Библиотека для работы с файлами

TXV 003 41.01
второе издание
март 2008г.
фирма оставляет за собой право проводить изменения

История изменений

Дата	Издание	Описание изменений
Январь 2008	1	Первое издание
Март 2008	2	Дополнено описание функции DiskInfo

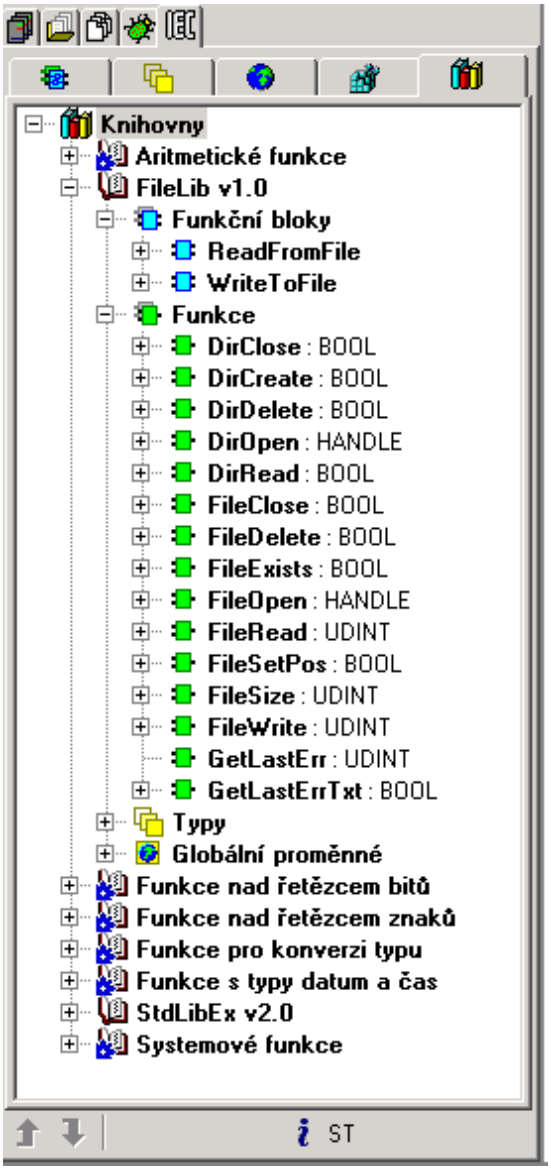
СОДЕРЖАНИЕ

1 *Введение*

Библиотека FileLib содержит набор функций, необходимых для работы с файлами. Эту библиотеку можно использовать только для системы управления, оснащенной системой файлов.

Библиотека FileLib содержит основные функции (низкого уровня) для работы с файлами и книгами файлов, а также необходимые типы данных и постоянные величины, и наконец функциональные блоки для считывания и записи файлов. Функциональные блоки используют основные функции файла и разделяют считывание или же запись файлов на большее количество циклов, чтобы оптимизировать трудоемкость данных операций по времени .

1. На следующем рисунке указана структура библиотеки FileLib в среде „Mosaic“.



Если мы хотим использовать функции из библиотеки FileLib в программе приложения ПЛК, необходимо сначала добавить эту библиотеку к проекту. Библиотека поставляется как составная часть установки среды „Mosaic“ начиная версией 2.6.0. Центральный процессор системы ПЛК должен иметь внедренную поддержку для операций с файлами.

Структура каталогов файлов

Загрузочный каталог файлов для операций с файлами в системе ПЛК называется ROOT. Программатор системы ПЛК может работать только с теми файлами и книгами файлов, которые находятся в каталоге файлов ROOT. Остальные файлы и каталог файлов не доступны из программы ПЛК. Каталог файлов ROOT, таким образом, является рабочим каталогом файлов программатора ПЛК.

Названия файлов

Система файлов поддерживает названия файлов в конвенции DOS 8.3. Название файла состоит из самого названия файла (максимум 8 знаков) и из расширения (максимум 3 знака). Эти две части разделены точкой. В названиях файлов нельзя использовать знаки препинания, пробелы и, наконец, знаки " * "," ? ". Знаки национальных азбук не поддерживаются в названиях. Большие и маленькие буквы в названиях файлов не различаются. Заменимые знаки в названиях файлов (напр. *.*) не поддерживаются.

Путь к файлу

Путь к файлу – это определение позиции файла на диске по отношению к каталогу файлов ROOT. То есть путь содержит названия каталогов файлов, в которых сохранен файл. Для названия каталогов файлов на пути действуют такие же правила, как для названия файла. Отдельные названия каталогов файлов на пути разделены знаком черты дроби „/“. Система файлов ПЛК поддерживает только абсолютные пути. Относительные пути или изменение рабочего каталога файлов не поддерживаются.
Максимальная длина названия файла включая пути ограничена на 65 знаков (см. постоянная *MAX_PATH*).

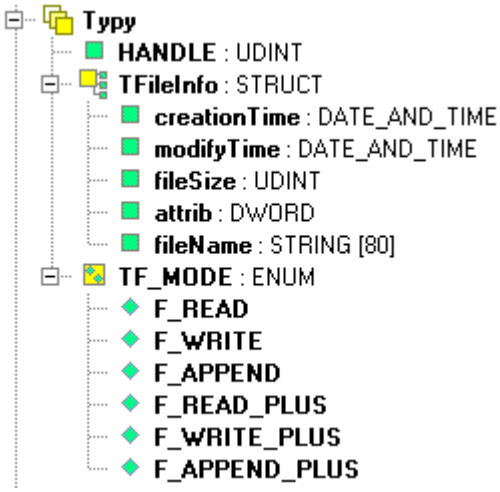
Видимость файлов для веб-серверов ПЛК

Веб-сервер ПЛК использует в качестве загрузочного каталога файлов ROOT/WWW. Если файлы должны быть доступны через веб-интерфейс, они должны быть сохранены на данном пути (напр. в каталоге файлов ROOT/WWW/data).

Более подробная информация о типе поддерживаемого файла системы указана в описании центральных процессоров ПЛК.

2 Типы данных

В библиотеке FileLib определены следующие типы данных:

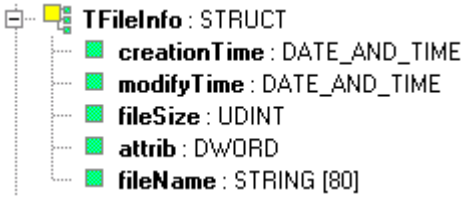


2.1 Тип HANDLE

HANDLE - это тип данных, который является производным от основного типа UDINT и используется для идентификатора файла. К каждому открытому файлу назначен идентификатор, который используется для проведения операции считывания из файла, записи в файл, закрытия файла и т.п.

2.2 Тип TFileInfo

TFileInfo – это структура, которую функция *DirOpen* возвращает и, наконец, *DirRead* содержит информацию о файле.

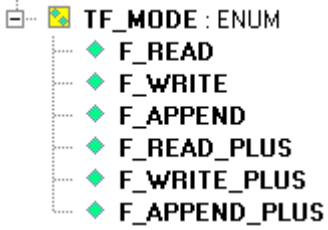


Значение отдельных позиций структуры *TFileInfo* следующее:

- *creationTime* Дата и время создания файла
- *modifyTime* Дата и время последней модификации файла
- *fileSize* Размер файла в байтах
- *attrib* Описатели файла
- *fileName* Название файла

2.3 Тип TF_MODE

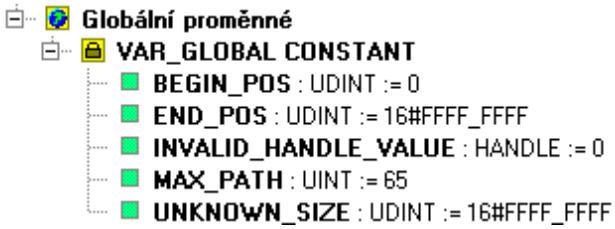
TF_MODE – это номенклатурный тип, который используется для постоянной величины, определяющей способ открытия файла функций *FileOpen*.



Значение отдельных позиций перечня см. в описании функции FileOpen.

3 Постоянные величины

В библиотеке FileLib определены следующие постоянные величины:



3.1 Постоянная BEGIN_POS

Постоянная *BEGIN_POS* - это постоянная типа UDINT и используется в качестве параметра функции *FileSetPos* в том случае, если мы хотим настроить считывание или запись от начала файла, постоянная имеет значение 0.

3.2 Постоянная END_POS

Постоянная *END_POS* - это постоянная типа UDINT и используется в качестве параметра функции *FileSetPos* в том случае, если мы хотим настроить считывание или запись от конца файла, постоянная имеет значение 16#FFFF_FFFF.

3.3 Постоянная INVALID_HANDLE_VALUE

Постоянная *INVALID_HANDLE_VALUE* - это постоянная типа HANDLE и используется в качестве величины возврата функции *FileOpen* или же *DirOpen* в том случае, если не удастся открыть файл или же каталог файлов. Постоянная имеет значение 0 и наконец означает недействительный идентификатор файла.

3.4 Постоянная MAX_PATH

Постоянная *MAX_PATH* - это постоянная типа UINT и задает максимальную возможную длину названия файла или каталога файлов (включая пути). Постоянная имеет значение 65.

3.5 Постоянная UNKNOWN_SIZE

Постоянная *UNKNOWN_SIZE* - это постоянная типа UDINT и используется в качестве величины возврата функции FileSize в том случае, если не удастся определить размер файла, постоянная имеет значение 16#FFFF_FFFF.

4 Функции для работы с файлами

Библиотека FileLib содержит следующие функции для работы с файлами:

- *FileOpen* открытие файла
- *FileRead* считывание из файла
- *FileWrite* запись в файл
- *FileClose* закрытие файла
- *FileExists* тестирование существования файла
- *FileSize* определение размера файла
- *FileDelete* отмена файла
- *FileSetPos* настройка позиции в файле

Для работы с каталогом файлов библиотека FileLib содержит следующие функции:

- *DirOpen* открытие каталога файлов
- *DirRead* считывание с каталога файлов
- *DirClose* закрытие каталога файлов
- *DirCreate* создание каталога файлов
- *DirDelete* отмена каталога файлов

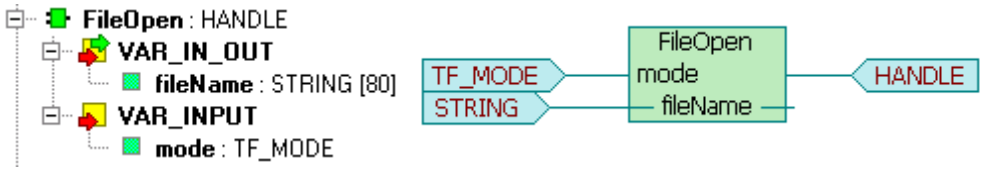
Следующей группой функций являются функции, позволяющие специфицировать ошибку, возникшую при проведении операций с файлами:

- *GetLastErr* определить код последней ошибки
- *GetLastErrTxt* перевод кода ошибки на текст сообщения

И наконец в распоряжении имеется функция, позволяющая определить информацию о диске:

- *DiskInfo* *определить размер диска и количество свободного места на диске*

4.1 Функция FileOpen



Функция **FileOpen** откроет файл специфицированный переменной *fileName*. Переменная *mode* задает требуемый способ доступа к файлу.

Функция **FileOpen** вернет идентификатор открытого файла. Если файл не удастся открыть, функция вернет недействительный идентификатор (INVALID_HANDLE_VALUE).

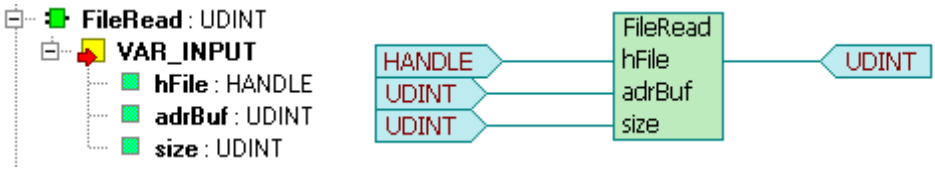
Описание переменных :

	Переменная	***Тип***	***Значение***
VAR_IN_OUT			
	fileName	STRING	Название файла включая пути
VAR_INPUT			
	mode	TF_MODE	Тип доступа к файлу
			F_READ Открыть файл для считывания, файл должен
			F_WRITE
			Открыть файл для записи Данные будут записываться на начало файла
			F_APPEND
			Открыть файл для записи Данные будут записываться на конец файла
FileOpen			
	величины возврата	HANDLE	Идентификатор открытого файла. Если файл не удастся открыть недействительный идентификатор возвращается (INVALID_HANDLE_VALUE)

Пример программы с вызовом функции FileOpen :

```
PROGRAM ExampleFileOpen
  VAR
        path         : STRING := 'data/';
        name         : STRING := 'log.txt';
        hf           : HANDLE;                // file handle
       END_VAR
  VAR_TEMP
    result       : BOOL;
        fullFileName : STRING;
       END_VAR

  fullFileName := path + name;
  hf := FileOpen(fileName := fullFileName, mode := F_READ);
  IF hf <> INVALID_HANDLE_VALUE THEN
        result := FileClose( hFile := hf);
       END_IF;

END_PROGRAM
```

4.2 Функция FileRead



Функция **FileRead** запишет из файла определенное количество знаков и, наконец, сохранит их в память ПЛК. Файл специфицирован переменной *hFile*. Переменная, в которую введенные данные будут сохранены, специфицирована в *adrBuf*. Переменная *size* задает количество знаков, которое будет из файла записано.

Функция **FileRead** вернет количество фактически записанных знаков.

Описание переменных :

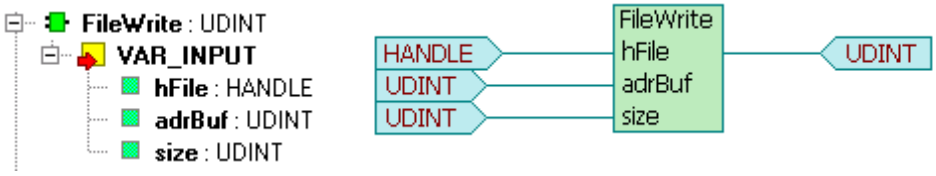
	Переменная	***Тип***	***Значение***
VAR_INPUT			
	hFile	HANDLE	Идентификатор файла
	adrBuf	UDINT	Адрес переменной, на который будут записаны знаки из файла
	size	UDINT	Количество знаков, которые должны быть записаны
FileRead			
	величины возврата	UDINT	Количество фактически записанных знаков

Пример программы с вызовом функции FileRead :

```
PROGRAM ExampleFileRead
       VAR
        path   : STRING := 'data/';
        name   : STRING := 'log.txt';
        hf     : HANDLE;                // file handle
        buffer : STRING[200];
        lenght : UDINT;
       END_VAR
  VAR_TEMP
    result       : BOOL;
        fullFileName : STRING;
       END_VAR

  fullFileName := path + name;
  hf := FileOpen(fileName := fullFileName, mode := F_READ);
  IF hf <> INVALID_HANDLE_VALUE THEN
       lenght := FileRead( hFile  := hf,
                 adrBuf := PTR_TO_UDINT( ADR(buffer)),
                 size   := 100);
```

```
    result := FileClose( hFile := hf);
  END_IF;
END_PROGRAM
```

4.3 *Функция FileWrite*



Функция **FileWrite** копирует содержание переменной ПЛК в файл. Файл специфицирован переменной *hFile*. Переменная, из которой будут данные копироваться, специфицирована в *adrBuf*. Переменная *size* задает количество знаков, которое будет записано в файл.

Функция **FileWrite** вернет количество фактически записанных знаков.

Описание переменных :

	Переменная	***Тип***	***Значение***
VAR_INPUT			
	hFile	HANDLE	Идентификатор файла
	adrBuf	UDINT	Адрес переменной, содержание которой будет записано в файл
	size	UDINT	Количество записываемых знаков
FileWrite			
	величины возврата	UDINT	Количество фактически записанных знаков

Пример программы с вызовом функции FileWrite :

```
PROGRAM ExampleFileWrite
       VAR
    start  : BOOL;
        fname  : STRING := 'data/log.txt';
        hf     : HANDLE;                // file handle
        buffer : STRING[200];
        lenght : UDINT;
       END_VAR
  VAR_TEMP
    result       : BOOL;
       END_VAR

  IF start THEN
       start := FALSE;
       buffer := 'Hello world!';
    hf := FileOpen(fileName := fname, mode := F_WRITE);
    IF hf <> INVALID_HANDLE_VALUE THEN
         lenght := FileWrite( hFile  := hf,
                    adrBuf := PTR_TO_UDINT( ADR(buffer)),
                    size   := len( buffer));
      result := FileClose( hFile := hf);
```

```
    END_IF;
       END_IF;
END_PROGRAM
```

4.4 Функция FileClose

Функция **FileClose** закроет файл специфицированный переменной *hFile*. Все внутренние буфера для записи перед закрытием записаны в файл. После закрытия файла невозможно из файла считывать или в него записывать. Идентификатор файла перестанет ассоциироваться с файлом.

Функция **FileClose** возвращает величину TRUE, если файл удастся закрыть, в противном случае возвращает величину FALSE. Причину или же ошибки можно определить с помощью функции **GetLastErr**.

Описание переменных :

	Переменная	***Тип***	***Значение***
VAR_INPUT			
	hFile	HANDLE	Идентификатор файла
FileClose			
	величины возврата	BOOL	TRUE Если файл удачно закрыт, FALSE в остальных случаях

Пример программы с вызовом функции **FileClose** :

```
PROGRAM ExampleFileClose
  VAR
        path          : STRING := 'data/';
        name          : STRING := 'log.txt';
        hf            : HANDLE;                // file handle
       END_VAR
  VAR_TEMP
    result       : BOOL;
        fullFileName : STRING;
       END_VAR

  fullFileName := path + name;
  hf := FileOpen(fileName := fullFileName, mode := F_READ);
  IF hf <> INVALID_HANDLE_VALUE THEN
        result := FileClose( hFile := hf);
       END_IF;

END_PROGRAM
```

4.5 Функция FileExists

Функция **FileExists** тестирует существование файла, специфицированного переменной *fileName.*

Функция **FileExists** вернет TRUE если файл существует. Если файл не удастся найти, функция вернет FALSE.

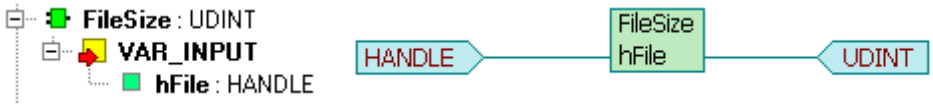
Описание переменных :

	Переменная	***Тип***	***Значение***
VAR_IN_OUT			
	fileName	STRING	Название файла, включая пути
FileExists			
	величины возврата	BOOL	TRUE если файл существует, FALSE в остальных случаях

Пример программы с вызовом функции FileExists :

```
PROGRAM ExampleFileExist
       VAR
        fname  : STRING;
        del    : BOOL;
       END_VAR

  IF NOT del THEN
        fname := 'data/log.txt';
        IF FileExists( fileName := fname) THEN
         del := FileDelete( fileName := fname);
        END_IF;
       END_IF;
END_PROGRAM
```

4.6 Функция FileSize



Функция **FileSize** определит размер файла, специфицированного переменной *hFile*.

Функция **FileSize** возвращает размер файла в байтах. Если при определении размера файла произойдет ошибка, функция вернет UKNOWN_SIZE. Причину ошибки можно определить с помощью функции **GetLastErr**.

Описание переменных :

	Переменная	***Тип***	***Значение***
VAR_INPUT			
	hFile	HANDLE	Идентификатор файла
FileSize			
	величины возврата	UDINT	Размер файла При возникновении ошибки возвращает UKNOWN_SIZE, т.е. 16#FFFF_FFFF

Пример программы с вызовом функции FileSize :

```
PROGRAM ExampleFileSize
       VAR
        fname  : STRING;
    hf     : HANDLE;
    lenght : UDINT;
        result : BOOL;
       END_VAR

  fname := 'data/log.txt';
  hf := FileOpen(fileName := fname, mode := F_READ);
  IF hf <> INVALID_HANDLE_VALUE THEN
        lenght := FileSize(hFile := hf);
    result := FileClose( hFile := hf);
    result := result AND lenght <> UNKNOWN_SIZE;
       END_IF;
END_PROGRAM
```

4.7 Функция FileDelete

Функция **FileDelete** сотрет файл, специфицированный переменной *fileName.*
Эту функцию можно использовать также для стирания каталога файлов. Для успешного стирания каталога файлов неизбежно, чтобы каталог файлов был пустой.

Функция **FileDelete** вернет TRUE если файл удастся стереть. В противном случае функция вернет FALSE. Причину или же ошибки можно определить с помощью функции **GetLastErr**.

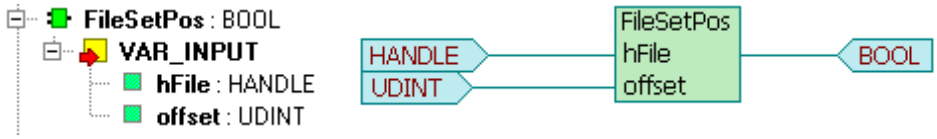
Описание переменных :

	Переменная	***Тип***	***Значение***
VAR_IN_OUT			
	fileName	STRING	Название файла, включая пути
FileDelete			
	величины возврата	BOOL	TRUE если файл стерт, FALSE в остальных случаях

Пример программы с вызовом функции FileDelete :

```
PROGRAM ExampleFileDelete
  VAR
        fname  : STRING;
        del    : BOOL;
       END_VAR

  IF NOT del THEN
        fname := 'data/log.txt';
        IF FileExists( fileName := fname) THEN
         del := FileDelete( fileName := fname);
        END_IF;
       END_IF;
END_PROGRAM
```

4.8 Функция FileSetPos



Функция **FileSetPos** позволяет настроить позицию для считывания из файла или же для записи в файл. Файл специфицирован переменной *hFile*. Требуемую позицию в файле задает переменная *offset*.

Функция **FileSetPos** возвращает величину TRUE, если удастся настроить требуемую позицию. В противном случае возвращает величину FALSE. Причину или же ошибки можно определить с помощью функции **GetLastErr**.

Описание переменных :

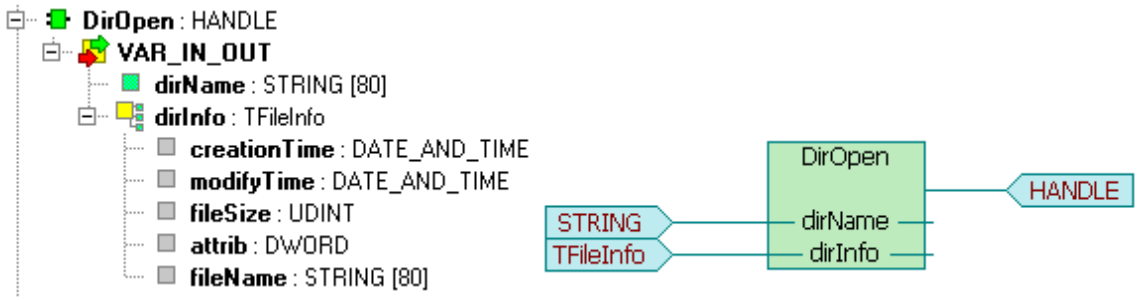
	Переменная	***Тип***	***Значение***
VAR_INPUT			
	hFile	HANDLE	Идентификатор файла
	offset	UDINT	Позиция в файле
			0 или BEGIN_POS — На начало файла
			0 > offset > END_POS — На заданную позицию
			16#FFFF_FFFF или END_POS — На конец файла
FileSetPos			
	величины возврата	BOOL	Возвращает TRUE, если удастся настроить требуемую позицию в файле В противном случае возвращает FALSE

Пример программы с вызовом функции FileSetPos:

```
PROGRAM ExampleFileSetPos
 VAR
        path    : STRING := 'data/';
        name  : STRING := 'log.txt';
        hf    : HANDLE;         // file handle
        buffer  : STRING[200];
        lenght : UDINT;
       END_VAR
  VAR_TEMP
  result        : BOOL;
        fullFileName : STRING;
       END_VAR

 fullFileName := path + name;
```

```
  hf := FileOpen(fileName := fullFileName, mode := F_READ);
 IF hf <> INVALID_HANDLE_VALUE THEN
        IF FileSetPos( hFile := hf, offset := 10) THEN
         lenght := FileRead( hFile  := hf,
                   adrBuf := PTR_TO_UDINT( ADR(buffer)),
                   size  := 100);
      result := FileClose( hFile := hf);
  END_IF;
       END_IF;
END_PROGRAM
```

4.9 Функция DirOpen



Функция **DirOpen** откроет каталог файлов, специфицированный переменной *dirName*. После этого определит информацию о первом файле в каталоге файлов и наконец эту информацию запишет в переменную *dirInfo*. Информацию о следующих файлах в каталоге файлов можно определить с помощью функции DirRead.

Функция **DirOpen** вернет идентификатор открытого каталога файлов. Если файл не удастся открыть, функция вернет недействительный идентификатор (INVALID_HANDLE_VALUE).

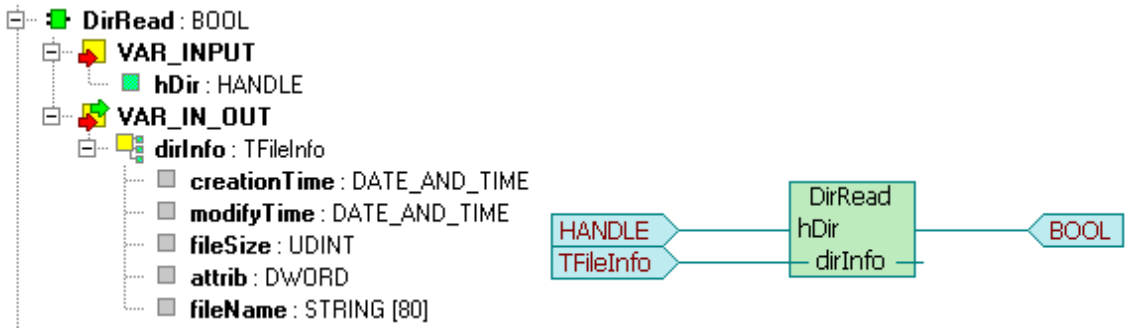
Описание переменных :

	Переменная	***Тип***	***Значение***
VAR_IN_OUT			
	dirName	STRING	Название каталога файлов, включая пути
	dirInfo	TFileInfo	Структура с информацией о первом файле в каталоге
	creationTime	DT	
	modifyTime	DT	
	fileSize	UDINT	
	attrib	DWORD	
	fileName	STRING	Название файла
DirOpen			
	величины возврата	HANDLE	Идентификатор открытого каталога файлов. Если каталог файлов не удастся открыть, возвращается недействительный идентификатор (INVALID_HANDLE_VALUE)

Пример программы с вызовом функции DirOpen:

```
PROGRAM ExampleDirOpen
       VAR
        dn     : STRING := 'data/';     // dir name
        hd     : HANDLE;                // dir handle
        fi     : TFileInfo;
        result : BOOL;
       END_VAR

 hd := DirOpen(dirName := dn, dirInfo := fi);
       IF hd <> INVALID_HANDLE_VALUE THEN
        result := DirClose(hDir := hd);
       END_IF;
END_PROGRAM
```

4.10 *Функция DirRead*



Функция **DirRead** определит информацию о следующем файле в каталоге файлов и, наконец, эту информацию запишет в переменную *dirInfo*. Каталог файлов специфицирован переменной *dirHandle*. Перед вызовом данной функции каталог файлов должен быть открыт функцией DirOpen.

Функция **DirRead** вернет TRUE если операция была проведена успешно. В противном случае вернет величину FALSE.

Описание переменных :

	Переменная	***Тип***	***Значение***
VAR_INPUT			
	dirHandle	HANDLE	Идентификатор каталога файлов
VAR_IN_OUT			
	dirInfo	TFileInfo	Структура с информацией о следующем файлае в
	creationTime	DT	
	modifyTime	DT	
	fileSize	UDINT	
	attrib	DWORD	
	fileName	STRING	Название файла
DirRead			
	величины возврата	BOOL	TRUE если операция будет удачно проведена, FALSE в остальных случаях

Пример программы с вызовом функции DirRead:

```
TypE
  T_FILE_INFO :
        STRUCT
         name : STRING[12];          // DOS name 8.3
         len  : UDINT;               // file length
        END_STRUCT;
END_TypE

      VAR_GLOBAL
  dirList : ARRAY[0..31] OF T_FILE_INFO;
END_VAR

PROGRAM ExampleDirRead
       VAR
        dn     : STRING := 'data/';   // dir name
        hd     : HANDLE;              // dir handle
        di     : TFileInfo;           // dir info
        count  : USINT;               // number of files
        done   : BOOL;
        err    : BOOL;
        open   : BOOL;
        result : BOOL;
       END_VAR

  IF NOT done THEN
       IF NOT open THEN
        hd := DirOpen( dirName := dn, dirInfo := di);
        IF hd <> INVALID_HANDLE_VALUE THEN
         count := 1; open := TRUE;
         dirList[0].name := di.fileName;
         dirList[0].len  := di.fileSize;
        ELSE
         count := 0; err := TRUE; done := TRUE;
        END_IF;
       ELSE
        IF DirRead( hDir := hd, dirInfo := di) THEN
         dirList[count].name := di.fileName;
         dirList[count].len  := di.fileSize;
         count := count + 1;
        ELSE
         done := TRUE; err := FALSE; open := FALSE;
       result := DirClose( hDir := hd);
     END_IF;
       END_IF;
      END_IF;
END_PROGRAM
```

4.11 Функция DirClose

Функция **DirClose** закроет каталог файлов, специфицированной переменной *hDir*. После закрытия каталога файлов невозможно из каталога файлов считывать функцию ReadDir. Идентификатор каталога файлов перестанет ассоциироваться с файлом.

Функция **DirClose** возвращает величину TRUE, если каталог файлов удастся закрыть, в противном случае возвращает величину FALSE. Причину или же ошибки можно определить с помощью функции **GetLastErr**.

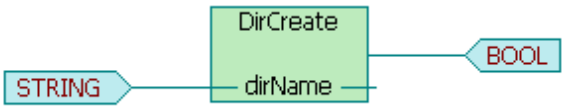
Описание переменных :

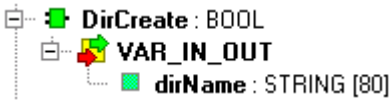
	Переменная	***Тип***	***Значение***
VAR_INPUT			
	hFile	HANDLE	Идентификатор файла
DirClose			
	величины возврата	BOOL	TRUE если каталог файлов удачно закрыт, FALSE в остальных случаях

Пример программы с вызовом функции DirClose:

```
PROGRAM ExampleDirClose
       VAR
        dn     : STRING := 'data/';     // dir name
        hd     : HANDLE;                // dir handle
        fi     : TFileInfo;
        result : BOOL;
       END_VAR

 hd := DirOpen(dirName := dn, dirInfo := fi);
       IF hd <> INVALID_HANDLE_VALUE THEN
        result := DirClose(hDir := hd);
       END_IF;
END_PROGRAM
```

4.12 Функция DirCreate





Функция **DirCreate** создаст каталог файлов специфицированной переменной *dirName*.

Функция **DirCreate** возвращает величину TRUE, если каталог файлов удастся создать, в противном случае возвращает величину FALSE. Причину или же ошибки можно определить с помощью функции **GetLastErr**.

Описание переменных :

	Переменная	***Тип***	***Значение***
VAR_IN_OUT			
	dirName	STRING	Название файла, включая пути
DirCreate			
	величины возврата	BOOL	TRUE если каталог файлов удачно создан, FALSE в остальных случаях

Пример программы с вызовом функции DirCreate:

```
PROGRAM ExampleDirCreate
       VAR
        dn     : STRING := 'NEW/';      // dir name
        result : BOOL;                  // dir exists
       END_VAR

  IF NOT FileExists( fileName := dn) THEN
        result := DirCreate( dirName := dn);
       ELSE
        result := TRUE;
       END_IF;
END_PROGRAM
```

4.13 Функция DirDelete

Функция **DirDelete** сотрет каталог файлов, специфицированный переменной *dirName*. Для успешного стирания каталога файлов необходимо, чтобы каталог файлов был пустой (чтобы не содержал какие-либо файлы).

Функция **DirDelete** вернет TRUE если каталог файлов удастся стереть. В противном случае функция вернет FALSE. Причину или же ошибки можно определить с помощью функции **GetLastErr**.

Описание переменных :

	Переменная	*Тип*	*Значение*
VAR_IN_OUT			
	fileName	STRING	Название каталога файлов, включая пути
DirDelete			
	величины возврата	BOOL	TRUE если каталог файлов стерт, FALSE в остальных случаях

Пример программы с вызовом функции DirDelete:

```
PROGRAM ExampleDirDelete
       VAR
        dn     : STRING := 'NEW/';      // dir name
        fi     : TFileInfo;
    dh     : HANDLE;
    result : BOOL;
       END_VAR

  IF FileExists( fileName := dn) THEN
       dh := DirOpen( dirName := dn, dirInfo := fi);
       IF dh = INVALID_HANDLE_VALUE THEN
        // directory is empty
        result := DirDelete( dirName := dn);
       ELSE
        // directory is not empty
      // it means we can't delete dir
      result := DirClose(hDir := dh);
        END_IF;
       END_IF;
```

```
END_PROGRAM
```

4.14 Функция GetLastErr

Функция **GetLastErr** вернет код последней зарегистрированной ошибки, возникшей при проведении операции в файле. Этот код может использоваться в качестве параметра функции GetLastErrTxt, которая потом вернет текст описания ошибки.

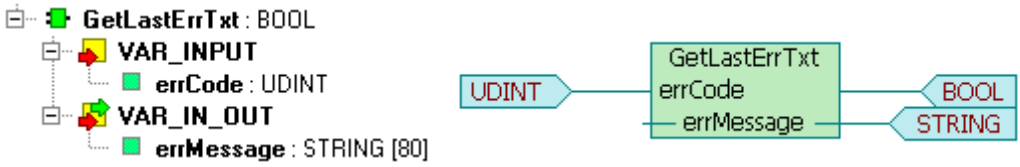
Описание переменных :

	Переменная	*Тип*	*Значение*
GetLastErr			
	величины возврата	UDINT	Код последней ошибки при проведении операции в файле

Пример программы с вызовом функции GetLastErr:

```
PROGRAM ExampleGetLastErr
       VAR
        fname   : STRING;
        error   : UDINT;
        errTxt  : STRING;
        valid   : BOOL;
       END_VAR

  fname := 'data/log.txt';
       IF NOT FileExists( fileName := fname) THEN
        error := GetLastErr();
    valid := GetLastErrTxt(errCode := error, errMessage := errTxt);
  END_IF;
END_PROGRAM
```

5 Функция GetLastErrTxt



Функция **GetLastErrTxt** запишет текст описания ошибки в переменную *errMessage*. Текст описания ошибки отвечает коду ошибки, который специфицирован переменной *errCode*.

Функция **GetLastErrTxt** возвращает TRUE, если к заданному коду ошибки существует текст описания. В противном случае возвращает FALSE.

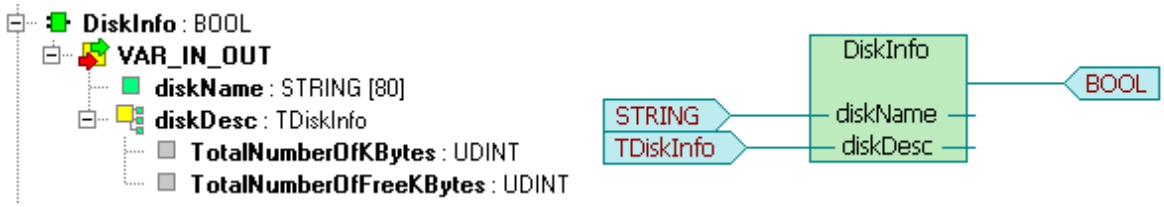
Описание переменных :

	Переменная	***Тип***	***Значение***
VAR_INPUT			
	errCode	UDINT	Код ошибки
VAR_IN_OUT			
	errMessage	STRING	Текст описания ошибки
GetLastErrTxt			
	величины возврата	BOOL	Возвращает TRUE, если существует описание ошибки в противном случае возвращает FALSE

Пример программы с вызовом функции GetLastErrTxt:

```
PROGRAM ExampleGetLastErrTxt
       VAR
        fname   : STRING;
        error   : UDINT;
        errTxt  : STRING;
        valid   : BOOL;
       END_VAR

  fname := 'data/log.txt';
       IF NOT FileExists( fileName := fname) THEN
        error := GetLastErr();
    valid := GetLastErrTxt(errCode := error, errMessage := errTxt);
  END_IF;
END_PROGRAM
```

5.1 Функция DiskInfo



Функция **DiskInfo** определит общий размер диска и количество свободного места на диске. Название диска должно быть указано в переменной diskName. Если данная переменная пустая, функция **DiskInfo** вернет информацию о диске, который установлен в качестве дефо (напр. в системах Foxtrot это карта памяти).

Функция **DiskInfo** возвращает TRUE, если удастся определить информацию о заданном диске. В противном случае возвращает FALSE.
Функция **DiskInfo** заполнит позиции *TotalNumberOfKBytes* и наконец *TotalNumberOfFreeKBytes* в переменной *diskDesc*. Обе величины указаны в килобайтах.

Описание переменных :

	Переменная	***Тип***	***Значение***
VAR_IN_OUT			
	diskName	STRING	Название диска
	diskDesc	TDiskInfo	Информация о диске
	. *TotalNumberOfKBytes*	UDINT	Общий размер диска в KB
	. *TotalNumberOfFreeKBytes*	UDINT	Количество свободного места на диске в KB
DiskInfo			
	величины возврата	BOOL	Вернет TRUE, если удастся определить информацию о диске. В противном случае вернет FALSE

Пример программы с вызовом функции DiskInfo:

```
PROGRAM ExampleDiskInfo
       VAR
        disc   : STRING := '';    // defo disk
        info   : TDiskInfo;       // info about disk
        result : BOOL;
       END_VAR

  result := DiskInfo(diskName := disc, diskDesc := info);
END_PROGRAM
```

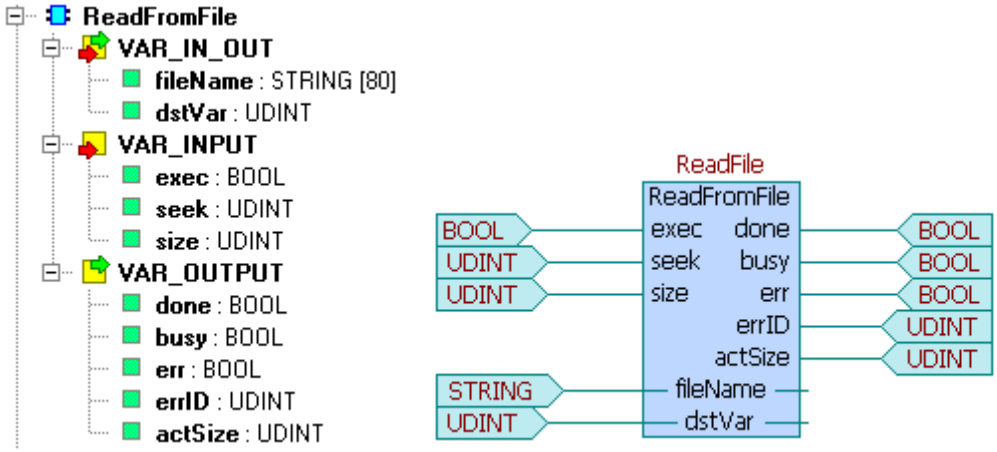
6 *Функциональные блоки для работы с файлами*

Библиотека FileLib содержит следующие функциональные блоки для работы с файлами:

- *ReadFromFile* считывание данных из файла
- *WriteToFile* запись данных в файл

Функциональные блоки используют основные функции файла и разделяют считывание или же запись файлов на большее количество циклов так, чтобы была оптимизирована трудоемкость данных операций по времени.

6.1 Функциональный блок ReadFromFile



Функциональный блок **ReadFromFile** прочитает данные из файла и сохранит их в переменной памяти ПЛК. Название файла, из которого будет происходить считывание, задает переменная *fileName*. Адрес переменной, в которую будут данные сохранены, специфицирован переменной *dstVar*. Считывание из файла будет начато на передний фронт переменной *exec*. Переменная *seek* задает позицию от начала файла, с которого будут данные считываться. Считывание данных закончится, если будут введены все требуемые данные, размер которых задает переменная *size* или если при считывании будет достигнуто конца файла.

Функциональный блок **ReadFromFile** настроит TRUE в переменной *done* в момент, когда запишет последней блок данных из файла. Во время записи данных переменная *done* имеет величину FALSE, а переменная *busy* величину TRUE. Количество фактически записанных байтов задает переменная *actSize*. Если было проведено считывание без ошибки, переменная *err* имеет значение FALSE, в случае возникновения ошибки имеет значение TRUE и, наконец, в переменной *errID* сохранен код ошибки. Он может использоваться в качестве входной переменной функции GetLastErrTxt для получения текста описания возникшей ошибки.

Описание переменных :

	Переменная	***Тип***	***Значение***
VAR_IN_OUT			
	fileName	STRING	Название файла, включая пути
	dstVar	UDINT	Адрес переменной, в которой будут сохранены данные считанные из файла
VAR_INPUT			
	exec	BOOL	Управляющая переменная. Передний фронт (переход из величины FALSE на величину TRUE) начнет считывание из файла
	seek	UDINT	Offset от начала файла, от которого считывание начато

	Переменная	***Тип***	***Значение***
			0 От начала файла
			seek <> 0 От заданной позиции
	size	UDINT	Требуемые размеры считываемых данных
VAR_OUTPUT			
	done	BOOL	Имеет значение TRUE в момент окончания считывания из файла В противном случае возвращает FALSE
	busy	BOOL	Имеет значение TRUE во время считывания данных из файла
	err	BOOL	Маркер ошибки при считывании из файла Если операция была проведена успешно, имеет значение FALSE, в противном случае TRUE.
	errID	UDINT	Код ошибки. Если операция была проведена успешно, errID = 0. в случае ошибки errID <> 0.
	actSize	UDINT	Количество фактически введенных байтов

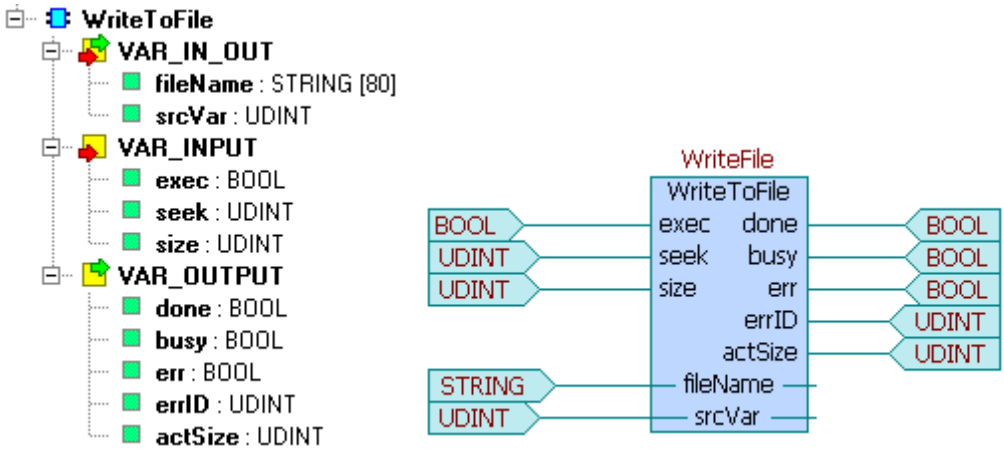
Следующий пример показывает определение функционального блока *ReadFromFile* в языке ST, который указан здесь для возможности подробного понимания функции. Использование данного блока указано на примере в следующей главе.

```
FUNCTION_BLOCK ReadFromFile
(* Copy data from file to variable *)
  VAR_IN_OUT
        fileName     : STRING;       // file name
        dstVar       : UDINT;        // destination variable
      END_VAR
  VAR_INPUT
    exec          : BOOL;         // request
        seek         : UDINT;        // data offset in file
        size         : UDINT;        // data length <= size of variable
      END_VAR
  VAR_OUTPUT
    done          : BOOL;         // action is done
        busy         : BOOL;         // action in progress
        err          : BOOL;         // error flag
        errID        : UDINT;        // error number
        actSize      : UDINT;        // really read
      END_VAR
  VAR
    execTrig     : R_TRIG;
    errTrig      : R_TRIG;
        hnd          : HANDLE;
    adrVar       : UDINT;
      END_VAR
  VAR_TEMP
    restSize     : UDINT;
        reqSize      : UDINT;
```

```
      read         : UDINT;
     END_VAR

adrVar := PTR_TO_UDINT( ADR( dstVar));
execTrig( CLK := exec);

// open file and seek position
     IF execTrig.Q THEN
      busy := TRUE; done := FALSE; err := FALSE; errID := 0; actSize := 0;
      errTrig( CLK := FALSE);
  hnd := FileOpen(fileName := fileName, mode := F_READ);
  IF hnd = INVALID_HANDLE_VALUE THEN
       err := TRUE; busy := FALSE;
      ELSE
       IF seek <> 0 THEN
        IF FileSetPos( hFile := hnd, offset := seek) = FALSE THEN
         err := TRUE; busy := FALSE;
        END_IF;
       END_IF;
      END_IF;
     END_IF;

// read данные from file to variable (one sector per one ПЛК cycle)
     IF busy THEN
      IF size > actSize THEN
       restSize := size - actSize;
       IF restSize > 512 THEN reqSize := 512;          // more then one sector
                 ELSE reqSize := restSize;     // last sector
       END_IF;
    read := FileRead( hFile := hnd, adrBuf := adrVar + actSize,
                      size   := reqSize);
       actSize := actSize + read;
    IF read <> reqSize THEN
        busy := FALSE;
        IF read > 0 THEN done := TRUE;          // end of file
              ELSE err  := TRUE;          // propably any error
        END_IF;
       ELSE
        IF actSize = size THEN
         done := TRUE; busy := FALSE;
        END_IF;
       END_IF;
      ELSE
       done := TRUE;
      END_IF;
     ELSE
      done := done AND exec;
      err  := err  AND exec;
     END_IF;

// set error code, if any
     errTrig( CLK := err);
     IF errTrig.Q THEN errID := GetLastErr(); END_IF;

// close file
     IF errTrig.Q OR done THEN
      IF hnd <> INVALID_HANDLE_VALUE THEN
       IF FileClose( hFile := hnd) THEN
        hnd := INVALID_HANDLE_VALUE;
       END_IF;
      END_IF;
```

```
      END_IF;
END_FUNCTION_BLOCK
```

6.2 Функциональный блок WriteToFile



Функциональный блок **WriteToFile** запишет содержание переменной ПЛК в файл. Название файла, в который будет записываться, задает переменная *fileName*. Если файл не существует, будет основан адрес переменной, содержание которой будет записано в файл, специфицированный переменной src*Var*. Запись в файл будет начата на передний фронт переменной *exec*. Переменная *seek* задает позицию от начала файла, от которого будут данные записаны. Запись данных прекратится, если будут введены все требуемые данные, размер которых задает переменная *size*.

Функциональный блок **WriteToFile** настроит TRUE в переменной *done* в момент, когда будет записан последней блок данных в файле. Во время проведения записи данных переменная *done* имеет величину FALSE и переменная *busy* величину TRUE. Количество фактически записанных байтов задает переменная *actSize*. Если запись в файл была проведена без ошибки, переменная *err* имеет значение FALSE, в случае ошибки - имеет значение TRUE и наконец в переменной *errID* сохранен код ошибки. Он может использоваться в качестве входной переменной функции GetLastErrTxt для получения текста описания возникшей ошибки.

Описание переменных :

	Переменная	***Тип***	***Значение***
VAR_IN_OUT			
	fileName	STRING	Название файла, включая пути
	srcVar	UDINT	Адрес переменной, содержание которой будет введено в файл
VAR_INPUT			
	exec	BOOL	Управляющая переменная. Передний фронт (переход с величины FALSE на величину TRUE) начнет запись в файл
	seek	UDINT	Offset в файле, от которого начата запись
			0 От начала файла

	Переменная	*Тип*	*Значение*
			0 < seek < 16#FFFF_FFFF От заданной позиции
			16#FFFF_FFFF Придать данные на конец файла
	size	UDINT	Требуемые размеры вводимых данных
VAR_OUTPUT			
	done	BOOL	Имеет значение TRUE в момент окончания записи в файл. В противном случае возвращает FALSE
	busy	BOOL	Имеет значение TRUE во время записи в файл
	err	BOOL	Маркер ошибки при записи в файл Если операция была проведена успешно, имеет значение FALSE, в противном случае TRUE.
	errID	UDINT	Код ошибки. Если операция была проведена успешно, errID = 0. в случае ошибки errID <> 0.
	actSize	UDINT	Количество фактически введенных байтов

На следующем примере показано определение функционального блока *WriteToFile* в языке ST, который указан здесь для возможности подробного понимания функции. Использование данного блока указано на примере в следующей главе.

```
FUNCTION_BLOCK WriteToFile
(*
      Copy данные from variable to file.
      If file does not exist new file is created.
      If file exists file content is overwitten.
     *)
  VAR_IN_OUT
       fileName     : STRING;       // file name
       srcVar       : UDINT;        // variable address
      END_VAR
  VAR_INPUT
    exec          : BOOL;          // request
        seek          : UDINT;         // data offset in file
        size          : UDINT;         // data length (size of variable)
      END_VAR
  VAR_OUTPUT
    done          : BOOL;          // action is done
        busy          : BOOL;          // action in progress
        err           : BOOL;          // error flag
        errID         : UDINT;         // error number
        actSize       : UDINT;         // really written
      END_VAR
  VAR
    eTrig         : R_TRIG;
    errTrig       : R_TRIG;
        hnd           : HANDLE;
    adrVar        : UDINT;
      END_VAR
  VAR_TEMP
    restSize      : UDINT;
```

```
      written      : UDINT;
      reqSize      : UDINT;
  mode         : TF_MODE;
END_VAR



adrVar := PTR_TO_UDINT( ADR( srcVar));
eTrig(CLK := exec);

// open file and seek position
     IF eTrig.Q THEN
      busy := TRUE; done := FALSE; err := FALSE; errID := 0; actSize := 0;
      errTrig( CLK := FALSE);
      IF seek = 0 THEN mode := F_WRITE; ELSE mode := F_APPEND; END_IF;
  hnd := FileOpen( fileName := fileName, mode := mode);
  IF hnd = INVALID_HANDLE_VALUE THEN
       err := TRUE; busy := FALSE;
      ELSE
       IF seek <> 0 AND seek <> 16#FFFF_FFFF THEN
        IF FileSetPos( hFile := hnd, offset := seek) = FALSE THEN
         err := TRUE; busy := FALSE;
        END_IF;
       END_IF;
      END_IF;
     END_IF;

// write data to file to variable (one sector per one ПЛК cycle)
     IF busy THEN
      IF size > actSize THEN
       restSize := size - actSize;
       IF restSize > 512 THEN reqSize := 512;          // more then one sector
                 ELSE reqSize := restSize;     // last sector
       END_IF;
    written := FileWrite(hFile := hnd, adrBuf := adrVar + actSize,
                         size   := reqSize);
       actSize := actSize + written;
    IF written <> reqSize THEN
        busy := FALSE; err := TRUE;
       ELSE
        IF actSize = size THEN
         done := TRUE; busy := FALSE;
        END_IF;
       END_IF;
      ELSE
       done := TRUE;
      END_IF;
     ELSE
      done := done AND exec;
      err  := err  AND exec;
     END_IF;

// set error code, if any
     errTrig( CLK := err);
     IF errTrig.Q THEN errID := GetLastErr(); END_IF;

// close file
     IF errTrig.Q OR done THEN
      IF hnd <> INVALID_HANDLE_VALUE THEN
       IF FileClose( hFile := hnd) THEN
        hnd := INVALID_HANDLE_VALUE;
```

```
        END_IF;
      END_IF;
    END_IF;

END_FUNCTION_BLOCK
```

6.3 Использование функциональных блоков ReadFromFile и WriteToFile

На следующем примере указано использование функциональных блоков ReadFromFile и WriteToFile. Программа в примере после повторного запуска отестирует систему на существование каталога файлов *данных*. Если он не существует, то создаст его. После этого в данном каталоге файлов создаст файл *test.txt* и, наконец, запишет в него данные, сохраненные в переменной *record*. После этого содержание *test.txt* будет записано в переменную *copy*, которая сопоставится с переменной *record*. Содержание обоих переменных должно быть одинаковым. Речь идет таким образом о простом тесте записи данных в файл и их считывании для проведения контроля. Результаты теста заданы переменной *test_OK* и, наконец, *test_ERR*. Причиной возникновения возможных осложнений является видеть потом в переменной *проблемму*.

```
TypE
  TRecord : STRUCT
        text   : STRING;
        data   : ARRAY[0..999] of USINT;
      END_STRUCT;
END_TypE

     VAR_GLOBAL
    test_OK   : bool;              // result
        test_ERR  : bool;                 // error flag
        record    : TRecord;           // written data
        copy      : TRecord;           // read data
END_VAR


FUNCTION InitRecord : BOOL
      VAR  i : UINT; END_VAR

  record.text := 'Begin of file data/test.txt ...';
      FOR i := 0 TO 999 DO  record.data[i] := UINT_TO_USINT(i); END_FOR;
      InitRecord := TRUE;
END_FUNCTION


PROGRAM Test_MMC_card
      VAR
        dir,
        read, init          : BOOL;
        write               : BOOL;
        fn                  : STRING := 'data/test.txt';
```

```
        dn                  : STRING := 'data/';
    WriteFile           : WriteToFile;
        ReadFile            : ReadFromFile;
    count               : UINT;
        problem             : STRING := 'No problem';
       END_VAR

  IF NOT init THEN
        init := InitRecord();
    dir := FileExists( fileName := dn);
    IF NOT dir THEN
         dir := DirCreate(dirName := dn);
         IF NOT dir THEN
          test_ERR := GetLastErrTxt( errCode := GetLastErr(),
                                     errMessage := problem);
      END_IF;
        END_IF;
       END_IF;


 IF dir THEN
        IF NOT WriteFile.err AND NOT ReadFile.err THEN
         IF NOT write THEN
          count := count + 1;
          WriteFile( fileName := fn, srcVar := void( record), exec := true,
                seek := 0, size := sizeof( record), done => write);
          IF WriteFile.err THEN
           test_ERR := GetLastErrTxt( errCode := WriteFile.errID,
                                      errMessage := problem);
        END_IF;
         END_IF;
         IF write AND NOT read THEN
          count := count + 1;
          ReadFile( fileName := fn, dstVar := void( copy),
               exec := true, seek := 0,
               size := sizeof( copy), done => read);
          IF ReadFile.err THEN
           test_ERR := GetLastErrTxt( errCode := ReadFile.errID,
                                      errMessage := problem);
        END_IF;
         END_IF;
        END_IF;

    IF write AND read THEN
         IF record = copy THEN test_OK :=  true; END_IF;
        END_IF;
       END_IF;
END_PROGRAM
```