Knihovna pro práci se soubory

TXV 003 41.01
páté vydání
červen 2011
změny vyhrazeny

Historie změn

Datum	Vydání	Popis změn
Leden 2008	1	První vydání
Březen 2008	2	Doplněn popis funkce DiskInfo
Srpen 2009	3	Opraveno jméno knihovny v helpu Mosaic Doplněn popis datového typu TdiskInfo, funkce FindFreeCluster a funkčních bloků ReadDbxFormFile, WriteDbxToFile a WriteToFileSeq
Říjen 2010	4	Opraven překlep v popisu funkce DirRead Opraven popis funkčního bloku WriteToFileSeq
Červen 2011	5	Doplněn popis funkce FileInfo() Popis odpovídá knihovně FileLib v1.8

OBSAH

1 ÚVOD

Knihovna FileLib obsahuje sadu funkcí potřebných pro práci se soubory. Tuto knihovnu lze použít pouze pro řídící systémy vybavené souborovým systémem.

Knihovna FileLib obsahuje základní (nízkoúrovňové) funkce pro práci se soubory a adresáři, dále potřebné datové typy a konstanty, a konečně funkční bloky pro čtení a zápis souborů. Funkční bloky využívají základní souborové funkce a rozkládají čtení resp. zápis souborů na více cyklů tak, aby se optimalizovala časová náročnost těchto operací.

Následují obrázek ukazuje strukturu knihovny FileLib v prostředí Mosaic.

Pokud chceme funkce z knihovny FileLib použít v aplikačním programu PLC, je třeba nejprve přidat tuto knihovnu do projektu. Knihovna je dodávaná jako součást instalace prostředí Mosaic od verze 2.6.0. Centrální jednotka PLC systému musí mít implementovanou podporu pro souborové operace.

Struktura adresářů

Kořenový adresář pro souborové operace v PLC systému se jmenuje ROOT. Programátor PLC systému může pracovat pouze s těmi soubory a adresáři, které jsou umístěny v adresáři ROOT. Ostatní soubory a adresáře nejsou z PLC programu dostupné. Adresář ROOT je tedy pracovním adresářem programátora PLC.

Jména souborů

Souborový systém podporuje jména souborů v konvenci DOS 8.3. Jméno souboru se skládá z vlastního jména souboru (maximálně 8 znaků) a přípony (maximálně 3 znaky). Tyto dvě části jsou odděleny tečkou. Ve jménech souborů nelze používat interpukční znaky, mezery a znaky " * "," ? ". Znaky národních abeced nejsou ve jménech podporovány. Velká a malá písmena ve jménech souborů nejsou rozlišována. Zástupné znaky ve jménech souborů (např. *.*) nejsou podporovány.

Cesta k souboru

Cesta k souboru je určení polohy souboru na disku vzhledem k adresáři ROOT. Cesta tedy obsahuje jména adresářů, ve kterých je soubor uložen. Pro jména adresářů v cestě platí stejná pravidla jako pro jméno souboru. Jednotlivá jména adresářů v cestě jsou oddělena znakem lomítko „/“. Souborový systém PLC podporuje pouze absolutní cesty. Relativní cesty cesty ani změna pracovního adresáře nejsou podporovány.

Maximální délka jména souboru včetně cesty je omezena na 65 znaků (viz konstanta *MAX_PATH*).

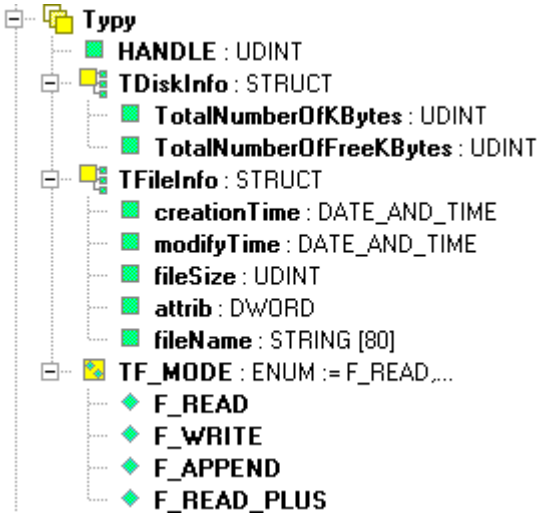
Viditelnost souborů pro web server PLC

Web server PLC používá jako kořenový adresář ROOT/WWW. Pokud mají být soubory dostupné přes web rozhraní, musí být uloženy v této cestě (např. v adresáři ROOT/WWW/DATA).

Podrobnosti o typu podporovaného souborového systému jsou uvedeny popisu příslušných centrálních jednotek PLC.

2 DATOVÉ TYPY

V knihovně FileLib jsou definovány následující datové typy:



Stručný popis datových typů udává následující tabulka:

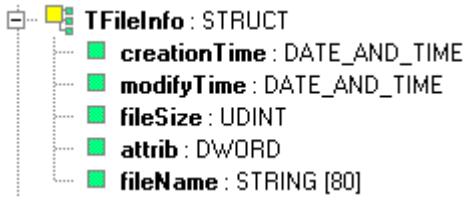
Identifikátor	*Typ*	*Význam*
HANDLE	UDINT	Identifikátor souboru
TDiskInfo	STRUCT	Informace o disku v PLC (SD/MMC karta)
TFileInfo	STRUCT	Informace o souboru
TF_MODE	ENUM	Konstanta definující způsob otevření souboru

2.1 Typ HANDLE

HANDLE je datový typ odvozený ze základního typu UDINT a používá se pro identifikátor souboru. Každému otevřenému souboru je přidělen identifikátor, který se používá pro operace čtení ze souboru, zápis do souboru, uzavření souboru atd.

2.2 Typ TFileInfo

TFileInfo je struktura, kterou vrací funkce *DirOpen, DirRead a FileInfo* a obsahuje informace o souboru.

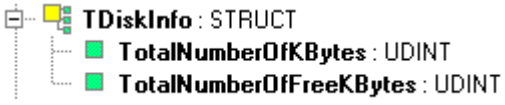


Význam jednotlivých položek struktury *TFileInfo* je následující:

- *creationTime* datum a cas vytvoření souboru
- *modifyTime* datum a cas poslední modifikace souboru
- *fileSize* velikost souboru v bytech
- *attrib* atributy souboru
- *fileName* jméno souboru

2.3 Typ TDiskInfo

TDiskInfo je struktura, kterou vrací funkce *DiskInfo* a obsahuje informace o celkové velikosti pevného disku v PLC (typicky SD nebo MMC katra) a velikosti volného místa na disku.

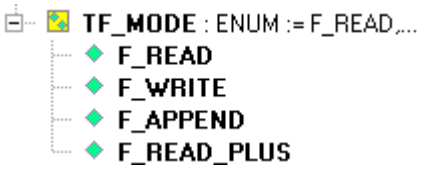


Význam jednotlivých položek struktury *TDiskInfo* je následující:

- *TotalNumberOfKBytes* celková velikost disku v KiloBytech
- *TotalNumberOfFreeKBytes* velikost volného místa na disku v KiloBytech

2.4 Typ TF_MODE

TF_MODE je výčtový typ, který se používá pro konstanty definující způsob otevření souboru funkcí *FileOpen*.

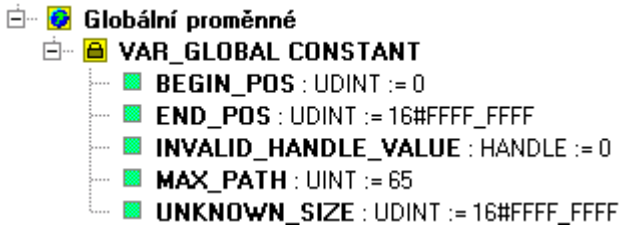


Význam jednotlivých položek výčtu je následující:

- *F_READ* otevřít soubor pro čtení
- *F_WRITE* otevřít soubor pro zápis, zápis dat bude od začátku souboru
- *F_APPEND* otevřít soubor pro zápis, data budou připojena na konec souboru
- *F_READ_PLUS* otevřít soubor pro čtení i pro zápis

3 KONSTANTY

V knihovně FileLib jsou definovány následující konstanty:



Identifikátor	*Typ*	*Hodnota*	*Význam*
BEGIN_POS	UDINT	0	Od začátku souboru
END_POS	UDINT	16#FFFF_FFFF	Od konce souboru
INVALID_HANDLE_VALUE	HANDLE	0	Neplatný identifikátor souboru
MAX_PATH	UINT	65	Maximální možná délka jména souboru (včetně cesty)
UNKNOWN_SIZE	UDINT	16#FFFF_FFFF	Neznámá velikost souboru

3.1 Konstanta BEGIN_POS

Konstanta *BEGIN_POS* je typu UDINT a používá se jako parametr funkce *FileSetPos* v případě, že chceme nastavit čtení nebo zápis od začátku souboru. Konstanta má hodnotu 0.

3.2 Konstanta END_POS

Konstanta *END_POS* je typu UDINT a používá se jako parametr funkce *FileSetPos* v případě, že chceme nastavit čtení nebo zápis od konce souboru. Konstanta má hodnotu 16#FFFF_FFFF.

3.3 Konstanta INVALID_HANDLE_VALUE

Konstanta *INVALID_HANDLE_VALUE* je typu HANDLE a používá se jako návratová hodnota funkce *FileOpen* resp. *DirOpen* v případě, že se nepodaří otevřít soubor resp. adresář. Konstanta má hodnotu 0 a znamená neplatný identifikátor souboru.

3.4 Konstanta MAX_PATH

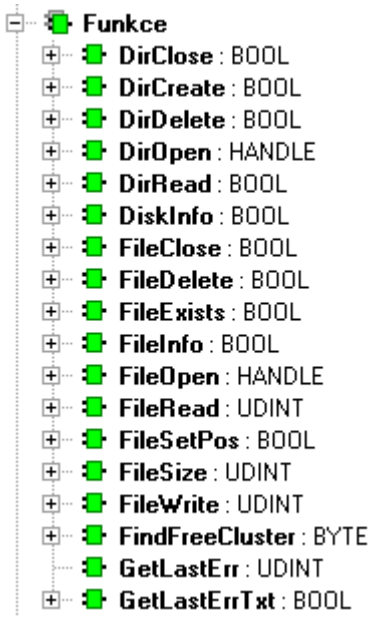
Konstanta *MAX_PATH* je typu UINT a udává maximální možnou délku jména souboru nebo adresáře (včetně cesty). Konstanta má hodnotu 65.

3.5 Konstanta UNKNOWN_SIZE

Konstanta *UNKNOWN_SIZE* je typu UDINT a používá se jako návratová hodnota funkce FileSize v případě, že se nepodaří zjistit velikost souboru. Konstanta má hodnotu 16#FFFF_FFFF.

4 FUNKCE PRO PRÁCI SE SOUBORY

Knihovna FileLib obsahuje následující funkce:



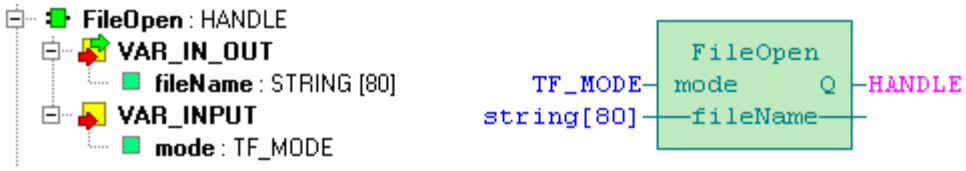
Stručný popis funkcí udává následující tabulka:

Funkce	***Popis***
DirOpen	Otevření adresáře
DirRead	Čtení z adresáře
DirClose	Uzavření adresáře
DirCreate	Vytvoření adresáře
DirDelete	Zrušení adresáře
FileOpen	Otevření souboru
FileRead	Čtení ze souboru
FileWrite	Zápis do souboru
FileClose	Uzavření souboru
FileExists	Otestování zda soubor existuje
FileInfo	Načíst informace o souboru
FileSize	Zjištení velikosti souboru
FileDelete	Zrušení souboru
FileSetPos	Nastavení pozice v souboru
FindFreeCluster	Najít volné místo na disku
GetLastErr	Zjistit kód poslední chyby
GetLastErrTxt	Převod kódu chyby na textové hlášení

Funkce	*Popis*
DiskInfo	Zjistit velikost disku a volné místo na disku

4.1 Funkce FileOpen

Knihovna : *FileLib*



Funkce **FileOpen** otevře soubor specifikovaný proměnnou *fileName.* Proměnná *mode* pak udává požadovaný způsob přístupu k souboru.

Funkce **FileOpen** vrátí identifikátor otevřeného souboru. Pokud se soubor otevřít nepodaří, funkce vrátí neplatný identifikátor (INVALID_HANDLE_VALUE).

Popis proměnných :

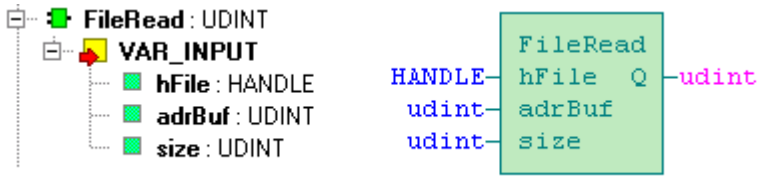
	Proměnná	*Typ*	*Význam*	
VAR_IN_OUT				
	fileName	STRING	Jméno souboru včetně cesty	
VAR_INPUT				
	mode	TF_MODE	Typ přístupu k souboru	
			F_READ	otevřít soubor pro čtení, soubor musí existovat
			F_WRITE	otevřít soubor pro zápis data budou zapisována od začátku souboru pokud soubor neexistuje funkce založí nový soubor pokud soubor existuje, obsah souboru je vymazán
			F_APPEND	otevřít soubor pro zápis data budou zapisována na konec souboru pokud soubor neexistuje funkce založí nový soubor pokud soubor existuje, data budou přidaná na konec souboru
FileOpen				
	Návratová hodnota	HANDLE	Identifikátor otevřeného souboru. Pokud se soubor nepodaří otevřít je vrácen neplatný identifikátor (INVALID_HANDLE_VALUE)	

Příklad programu s voláním funkce FileOpen :

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

Knihovna : *FileLib*



Funkce **FileRead** načte ze souboru určitý počet znaků a uloží je do paměti PLC. Soubor je specifikován proměnnou *hFile*. Proměná, do které budou načtena data uložena, je specifikována v *adrBuf*. A konečně proměnná *size* udává kolik znaků bude ze souboru přečteno.

Funkce **FileRead** vrátí počet skutečně načtených znaků.

Popis proměnných :

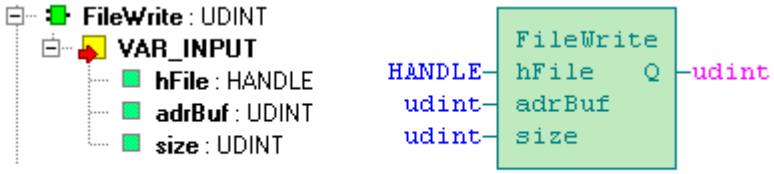
	Proměnná	*Typ*	*Význam*
VAR_INPUT			
	hFile	HANDLE	Identifikátor souboru
	adrBuf	UDINT	Adresa proměnné, do které budou načteny znaky ze souboru
	size	UDINT	Počet znaků, které mají být načteny
FileRead			
	Návratová hodnota	UDINT	Počet skutečně načtených znaků

Příklad programu s voláním funkce FileRead :

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

Knihovna : *FileLib*



Funkce **FileWrite** zkopíruje obsah proměnné PLC do souboru. Soubor je specifikován proměnnou *hFile*. Proměná, ze které budou data kopírována, je specifikována v *adrBuf*. A konečně proměnná *size* udává kolik znaků bude zapsáno do souboru.

Funkce **FileWrite** vrátí počet skutečně zapsaných znaků.

Popis proměnných :

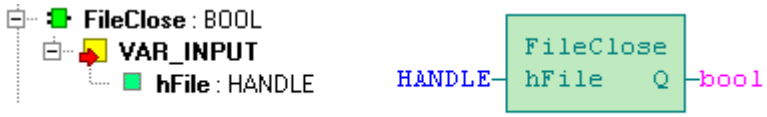
	Proměnná	***Typ***	***Význam***
VAR_INPUT			
	hFile	HANDLE	Identifikátor souboru
	adrBuf	UDINT	Adresa proměnné, jejíž obsah bude zapsán do souboru
	size	UDINT	Počet zapisovaných znaků
FileWrite			
	Návratová hodnota	UDINT	Počet skutečně zapsaných znaků

Příklad programu s voláním funkce FileWrite :

```
PROGRAM ExampleFileWrite
  VAR
    start  : BOOL;
    fname  : STRING := 'DATA/log.txt';
    hf     : HANDLE;                // file handle
    buffer : STRING[200];
    lenght : UDINT;
    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 Funkce FileClose

Knihovna : *FileLib*



Funkce **FileClose** uzavře soubor specifikovaný proměnnou *hFile*. Všechny vnitřní buffery pro zápis jsou před uzavřením zapsány do souboru. Po uzavření souboru není možné ze souboru číst nebo do něho zapisovat. Identifikátor souboru přestane být asociován se souborem.

Funkce **FileClose** vrací hodnotu TRUE, pokud se soubor podaří uzavřít, v opačném případě vrací hodnotu FALSE. Příčinu případné chyby lze zjistit pomocí funkce **GetLastErr**.

Popis proměnných :

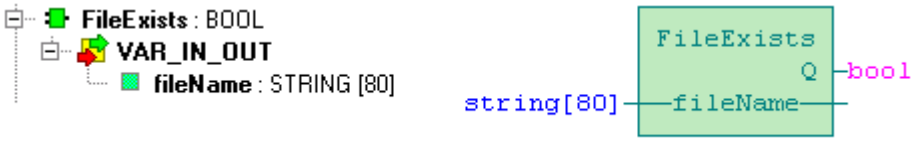
	Proměnná	*Typ*	*Význam*
VAR_INPUT			
	hFile	HANDLE	Identifikátor souboru
FileClose			
	Návratová hodnota	BOOL	TRUE pokud je soubor úspěšně uzavřen, FALSE v ostatních případech

Příklad programu s voláním funkce **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 Funkce FileExists

Knihovna : *FileLib*



Funkce **FileExists** otestuje jestli existuje soubor specifikovaný proměnnou *fileName.*

Funkce **FileExists** vrátí TRUE pokud soubor existuje. Pokud se soubor nepodaří najít, funkce vrátí FALSE.

Popis proměnných :

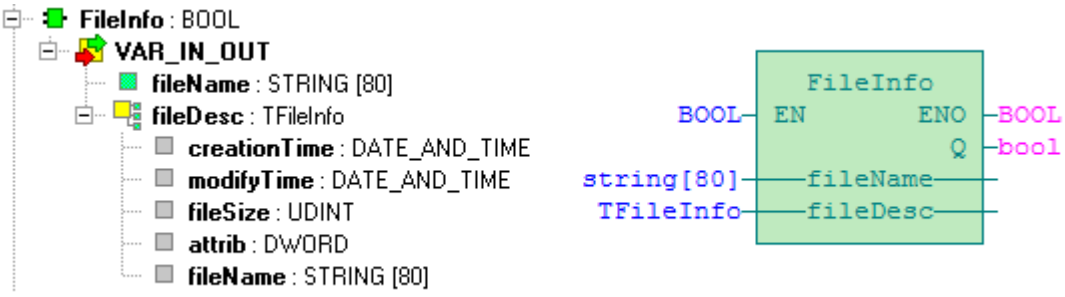
	Proměnná	***Typ***	***Význam***
VAR_IN_OUT			
	fileName	STRING	Jméno souboru včetně cesty
FileExists			
	Návratová hodnota	BOOL	TRUE pokud soubor existuje, FALSE v ostatních případech

Příklad programu s voláním funkce FileExists :

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

  IF start THEN
    start := FALSE;
    IF NOT del THEN
      fname := 'DATA/log.txt';
      IF FileExists( fileName := fname) THEN
        del := FileDelete( fileName := fname);
      END_IF;
    END_IF;
  END_IF;

END_PROGRAM
```

4.6 Funkce FileInfo

Knihovna : *FileLib*



Funkce **FileInfo** načte informace o souboru, který je specifikovaný proměnnou *fileName*. Načtené informace jsou uloženy do proměnné *fileDesc*.

Funkce **FileInfo** vrátí TRUE pokud soubor existuje a podaří se získat všechny požadované informace. Pokud se to nepodaří, funkce vrátí FALSE a proměnná *fileDesc* je vynulovaná.

Popis proměnných :

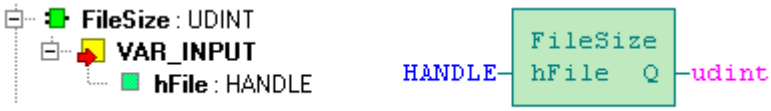
	Proměnná	*Typ*	*Význam*
VAR_IN_OUT			
	fileName	STRING	Jméno souboru včetně cesty
	fileDesc	TFileInfo	Struktura s informacemi o souboru
	.creationTime	DT	Datum a čas vytvoření souboru
	.modifyTime	DT	Datum a čas poslední modifikace souboru
	.fileSize	UDINT	Velikost souboru (počet bytů)
	.attrib	DWORD	Atributy souboru
	.fileName	STRING	Jméno souboru
FileInfo			
	Návratová hodnota	BOOL	TRUE pokud se podaří zjistit informace o souboru, FALSE v ostatních případech

Příklad programu s voláním funkce FileInfo :

```
PROGRAM ExampleFileInfo
  VAR
    fname  : STRING := 'DATA/log.txt';
    fdesc  : TFileInfo;
    result : BOOL;
  END_VAR

  result := FileInfo( fileName := fname, fileDesc := fdesc);
END_PROGRAM
```

4.7 Funkce FileSize

Knihovna : *FileLib*



Funkce **FileSize** zjistí velikost souboru specifikovaného proměnnou *hFile*.

Funkce **FileSize** vrací velikost souboru v bytech. Pokud dojde při zjišťování velikosti souboru k chybě, funkce vrátí UKNOWN_SIZE. Příčinu chyby lze zjistit pomocí funkce **GetLastErr**.

Popis proměnných :

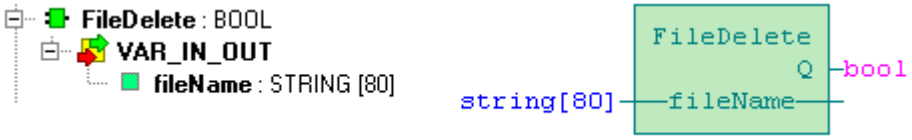
	Proměnná	*Typ*	*Význam*
VAR_INPUT			
	hFile	HANDLE	Identifikátor souboru
FileSize			
	Návratová hodnota	UDINT	Velikost souboru Při chybě vrací UKNOWN_SIZE, tj. 16#FFFF_FFFF

Příklad programu s voláním funkce 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
```

4.8 Funkce FileDelete

Knihovna : *FileLib*



Funkce **FileDelete** smaže soubor specifikovaný proměnnou *fileName.*

Tuto funkci lze použít i pro smazání adresáře. Pro úspěšné smazání adresáře je nezbytné, aby byl adresář prázdný.

Funkce **FileDelete** vrátí TRUE pokud se soubor podaří smazat. Jinak funkce vrátí FALSE. Příčinu případné chyby lze zjistit pomocí funkce **GetLastErr**.

Popis proměnných :

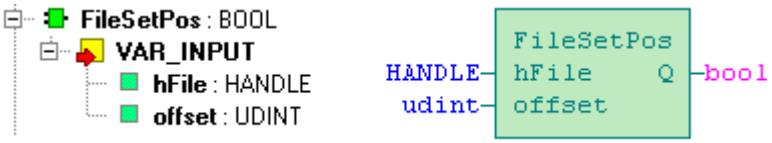
	Proměnná	***Typ***	***Význam***
VAR_IN_OUT			
	fileName	STRING	Jméno souboru včetně cesty
FileDelete			
	Návratová hodnota	BOOL	TRUE pokud je soubor smazán, FALSE v ostatních případech

Příklad programu s voláním funkce 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
```

4.9 Funkce FileSetPos

Knihovna : *FileLib*



Funkce **FileSetPos** umožňuje nastavit pozici pro čtení ze souboru resp. pro zápis do souboru. Soubor je specififován proměnnou *hFile*. Požadovanou pozici v souboru udává proměnná *offset*.

Funkce **FileSetPos** vrací hodnotu TRUE, pokud se podaří nastavit požadovanou pozici. V opačném případě vrací hodnotu FALSE. Příčinu případné chyby lze zjistit pomocí funkce **GetLastErr**.

Popis proměnných :

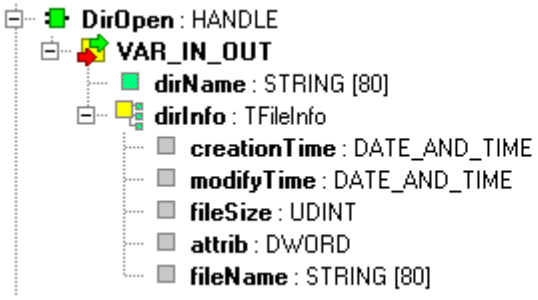
	Proměnná	***Typ***	***Význam***	
VAR_INPUT				
	hFile	HANDLE	Identifikátor souboru	
	offset	UDINT	Pozice v souboru	
			0 nebo BEGIN_POS	Na začátek souboru
			0 > offset > END_POS	Na zadanou pozici
			16#FFFF_FFFF nebo END_POS	Na konec souboru
FileSetPos				
	Návratová hodnota	BOOL	Vrací TRUE, pokud se podaří nastavit žádanou pozici v souboru Jinak vrací FALSE	

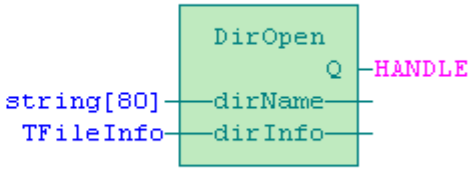
Příklad programu s voláním funkce FileSetPos:

```
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
```

4.10 Funkce DirOpen

Knihovna : *FileLib*





Funkce **DirOpen** otevře adresář specifikovaný proměnnou *dirName*. Poté zjistí informace o prvním souboru v adresáři a tyto informace zapíše do proměnné *dirInfo*. Informace o dalších souborech v adresáři lze zjistit funkcí DirRead.

Funkce **DirOpen** vrátí identifikátor otevřeného adresáře. Pokud se soubor otevřít nepodaří, funkce vrátí neplatný identifikátor (INVALID_HANDLE_VALUE).

Popis proměnných :

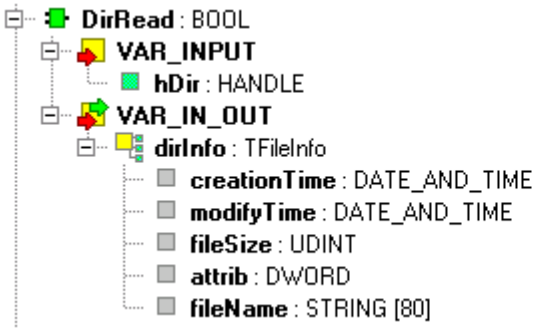
	Proměnná	***Typ***	***Význam***
VAR_IN_OUT			
	dirName	STRING	Jméno adresáře včetně cesty
	dirInfo	TFileInfo	Struktura s informacemi o prvním souboru v adresáři
	.creationTime	DT	Datum a čas vytvoření souboru
	.modifyTime	DT	Datum a čas poslední modifikace souboru
	.fileSize	UDINT	Velikost souboru (počet bytů)
	.attrib	DWORD	Atributy souboru
	.fileName	STRING	Jméno souboru
DirOpen			
	Návratová hodnota	HANDLE	Identifikátor otevřeného adresáře. Pokud se adresář nepodaří otevřít je vrácen neplatný identifikátor (INVALID_HANDLE_VALUE)

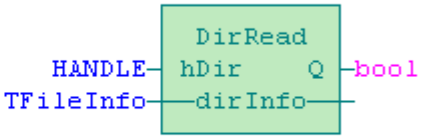
Příklad programu s voláním funkce DirOpen:

```
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
```

4.11 Funkce DirRead

Knihovna : *FileLib*





Funkce **DirRead** zjistí informace o dalším souboru v adresáři a tyto informace zapíše do proměnné *dirInfo*. Adresář je specifikován proměnnou *dirHandle*. Před voláním této funkce musí být adresář otevřen funkcí DirOpen.

Funkce **DirRead** vrátí TRUE pokud operace dopadla úspěšně. V opačném případě vrátí hodnotu FALSE.

Popis proměnných :

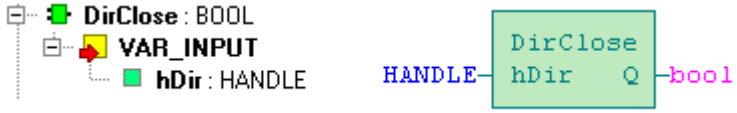
	Proměnná	***Typ***	***Význam***
VAR_INPUT			
	dirHandle	HANDLE	Identifikátor adresáře
VAR_IN_OUT			
	dirInfo	TFileInfo	Struktura s informacemi o dalším souboru v adresáři
	.creationTime	DT	Datum a čas vytvoření souboru
	.modifyTime	DT	Datum a čas poslední modifikace souboru
	.fileSize	UDINT	Velikost souboru (počet bytů)
	.attrib	DWORD	Atributy souboru
	.fileName	STRING	Jméno souboru
DirRead			
	Návratová hodnota	BOOL	TRUE pokud operace dopadne úspěšně, FALSE v ostatních případech

Příklad programu s voláním funkce 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
```

4.12 Funkce DirClose

Knihovna : *FileLib*



Funkce **DirClose** uzavře adresář specifikovaný proměnnou *hDir*. Po uzavření adresáře není možné z adresáře číst funkcí ReadDir. Identifikátor adresáře přestane být asociován se souborem.

Funkce **DirClose** vrací hodnotu TRUE, pokud se adresář podaří uzavřít, v opačném případě vrací hodnotu FALSE. Příčinu případné chyby lze zjistit pomocí funkce **GetLastErr**.

Popis proměnných :

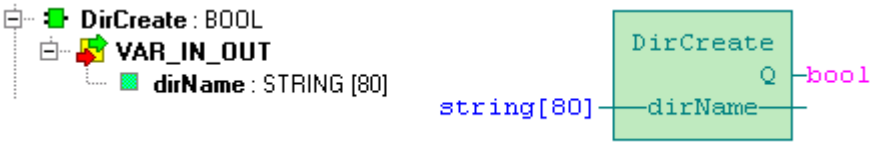
	Proměnná	*Typ*	*Význam*
VAR_INPUT			
	hFile	HANDLE	Identifikátor souboru
DirClose			
	Návratová hodnota	BOOL	TRUE pokud je adresář úspěšně uzavřen, FALSE v ostatních případech

Příklad programu s voláním funkce 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
```

4.13 Funkce DirCreate

Knihovna : *FileLib*



Funkce **DirCreate** vytvoří adresář specifikovaný proměnnou *dirName*.

Funkce **DirCreate** vrací hodnotu TRUE, pokud se adresář podaří vytvořit, v opačném případě vrací hodnotu FALSE. Příčinu případné chyby lze zjistit pomocí funkce **GetLastErr**.

Popis proměnných :

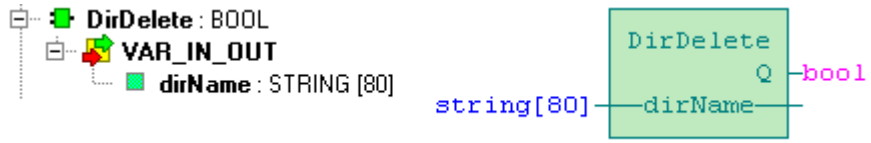
	Proměnná	*Typ*	*Význam*
VAR_IN_OUT			
	dirName	STRING	Jméno souboru vřetně cesty
DirCreate			
	Návratová hodnota	BOOL	TRUE pokud je adresář úspěšně vytvořen, FALSE v ostatních případech

Příklad programu s voláním funkce 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
```

4.14 Funkce DirDelete

Knihovna : *FileLib*



Funkce **DirDelete** smaže adresář specifikovaný proměnnou *dirName*. Pro úspěšné smazání adresáře je nezbytné, aby byl adresář prázdný (aby neobsahoval žádné soubory).

Funkce **DirDelete** vrátí TRUE pokud se adresář podaří smazat. Jinak funkce vrátí FALSE. Příčinu případné chyby lze zjistit pomocí funkce **GetLastErr**.

Popis proměnných :

	Proměnná	*Typ*	*Význam*
VAR_IN_OUT			
	fileName	STRING	Jméno adresáře včetně cesty
DirDelete			
	Návratová hodnota	BOOL	TRUE pokud je adresář smazán, FALSE v ostatních případech

Příklad programu s voláním funkce 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
```

4.15 Funkce GetLastErr

Knihovna : *FileLib*

Funkce **GetLastErr** vrátí kód poslední zaznamenané chyby vzniklé při souborové operaci. Tento kód může být použit jako parametr funkce GetLastErrTxt, která pak vrátí textový popis chyby.

Popis proměnných :

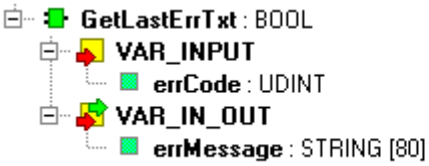
	Proměnná	*Typ*	*Význam*
GetLastErr			
	Návratová hodnota	UDINT	Kód poslední chyby při souborové operaci

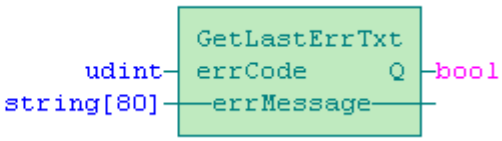
Příklad programu s voláním funkce GetLastErr:

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

  fname := 'DATA/log1.txt';
  IF NOT FileExists( fileName := fname) THEN
    error := GetLastErr();
    valid := GetLastErrTxt(errCode := error, errMessage := errTxt);
  END_IF;

END_PROGRAM
```

4.16 Funkce GetLastErrTxt

Knihovna : *FileLib*





Funkce **GetLastErrTxt** zapíše textový popis chyby do proměnné *errMessage*. Textový popis chyby odpovídá kódu chyby, který je specifikován proměnnou *errCode*.

Funkce **GetLastErrTxt** vrací TRUE, pokud k zadanému kódu chyby existuje textový popis. V opačném případě vrací FALSE.

Popis proměnných :

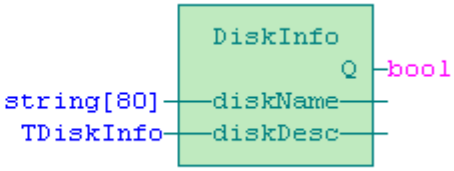
	Proměnná	*Typ*	*Význam*
VAR_INPUT			
	errCode	UDINT	Kód chyby
VAR_IN_OUT			
	errMessage	STRING	Textový popis chyby
GetLastErrTxt			
	Návratová hodnota	BOOL	Vrací TRUE, pokud existuje popis chyby Jinak vrací FALSE

Příklad programu s voláním funkce GetLastErrTxt:

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

  fname := 'DATA/log1.txt';
  IF NOT FileExists( fileName := fname) THEN
    error := GetLastErr();
    valid := GetLastErrTxt(errCode := error, errMessage := errTxt);
  END_IF;

END_PROGRAM
```

4.17 Funkce DiskInfo

Knihovna : *FileLib*



Funkce **DiskInfo** zjistí celkovou velikost disku a volné místo na disku. Jméno disku musí být uvedeno v proměnné diskName. Je-li tato proměnná prázdná, funkce **DiskInfo** vrátí informace o disku, který je nastaven jako default (např. v systémech Foxtrot je to paměťová karta).

Funkce **DiskInfo** vrací TRUE, pokud se podaří zjistit informace o zadaném disku.V opačném případě vrací FALSE.

Funkce **DiskInfo** naplní položky *TotalNumberOfKBytes* a *TotalNumberOfFreeKBytes* v proměnné *diskDesc*. Obě hodnoty jsou v KiloBytech.

Popis proměnných :

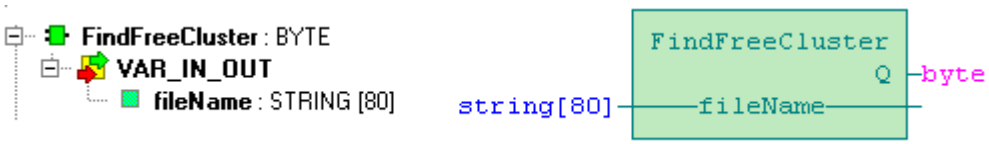
	Proměnná	***Typ***	***Význam***
VAR_IN_OUT			
	diskName	STRING	Název disku
	diskDesc	TDiskInfo	Informace o disku
	. *TotalNumberOfKBytes*	UDINT	Celková velikost disku v KB
	. *TotalNumberOfFreeKBytes*	UDINT	Volné místo na disku v KB
DiskInfo			
	Návratová hodnota	BOOL	Vrací TRUE, pokud se podaří zjistit informace o disku. Jinak vrací FALSE

Příklad programu s voláním funkce DiskInfo:

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

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

4.18 Funkce FindFreeCluster

Knihovna : *FileLib*



Funkce **FindFreeCluster** najde volné místo na disku. Vstupem funkce je jméno souboru, který bude posléze zapsán na disk. Účelem této funkce je zrychlit následný zápis dat do souboru a rozložit časově náročnou operaci vyhledání volného místa do více cyklů PLC. Tato funkce je využívána např. funkčním blokem **CreatePath**.

Funkce **FindFreeCluster** vrací hodnotu 0, pokud se podaří najít volné místo na disku. Protože operace hledání volného místa může být časově náročná (zejména při vyšších kapacitách disku) je doba hledání volného místa omezena na max. 10 ms. Pokud se do 10-ti ms nenajde volné místo, funkce **FindFreeCluster** vrátí hodnotu 13 a v hledání je možné pokračovat v dalším cyklu PLC. V případě chyby vrací funkce číslo chyby, které je různé od 0 a 13 (např. je-li disk plný nebo není dostupný, atd).

Popis proměnných :

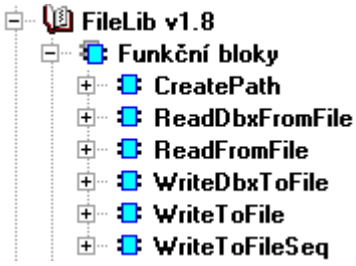
	Proměnná	***Typ***	***Význam***
VAR_IN_OUT			
	fileName	STRING	Jméno souboru
FindFreeCluster			
	Návratová hodnota	BYTE	Vrací 0 pokud je volné místo nalezeno nebo 13 pokud se v hledání má pokračovat v příštím cyklu PLC nebo jiné číslo pokud dojde k nějaké chybě.

Příklad programu s voláním funkce FindFreeCluster:

```
PROGRAM ExampleFindFreeCluster
  VAR
    path   : STRING := 'NEWDIR/';
    res    : BYTE;
    ok     : BOOL;
    err    : BOOL;
    busy   : BOOL;
  END_VAR

  IF NOT err THEN
    IF NOT FileExists( fileName := path) THEN
      busy := TRUE;
      res := FindFreeCluster(fileName := path);
      IF res = 0 THEN
        ok := DirCreate( dirName := path);
        IF NOT ok THEN
          err := TRUE; busy := FALSE;          // dir was not created
        END_IF;
      ELSE
        IF res <> 13 THEN
          err := TRUE; busy := FALSE;          // error disk full or not mounted
        END_IF;
      END_IF;
    ELSE
      ok := 1;
    END_IF;
  END_IF;

END_PROGRAM
```

5 FUNKČNÍ BLOKY PRO PRÁCI SE SOUBORY

Knihovna FileLib obsahuje následující funkční bloky pro práci se soubory:



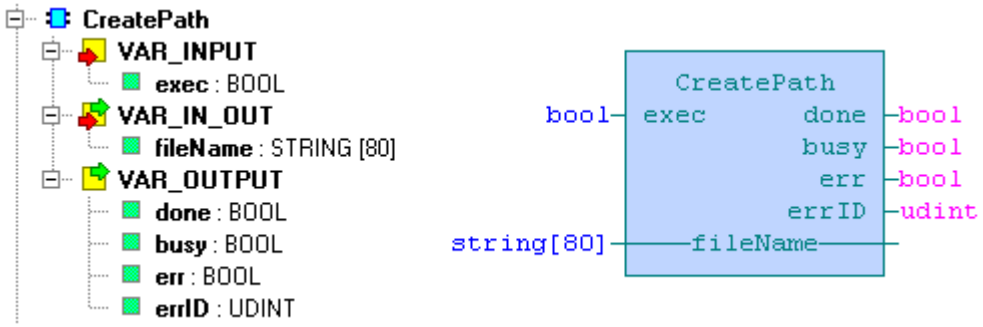
Stručný popis funkčních bloků udává následující tabulka:

Funkční blok	***Popis***
CreatePath	Zkontrolovat případně vytvořit cestu
ReadFromFile	Čtení dat ze souboru
WriteToFile	Zápis dat do souboru
ReadDbxFromFile	Čtení dat ze souboru do DataBoxu
WriteDbxToFile	Zápis dat z DataBoxu do souboru
WriteToFileSeq	Postupné ukládání dat do souboru

Funkční bloky využívají základní souborové funkce a rozkládají čtení resp. zápis souborů na více cyklů tak, aby se optimalizovala časová náročnost těchto operací.

5.1 Funkční blok CreatePath

Knihovna : *FileLib*



Funkční blok **CreatePath** zkontroluje zadanou cestu a pokud neexistuje, tak ji na disku vytvoří (založí potřebné adresáře). Jméno cesty udává proměnná *fileName*. Kontrola bude zahájena na náběžnou hranu proměnné *exec*.

Funkční blok **CreatePath** nastaví TRUE do proměnné *done* v okamžiku, kdy je zadaná cesta na disku dostupná. Během kontroly resp. vytváření cesty má proměnná *done* hodnotu FALSE a proměnná *busy* hodnotu TRUE. Pokud během práce nebyla detekována žádná chyba proměnná *err* má hodnotu FALSE, v případě chyby má hodnotu TRUE a v proměnné *errID* je uložen chybový kód. Ten může být použit jako vstupní proměnná funkce GetLastErrTxt pro získání textového popisu vzniklé chyby.

Popis proměnných :

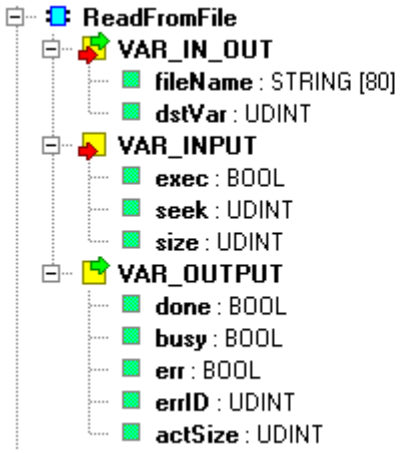
	Proměnná	***Typ***	***Význam***
VAR_IN_OUT			
	fileName	STRING	Název cesty (zakončený znakem /)
VAR_INPUT			
	exec	BOOL	Řídící proměnná. Náběžná hrana (přechod z hodnotu FALSE na hodnotu TRUE) zahájí kontrolu resp. vytváření cesty
VAR_OUTPUT			
	done	BOOL	Má hodnotu TRUE v okamžiku kdy je požadovaná cesta k dispozici Jinak vrací FALSE
	busy	BOOL	Má hodnotu TRUE během kontroly resp. vytváření cesty
	err	BOOL	Příznak chyby při vytváření cesty Pokud operace dopadla úspěšně má hodnotu FALSE, jinak TRUE.
	errID	UDINT	Chybový kód. Pokud operace dopadla úspěšně, errID = 0. V případě chyby errID <> 0.

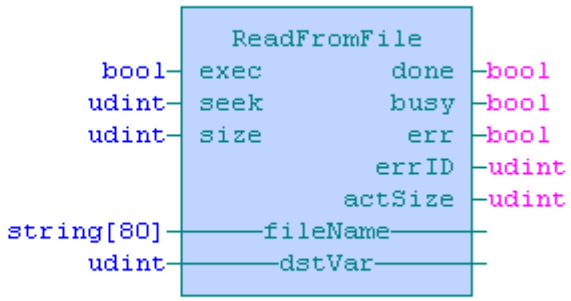
Následující příklad ukazuje použití funkčního bloku **CreatePath** pro vytvoření cesty včetně případného založení potřebných adresářů.

```
PROGRAM ExampleCreatePath
  VAR
    path               : STRING := 'DATA/2009/JUNE/';
    CheckAndCreatePath : CreatePath;
    pathOk             : BOOL;
    pathErr            : BOOL;
    txtErr             : STRING;
  END_VAR

  IF NOT pathOk AND NOT pathErr THEN
    CheckAndCreatePath( exec := TRUE, fileName := path);
    IF CheckAndCreatePath.done THEN
      pathOk := TRUE;
    ELSE
      IF CheckAndCreatePath.err THEN
        pathErr := TRUE;
       GetLastErrTxt(errCode := CheckAndCreatePath.errID, errMessage := txtErr);
      END_IF;
    END_IF;
  END_IF;

END_PROGRAM
```

5.2 Funkční blok ReadFromFile

Knihovna : *FileLib*





Funkční blok **ReadFromFile** přečte data ze souboru a uloží je do proměnné v paměti PLC. Jméno souboru, ze kterého se bude číst, udává proměnná *fileName*. Adresa proměnné, do které budou data uložena, je specifikována proměnnou *dstVar*. Čtení ze souboru bude zahájeno na náběžnou hranu proměnné *exec*. Proměnná *seek* udává pozici od začátku souboru, od které budou data načítána. Čtení dat se ukončí pokud jsou načtena všechna požadovaná data, jejichž velikost udává proměnná *size* nebo pokud je při čtení dosaženo konce souboru.

Funkční blok **ReadFromFile** nastaví TRUE do proměnné *done* v okamžiku, kdy se načte poslední blok dat ze souboru. Během načítání dat má proměnná *done* hodnotu FALSE a proměnná *busy* hodnotu TRUE. Počet skutečně načtených bytů udává proměnná *actSize*. Pokud bylo čtení bez chyby, proměnná *err* má hodnotu FALSE, v případě chyby má hodnotu TRUE a v proměnné *errID* je uložen chybový kód. Ten může být použit jako vstupní proměnná funkce GetLastErrTxt pro získání textového popisu vzniklé chyby.

Popis proměnných :

	Proměnná	***Typ***	***Význam***
VAR_IN_OUT			
	fileName	STRING	Jméno souboru včetně cesty
	dstVar	UDINT	Adresa proměnné, do které budou uložena data přečtená ze souboru
VAR_INPUT			
	exec	BOOL	Řídící proměnná. Náběžná hrana (přechod z hodnotu FALSE na hodnotu TRUE) zahájí čtení ze souboru
	seek	UDINT	Offset od začátku souboru, od kterého je čtení zahájeno 0 Od začátku souboru Seek <> 0 Od zadané pozice
	size	UDINT	Požadovaná velikost čtených dat v bytech

	Proměnná	*Typ*	*Význam*
VAR_OUTPUT			
	done	BOOL	Má hodnotu TRUE v okamžiku dokončení čtení ze souboru Jinak vrací FALSE
	busy	BOOL	Má hodnotu TRUE během čtení dat ze souboru
	err	BOOL	Příznak chyby při čtení ze souboru Pokud operace dopadla úspěšně má hodnotu FALSE, jinak TRUE.
	errID	UDINT	Chybový kód. Pokud operace dopadla úspěšně, errID = 0. V případě chyby errID <> 0.
	actSize	UDINT	Počet skutečně načtených bytů

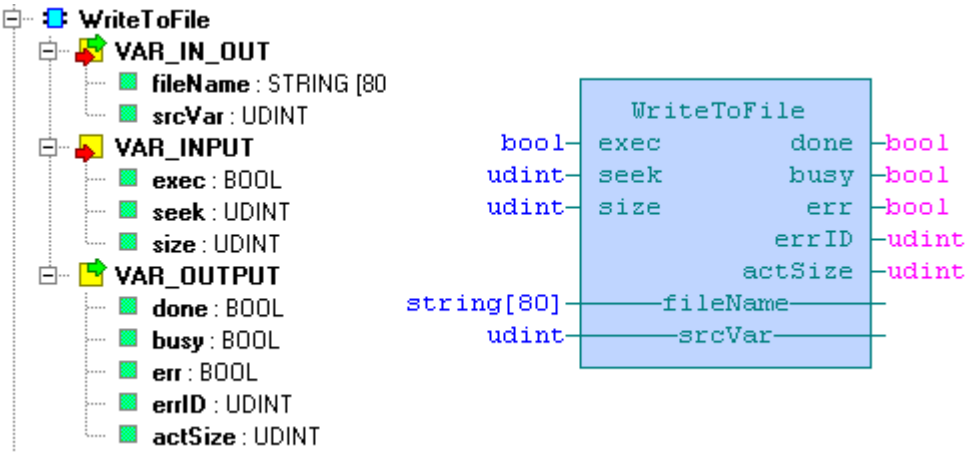
Následující příklad ukazuje definici funkčního bloku *ReadFromFile* v jazyce ST, která je zde uvedena pro možnost detilního pochopení funkce. Použití tohoto bloku pak ukazuje příklad v kapitole 6.

```
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 data from file to variable (one sector per one PLC 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
```

5.3 Funkční blok WriteToFile

Knihovna : *FileLib*



Funkční blok **WriteToFile** zapíše obsah proměnné PLC do souboru. Jméno souboru, do kterého se bude zapisovat, udává proměnná *fileName*. Pokud soubor neexistuje, bude založen. Adresa proměnné, jejíž obsah bude zapsán do souboru, je specifikována proměnnou src*Var*. Zápis do souboru bude zahájen na náběžnou hranu proměnné *exec*. Proměnná *seek* udává pozici od začátku souboru, od které budou data zapsaná. Zápis dat se ukončí pokud jsou zapsána všechna požadovaná data, jejichž velikost udává proměnná *size*.

Funkční blok **WriteToFile** nastaví TRUE do proměnné *done* v okamžiku, kdy se zapíše poslední blok dat do souboru. Během zápisu dat má proměnná *done* hodnotu FALSE a proměnná *busy* hodnotu TRUE. Počet skutečně zapsaných bytů udává proměnná *actSize*. Pokud byl zápis do souboru bez chyby, proměnná *err* má hodnotu FALSE, v případě chyby má hodnotu TRUE a v proměnné *errID* je uložen chybový kód. Ten může být použit jako vstupní proměnná funkce GetLastErrTxt pro získání textového popisu vzniklé chyby.

Popis proměnných :

	Proměnná	***Typ***	***Význam***
VAR_IN_OUT			
	fileName	STRING	Jméno souboru včetně cesty
	srcVar	UDINT	Adresa proměnné, jejíž obsah bude zapsán souboru
VAR_INPUT			
	exec	BOOL	Řídící proměnná. Náběžná hrana (přechod z hodnotu FALSE na hodnotu TRUE) zahájí zápis do souboru
	seek	UDINT	Offset v souboru, od kterého je zápis zahájen 0 Od začátku souboru 0 < seek < 16#FFFF_FFFF Od zadané pozice 16#FFFF_FFFF Přidat data na konec souboru
	size	UDINT	Požadovaná velikost zapisovaných dat v bytech

	Proměnná	*Typ*	*Význam*
VAR_OUTPUT			
	done	BOOL	Má hodnotu TRUE v okamžiku dokončení zápisu do souboru. Jinak vrací FALSE
	busy	BOOL	Má hodnotu TRUE během zápisu do souboru
	err	BOOL	Příznak chyby při zápisu do souboru Pokud operace dopadla úspěšně má hodnotu FALSE, jinak TRUE.
	errID	UDINT	Chybový kód. Pokud operace dopadla úspěšně, errID = 0. V případě chyby errID <> 0.
	actSize	UDINT	Počet skutečně zapsaných bytů

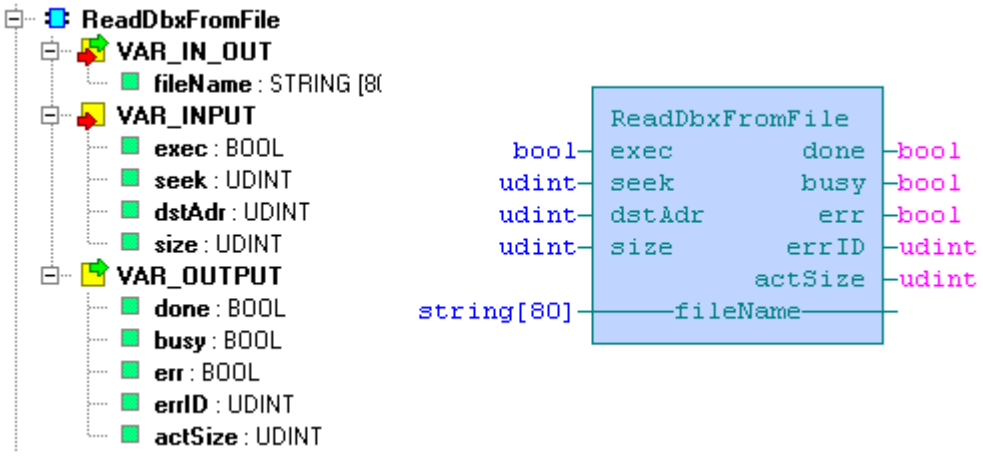
Následující příklad ukazuje definici funkčního bloku *WriteToFile* v jazyce ST, která je zde uvedena pro možnost detailního pochopení funkce. Použití tohoto bloku pak ukazuje příklad v kapitole 6.

```
FUNCTION_BLOCK WriteToFile
(*
  Copy data from variable to file.
  If file does not exist new file is created.
  If file exists file content is overwritten.
*)
  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 PLC 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
```

5.4 Funkční blok ReadDbxFromFile

Knihovna : *FileLib*



Funkční blok **ReadDbxFromFile** přečte data ze souboru a uloží je do paměti DataBox. Jméno souboru, ze kterého se bude číst, udává proměnná *fileName*. Adresa v DataBoxu, kam budou data uložena, je specifikována proměnnou *dstAdr*. Čtení ze souboru bude zahájeno na náběžnou hranu proměnné *exec*. Proměnná *seek* udává pozici od začátku souboru, od které budou data načítána. Čtení dat se ukončí pokud jsou načtena všechna požadovaná data, jejichž velikost udává proměnná *size* nebo pokud je při čtení dosaženo konce souboru.

Funkční blok **ReadDbxFromFile** nastaví TRUE do proměnné *done* v okamžiku, kdy se načte poslední blok dat ze souboru. Během načítání dat má proměnná *done* hodnotu FALSE a proměnná *busy* hodnotu TRUE. Počet skutečně načtených bytů udává proměnná *actSize*. Pokud bylo čtení bez chyby, proměnná *err* má hodnotu FALSE, v případě chyby má hodnotu TRUE a v proměnné *errID* je uložen chybový kód. Ten může být použit jako vstupní proměnná funkce GetLastErrTxt pro získání textového popisu vzniklé chyby.

Tento funkční blok je v centrálních jednotkách řady K podporován od verze v4.6.

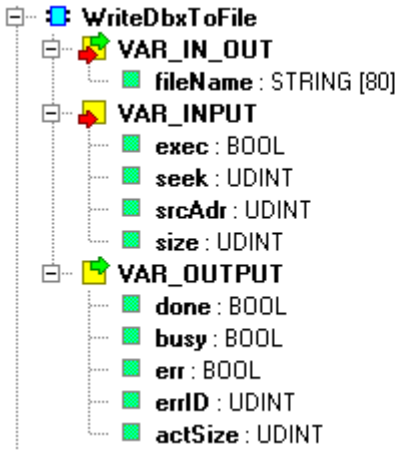
Popis proměnných :

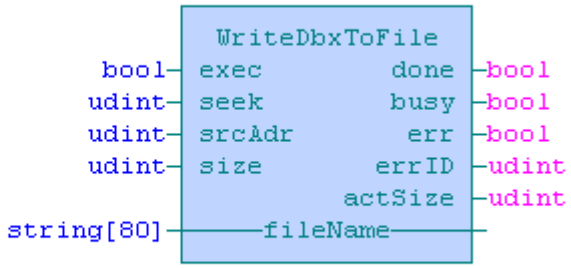
	Proměnná	*Typ*	*Význam*
VAR_IN_OUT			
	fileName	STRING	Jméno souboru včetně cesty
VAR_INPUT			
	exec	BOOL	Řídící proměnná. Náběžná hrana (přechod z hodnotu FALSE na hodnotu TRUE) zahájí čtení ze souboru
	seek	UDINT	Offset od začátku souboru, od kterého je čtení zahájeno 0 Od začátku souboru Seek <> 0 Od zadané pozice
	dstAdr	UDINT	Adresa DataBoxu, od které budou uložena data přečtená ze souboru
	size	UDINT	Požadovaná velikost čtených dat v bytech
VAR_OUTPUT			
	done	BOOL	Má hodnotu TRUE v okamžiku dokončení čtení ze souboru Jinak vrací FALSE
	busy	BOOL	Má hodnotu TRUE během čtení dat ze souboru
	err	BOOL	Příznak chyby při čtení ze souboru Pokud operace dopadla úspěšně má hodnotu FALSE, jinak TRUE.
	errID	UDINT	Chybový kód. Pokud operace dopadla úspěšně, errID = 0. V případě chyby errID <> 0.
	actSize	UDINT	Počet skutečně načtených bytů

Následující příklad ukazuje použití funkčního bloku *ReadDbxFromFile* v jazyce ST.

5.5 Funkční blok WriteDbxToFile

Knihovna : *FileLib*





Funkční blok **WriteDbxToFile** zapíše obsah paměti DataBox do souboru. Jméno souboru, do kterého se bude zapisovat, udává proměnná *fileName*. Pokud soubor neexistuje, bude založen. Adresa paměti DataBox, kde začínají ukládaná data, je specifikována proměnnou *srcAdr*. Zápis do souboru bude zahájen na náběžnou hranu proměnné *exec*. Proměnná *seek* udává pozici od začátku souboru, od které budou data zapsaná. Zápis dat se ukončí pokud jsou zapsána všechna požadovaná data, jejichž velikost udává proměnná *size.*

Funkční blok **WriteDbxToFile** nastaví TRUE do proměnné *done* v okamžiku, kdy se zapíše poslední blok dat do souboru. Během zápisu dat má proměnná *done* hodnotu FALSE a proměnná *busy* hodnotu TRUE. Počet skutečně zapsaných bytů udává proměnná *actSize*. Pokud byl zápis do souboru bez chyby, proměnná *err* má hodnotu FALSE, v případě chyby má hodnotu TRUE a v proměnné *errID* je uložen chybový kód. Ten může být použit jako vstupní proměnná funkce GetLastErrTxt pro získání textového popisu vzniklé chyby.

Tento funkční blok je v centrálních jednotkách řady K podporován od verze v4.6.

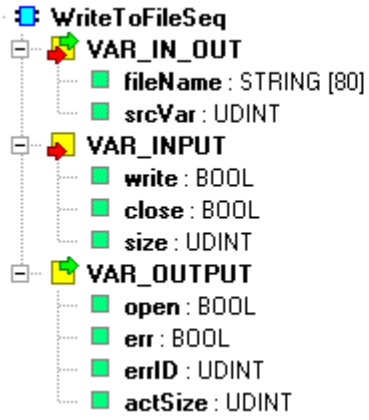
Popis proměnných :

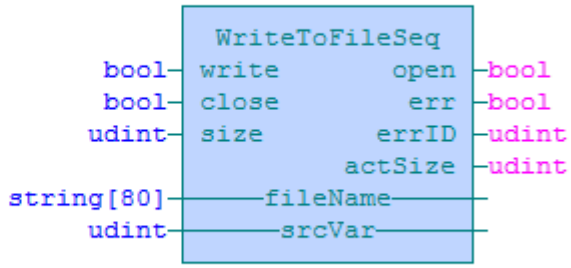
	Proměnná	*Typ*	*Význam*
VAR_IN_OUT			
	fileName	STRING	Jméno souboru včetně cesty
VAR_INPUT			
	exec	BOOL	Řídící proměnná. Náběžná hrana (přechod z hodnotu FALSE na hodnotu TRUE) zahájí zápis do souboru
	seek	UDINT	Offset v souboru, od kterého je zápis zahájen 0 Od začátku souboru 0 < seek < 16#FFFF_FFFF Od zadané pozice 16#FFFF_FFFF Přidat data na konec souboru
	srcAdr	UDINT	Adresa v DataBoxu, kde začínají data, která budou uložena do souboru
	size	UDINT	Požadovaná velikost zapisovaných dat v bytech
VAR_OUTPUT			
	done	BOOL	Má hodnotu TRUE v okamžiku dokončení zápisu do souboru. Jinak vrací FALSE
	busy	BOOL	Má hodnotu TRUE během zápisu do souboru
	err	BOOL	Příznak chyby při zápisu do souboru Pokud operace dopadla úspěšně má hodnotu FALSE, jinak TRUE.
	errID	UDINT	Chybový kód. Pokud operace dopadla úspěšně, errID = 0. V případě chyby errID <> 0.
	actSize	UDINT	Počet skutečně zapsaných bytů

Následující příklad ukazuje použití funkčního bloku *WriteDbxToFile* v jazyce ST.

5.6 Funkční blok WriteToFileSeq

Knihovna : *FileLib*





Funkční blok **WriteToFileSeq** zapíše obsah proměnné PLC do souboru při každém volání, kdy je proměnná *write* nastavena na hodnotu TRUE. Data jsou zapisována sekvenčně za sebe, až do doby než je soubor uzavřen nastavením vstupu *close* na hodnotu TRUE.

Jméno souboru, do kterého se bude zapisovat, udává proměnná *fileName*. Pokud soubor neexistuje, bude založen, pokud existuje bude přepsán. Jméno je akceptováno jen v případě, že soubor není ještě otevřen. Soubor se otevírá prvním zápisem a jeho stav je signalizován výstupem *open*. Adresa proměnné, jejíž obsah bude zapsán do souboru, je specifikována proměnnou *srcVar* a počet zapisovaných dat proměnnou *size*.

Funkční blok **WriteToFileSeq** signalizuje hodnotou TRUE v proměnné *open*, že je otevřen soubor pro zápis a bude jej nutné při nebo po posledním zápisu nastavením vstupu *close* zavřít.

Počet všech zapsaných bytů udává proměnná *actSize*. Pokud byl zápis do souboru bez chyby, proměnná *err* má hodnotu FALSE, v případě chyby má hodnotu TRUE a v proměnné *errID* je uložen chybový kód. Ten může být použit jako vstupní proměnná funkce GetLastErrTxt pro získání textového popisu vzniklé chyby.

Popis proměnných :

	Proměnná	***Typ***	***Význam***
VAR_IN_OUT			
	fileName	STRING	Jméno souboru včetně cesty
	srcVar	UDINT	Adresa proměnné, jejíž obsah bude zapsán souboru
VAR_INPUT			
	write	BOOL	Řídící proměnná. Při hodnotě TRUE je obsah proměnné zapsán do souboru.
	close	BOOL	Řídící proměnná. Při nastavení hodnoty TRUE je soubor uzavřen. Následující zápis se stejným jménem soubor přepíše!
	size	UDINT	Požadovaná velikost zapisovaných dat v bytech

	Proměnná	*Typ*	*Význam*
VAR_OUTPUT			
	open	BOOL	Má hodnotu TRUE pokud je aktuálně otevřen souboru pro zápis. Jinak vrací FALSE
	err	BOOL	Příznak chyby při zápisu do souboru Pokud operace dopadla úspěšně má hodnotu FALSE, jinak TRUE.
	errID	UDINT	Chybový kód. Pokud operace dopadla úspěšně, errID = 0. V případě chyby errID <> 0.
	actSize	UDINT	Počet všech zapsaných bytů

Následující příklad ukazuje použití bloku pro záznam dat přicházejících z komunikačního kanálu do souboru. V programu se využívá kromě této knihovny i knihovnu CommLib. Program zaznamenává všechna příchozí data z komunikačního kanálu pokud je proměnná *Record* nastavena na hodnotu TRUE.

```
PROGRAM ExampleWriteToFileSeq
  VAR
    Record          : BOOL;             //record data from CH1
    FileName        : STRING := 'CH1_COMM.TXT';
    iWriteToFileSeq : WriteToFileSeq;
    RecvFrom        : fbRecvFrom;
    Buffer          : ARRAY [0..379] OF USINT;
  END_VAR

  RecvFrom(rq := true, chanCode := CH1_uni, lenRx := 380,
           data := void(Buffer));

  iWriteToFileSeq(fileName := FileName, srcVar := void(Buffer),
                  write := Record AND RecvFrom.mesRec,
                  close := NOT Record,
                  size := UINT_TO_UDINT(RecvFrom.lenData));

END_PROGRAM
```

6 PŘÍKLADY

6.1 Použití funkčních bloků ReadFromFile a WriteToFile

Následující příklad ukazuje použití funkčních bloků ReadFromFile a WriteToFile. Program v příkladu otestuje po restartu systému zda-li existuje adresář *DATA*. Pokud neexistuje tak ho založí. Poté v tomto adresáři založí soubor *test.txt* a zapíše do něho data uložená v proměnné *record*. Poté se obsah souboru *test.txt* načte do proměnné *copy*, která se porovná s proměnnou *record*. Obsah obou proměnných musí být shodný. Jedná se tedy o jednoduchý test zápisu dat do souboru a jejich načtení pro kontrolu. Výsledek testu udávají proměnné *test_OK* a *test_ERR*. Důvod případných potíží je pak vidět v proměnné *problem*.

```
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
```