if setDBEntriesBOOL.error > 0 then
    for i := start to (start + count -1) do
        if setDBEntriesBOOL.error_array[i] <> 0 then
            (* access error handler *)
        end_if;
end_if;
```

Реализация на уровне библиотеки

- Перед чтением или записью в массиве ошибок (`Error_Array`) обнуляются `Subi_Count` элементов начиная с `Arrays_Start`
- Значения элементов массива (`Data_Array` и `Error_Array`) не попадающие в диапазон `Array_Start` .. (`Array_Start` + `Subi_Count` -1) в ФБ не изменяются

- Значения `Subi_Start` , `Subi_Count` и `Array_Start` должны находиться в диапазоне от 1 до 255
- Значения (`Subi_Start` + `Subi_Count`) и (`Array_Start` + `Subi_Count`) не должны превышать 255
- Массив данных в ФБ имеет фиксированную длину 255. Индексация массива с 1
- Массив ошибок в ФБ имеет фиксированную длину 255. Индексация массива с 1
- Из ФБ в функцию обёртки указатель на данные передаются с адресом начала массива, т.е. без смещения на `Subi_Start`
- Из ФБ в функцию обёртки указатель на ошибки передаются с адресом начала массива, т.е. без смещения на `Array_Offset`
- Минимальное значение начального подындекса: 1
- Максимальное кол-во подындексов: 255

Пользовательские ФБ для доступа к БД через структуру

Раздел посвящён алгоритму разработки пользовательских ФБ для доступа к индексам БД через структуры.

Для доступа к БД через структуру используется функции `getDBEntryStruct` и `setDBEntryStruct`. См. приложение А.

В функции необходимо передать:

- Ключ - путь к индексу (строка)
- Массив с номерами подындексов
- Кол-во подындексов
- Массив кодов типов для каждого подындекса
- Массив с сериализованными данными
- Массив для чтения ошибок доступа

Функции возвращают код ошибки доступа.

Массив с номерами подындексов представляет собой массив USINT. См. `SUBI_NUMBER_ARR : ARRAY [1..11] OF USINT;` в примере.

Массив кодов типов для каждого подындекса представляет собой массив USINT. См. `SUBI_TYPE_ARR : ARRAY [1..11] OF USINT;` в примере

ФБ сериализации данных

Для типов *LINT*, *ULINT*, *REAL32* (*REAL*) и *REAL64* (*LREAL*) необходима сериализация при записи в массив

- `LINT_SRLZ_TO_UDINT_PAIR(LINT:IN) => (UDINT:OUT_1, UDINT:OUT_2)`
- `ULINT_SRLZ_TO_UDINT_PAIR(ULINT:IN) => (UDINT:OUT_1, UDINT:OUT_2)`
- `LREAL_SRLZ_TO_UDINT_PAIR(LREAL:IN) => (UDINT:OUT_1, UDINT:OUT_2)`
- `REAL_SRLZ_TO_UDINT(REAL:IN) => (UDINT:OUT)`

И десериализация при чтении из массива данных:

- `UDINT_PAIR_DESRLZ_TO_LINT(UDINT:IN_1, UDINT:IN_2) => (LINT:OUT)`
- `UDINT_PAIR_DESRLZ_TO_ULINT(UDINT:IN_1, UDINT:IN_2) => (ULINT:OUT)`

- `UDINT_PAIR_DESRLZ_TO_LREAL(UDINT:IN_1, UDINT:IN_2) => (LREAL:OUT)`
- `UDINT_DESRLZ_TO_REAL(UDINT:IN) => (REAL:OUT)`

Для функций с парой параметров `UDINT`, первый параметр необходимо записать/прочитать в/из массива сериализованных данных с меньшим номером элемента.

Пример сериализации `ULINT`:

```
ULINT_SRLZ_TO_UDINT( IN := VAL.ULINT_VAL,
                     OUT_1 => DB_ARR_SEND_RCV[10],
                     OUT_2 => DB_ARR_SEND_RCV[11]);
```

Пример десериализации в `ULINT`:

```
UDINT_SRLZ_ULINT(IN_1 := DB_ARR_SEND_RCV[10],
                 IN_2 := DB_ARR_SEND_RCV[11],
                 OUT => OUT.ulint_val);
```

Для всех остальных типов можно использовать стандартные функции преобразования в/из *UDINT*.

Функция получения кода типа

`geyDBAccessTypeCode(DBTypes:DBAccessType) => (USINT:OUT)`

Пример:

`SUBS_TYPE[9]  := getDBTypeCode(TYPE_ULINT);`

Примеры ФБ записи и чтения в БД через структуру

Ниже представлен пример ФБ записи и чтения структуры словаря:

- Подындекс 1 - Тип `BOOL`
- Подындекс 2 - Тип `SINT`
- Подындекс 3 - Тип `INT`
- Подындекс 4 - Тип `DINT`
- Подындекс 5 - Тип `LINT`
- Подындекс 6 - Тип `USINT`
- Подындекс 7 - Тип `UINT`
- Подындекс 8 - Тип `UDINT`
- Подындекс 9 - Тип `ULINT`
- Подындекс 10 - Тип `REAL`
- Подындекс 11 - Тип `LREAL`

Структура в коде ПЛК (`user_type`) содержит поля: `<тип подындекса>_val`

Примеры ФБ записи в БД через структуру

```
FUNCTION_BLOCK set_db_struct
  VAR_INPUT
    Key : STRING;
    IN : user_type;
  END_VAR
  VAR_OUTPUT
    ERROR : UDINT;
    ERROR_ARR : ARRAY [1..11] OF UDINT;
  END_VAR
  VAR
    DB_ARR_SEND_RCV : ARRAY [1..14] OF UDINT;
    SUBS_TYPE : ARRAY [1..11] OF USINT;
    SUBS_NUMB : ARRAY [1..11] OF USINT;
    once_init : BOOL := False;
    ulint_srlz : ULINT_SRLZ_TO_UDINT_PAIR;
    lint_srlz : LINT_SRLZ_TO_UDINT_PAIR;
    real_srlz : REAL_SRLZ_TO_UDINT;
    lreal_srlz : LREAL_SRLZ_TO_UDINT_PAIR;
  END_VAR

(* Functional Block Code Start *)
  if once_init <> True then
    once_init := True;
    (* One-time initialization Start *)

    (* Types of subindexes Start *)
    SUBS_TYPE[1]  := getDBTypeCode(TYPE_BOOL);
    SUBS_TYPE[2]  := getDBTypeCode(TYPE_SINT);
    SUBS_TYPE[3]  := getDBTypeCode(TYPE_INT);
    SUBS_TYPE[4]  := getDBTypeCode(TYPE_DINT);
    SUBS_TYPE[5]  := getDBTypeCode(TYPE_LINT);
    SUBS_TYPE[6]  := getDBTypeCode(TYPE_USINT);
    SUBS_TYPE[7]  := getDBTypeCode(TYPE_UINT);
    SUBS_TYPE[8]  := getDBTypeCode(TYPE_UDINT);
    SUBS_TYPE[9]  := getDBTypeCode(TYPE_ULINT);
    SUBS_TYPE[10] := getDBTypeCode(TYPE_REAL);
    SUBS_TYPE[11] := getDBTypeCode(TYPE_LREAL);
    (* Types of subindexes End *)

    (* Subindex numbers Start *)
    SUBS_NUMB[1] := 1;
    SUBS_NUMB[2] := 2;
    SUBS_NUMB[3] := 3;
    SUBS_NUMB[4] := 4;
    SUBS_NUMB[5] := 5;
    SUBS_NUMB[6] := 6;
    SUBS_NUMB[7] := 7;
    SUBS_NUMB[8] := 8;
    SUBS_NUMB[9] := 9;
    SUBS_NUMB[10] := 10;
    SUBS_NUMB[11] := 11;
    (* Subindex numbers End *)

    (* One-time initialization End *)
  end_if;
```

```
56
57    (* Data Serialization Start *)
58
59    DB_ARR_SEND_RCV[1] := BOOL_TO_UDINT( IN.bool_val); (* Serialization of BOOL *)
60
61    DB_ARR_SEND_RCV[2] := SINT_TO_UDINT( IN.sint_val); (* Serialization of SINT *)
62    DB_ARR_SEND_RCV[3] := INT_TO_UDINT(  IN.int_val);  (* Serialization of INT *)
63    DB_ARR_SEND_RCV[4] := DINT_TO_UDINT( IN.dint_val); (* Serialization of DINT *)
64    lint_srlz(IN := IN.lint_val, OUT_1 => DB_ARR_SEND_RCV[5], OUT_2 => DB_ARR_SEND_RCV[6]);
(* Serialization of LINT *)
65
66    DB_ARR_SEND_RCV[7] := USINT_TO_UDINT(IN.usint_val); (* Serialization of USINT *)
67    DB_ARR_SEND_RCV[8] := UINT_TO_UDINT(IN.uint_val); (* Serialization of UDINT *)
68    DB_ARR_SEND_RCV[9] := IN.udint_val; (* Serialization of UDINT *)
69    ulint_srlz(IN := IN.ulint_val, OUT_1 => DB_ARR_SEND_RCV[10], OUT_2 =>
DB_ARR_SEND_RCV[11]); (* Serialization of ULINT *)
70
71    real_srlz(IN := IN.real_val, OUT => DB_ARR_SEND_RCV[12]); (* Serialization of REAL
(REAL32) *)
72    lreal_srlz(IN := IN.lreal_val, OUT_1 => DB_ARR_SEND_RCV[13], OUT_2 =>
DB_ARR_SEND_RCV[14]); (* Serialization of LREAL (REAL64) *)
73
74    (* Data Serialization End *)
75
76    (* Calling write function to DB Start *)
77    {{ uint32_t res =  setDBEntryStruct(GetFbVar(KEY),
78                                        GET_FB_ARRAY_POINTER(SUBS_NUMB), /* Number of subi
Start */ 11 /* Number of subi End */,
79                                        GET_FB_ARRAY_POINTER(SUBS_TYPE),
80                                        GET_FB_ARRAY_POINTER(DB_ARR_SEND_RCV),
81                                        GET_FB_ARRAY_POINTER(ERROR_ARR) ); }}
82    (* Calling write function to DB End *)
83
84    {{ SetFbVar(ERROR, res); }}
85  (* Functional Block Code End *)
86  END_FUNCTION_BLOCK
```

Пример ФБ чтения в БД через структуру

```
1   FUNCTION_BLOCK get_db_struct
2     VAR_INPUT
3       Key : STRING;
4     END_VAR
5     VAR_OUTPUT
6       OUT : user_type;
7       ERROR : UDINT;
8       ERROR_ARR : ARRAY [1..11] OF UDINT;
9     END_VAR
10    VAR
11      DB_ARR_SEND_RCV : ARRAY [1..14] OF UDINT;
12      SUBS_TYPE : ARRAY [1..11] OF USINT;
13      SUBS_NUMB : ARRAY [1..11] OF USINT;
14      once_init : BOOL := False;
15      UDINT_SRLZ_LINT : UDINT_PAIR_DESRLZ_TO_LINT;
16      UDINT_SRLZ_ULINT : UDINT_PAIR_DESRLZ_TO_ULINT;
```

```
    UDINT_DESRLZ_REAL : UDINT_DESRLZ_TO_REAL;
    UDINT_SRLZ_LREAL : UDINT_PAIR_DESRLZ_TO_LREAL;
  END_VAR

  (* Functional Block Code Start *)
  if once_init <> True then
    once_init := True;
    (* One-time initialization Start *)

    (* Types of subindexes Start *)
    SUBS_TYPE[1]  := getDBTypeCode(TYPE_BOOL);
    SUBS_TYPE[2]  := getDBTypeCode(TYPE_SINT);
    SUBS_TYPE[3]  := getDBTypeCode(TYPE_INT);
    SUBS_TYPE[4]  := getDBTypeCode(TYPE_DINT);
    SUBS_TYPE[5]  := getDBTypeCode(TYPE_LINT);
    SUBS_TYPE[6]  := getDBTypeCode(TYPE_USINT);
    SUBS_TYPE[7]  := getDBTypeCode(TYPE_UINT);
    SUBS_TYPE[8]  := getDBTypeCode(TYPE_UDINT);
    SUBS_TYPE[9]  := getDBTypeCode(TYPE_ULINT);
    SUBS_TYPE[10] := getDBTypeCode(TYPE_REAL);
    SUBS_TYPE[11] := getDBTypeCode(TYPE_LREAL);
    (* Types of subindexes End *)

    (* Subindex numbers Start *)
    SUBS_NUMB[1] := 1;
    SUBS_NUMB[2] := 2;
    SUBS_NUMB[3] := 3;
    SUBS_NUMB[4] := 4;
    SUBS_NUMB[5] := 5;
    SUBS_NUMB[6] := 6;
    SUBS_NUMB[7] := 7;
    SUBS_NUMB[8] := 8;
    SUBS_NUMB[9] := 9;
    SUBS_NUMB[10] := 10;
    SUBS_NUMB[11] := 11;
    (* Subindex numbers End *)

    (* One-time initialization End *)
  end_if;

  (* Calling read function to DB Start *)
  {{ uint32_t res =  getDBEntryStruct(GetFbVar(KEY),
                                      GET_FB_ARRAY_POINTER(SUBS_NUMB), /* Number of sub-
indexes Start */ 11 /* Number of sub-indexes End */,
                                      GET_FB_ARRAY_POINTER(SUBS_TYPE),
                                      GET_FB_ARRAY_POINTER(DB_ARR_SEND_RCV),
                                      GET_FB_ARRAY_POINTER(ERROR_ARR) ); }}
  (* Calling read function to DB End *)

  (* Data Deserialization Start *)

  OUT.bool_val := UDINT_TO_BOOL(DB_ARR_SEND_RCV[1]);

  OUT.sint_val := UDINT_TO_SINT(DB_ARR_SEND_RCV[2]);
  OUT.int_val  := UDINT_TO_INT(DB_ARR_SEND_RCV[3]);
  OUT.dint_val := UDINT_TO_DINT(DB_ARR_SEND_RCV[4]);
  UDINT_SRLZ_LINT(IN_1 := DB_ARR_SEND_RCV[5], IN_2 := DB_ARR_SEND_RCV[6], OUT =>
OUT.lint_val);
```

```
  OUT.usint_val := UDINT_TO_USINT(DB_ARR_SEND_RCV[7]);
  OUT.uint_val  := UDINT_TO_UINT(DB_ARR_SEND_RCV[8]);
  OUT.udint_val := DB_ARR_SEND_RCV[9];
  UDINT_SRLZ_ULINT(IN_1 := DB_ARR_SEND_RCV[10], IN_2 := DB_ARR_SEND_RCV[11], OUT => OUT.ulint_val);

  UDINT_DESRLZ_REAL(IN_1 := DB_ARR_SEND_RCV[12], OUT => OUT.real_val);
  UDINT_SRLZ_LREAL(IN_1 := DB_ARR_SEND_RCV[13], IN_2 := DB_ARR_SEND_RCV[14], OUT => OUT.lreal_val);

  (* Data Deserialization End *)

  {{ SetFbVar(ERROR, res); }}
  (* Functional Block Code End *)
END_FUNCTION_BLOCK
```

Пример вызова ФБ

```
set_db_struct_instance(Key := 'App/test struct access', IN := user_struct, ERROR => error_code);
```

```
get_db_struct_instance(Key := 'App/test struct access', OUT => user_struct, ERROR => error_code);
```

Приложение A. API доступа к БД

API чтения из БД

```
/**
 * @brief Запись в БД
 *
 * @param key       - Ключ к индексу в БД
 * @param subs      - Указатель на массив номеров подындексов. Тип элементов массива: `uint8_t`. Массив длиной `subs_num`
 * @param subs_num  - Количество подындексов (кол-во элементов массивов `subs` и `sub_types`)
 * @param sub_types - Указатель на массив с кодом типа подындексов. Тип элементов массива: `uint8_t`. Массив длиной `subs_num`
 * @param values    - Указатель на массив с сырыми данными подындексов. Тип элементов массива: `uint8_t`
 * @param errors    - Указатель на массив с кодом ошибок доступа. Тип массива: `uint32_t`
 * @return uint32_t - Код ошибки доступа
 */
uint32_t getDBEntryStruct(struct IEC_STRING key,
                          const uint8_t     *subs,
                          uint8_t           subs_num,
                          const uint8_t     *sub_types,
                          uint8_t           *values,
                          uint32_t          *errors);
```

API записи в БД

```
/**
 * @brief Запись в БД
 *
 * @param key       - Ключ к индексу в БД
 * @param subs      - Указатель на массив номеров подындексов. Тип элементов массива: `uint8_t`. Массив длиной `subs_num`
 * @param subs_num  - Количество подындексов (кол-во элементов массивов `subs` и `sub_types`)
 * @param sub_types - Указатель на массив с кодом типа подындексов. Тип элементов массива: `uint8_t`. Массив длиной `subs_num`
 * @param values    - Указатель на массив с сырыми данными подындексов. Тип элементов массива: `uint8_t`
 * @param errors    - Указатель на массив с кодом ошибок доступа. Тип массива: `uint32_t`
 * @return uint32_t - Код ошибки доступа
 */
uint32_t setDBEntryStruct(struct IEC_STRING key,
                          const uint8_t     *subs,
                          uint8_t           subs_num,
                          const uint8_t     *sub_types,
                          const uint8_t     *values,
                          uint32_t          *errors);
```

Приложение Б. Вспомогательные функции и макросы на языке "C"

Макрос получения указателя на массив объявленный в ФБ

```
/**
* @brief Получения указателя на массив объявленный в ФБ
* @param _FB_ARRAY_NAME_ - Имя массива. Указывается в верхенем регистре (заглавными буквами)
*/
GET_FB_ARRAY_POINTER(_FB_ARRAY_NAME_)
```

Макросы чтения и записи параметров ФБ

```
GetFbVar(var,...)
SetFbVar(var,val,...)
```