Monitorovací a vizualizační systém ProCop 3.8

Jazyk Bára

Programátorská příručka

Ostrava 2022

Jazyk Bára

Programátorská příručka

ALFA Mikrosystémy, s.r.o.

Monitorovací a vizualizační systém ProCop 3.8 je specializovaný software pro monitorování technologických procesů, uživatelské řízení technologií, archivaci historických trendů technologických veličin a alarmních stavů.

Zahrnuje zkušenosti z předchozích verzí systému, které mají počátky již v roce 1993, kdy byl poprvé nasazen první předchůdce stávajícího monitorovacího systému. Dlouholetou praxí v oboru monitorování technologií, zejména pak v tepelném hospodářství, se podařilo dle zkušeností a požadavků zákazníků a samotných dispečerů vyvinout produkt, který se Vám v tuto chvíli dostává do rukou.

Vytištěno: červen 2022

Obsah

1 Základní rysy jazyka

Jazyk Bára je výpočetní jazyk určený pro numerické výpočty v systému ProCop. Používá se pro vyhodnocování numerických výrazů a podmínek dynamizací a zároveň pro tvorbu složitějších výpočetních algoritmů prostřednictvím tzv. **Bára Scriptu**.

V podadresáři "Scripts" adresáře ProCop 3.8 naleznete několik příkladů a hlavičkových souborů. ve stejnojmenném podadresáři monitorovacího projektu se pak nacházejí projektové skripty.

1.1 Speciální symboly a rezervovaná slova

Definice základních znaků:

- **Písmena** - znaky 'A' - 'Z', 'a' - 'z', velké i malé znaky národních abeced (ěščř…)
- **Podtržítko** - znak '_'
- **Dekadické číslice** (někdy jen číslice) - arabské číslice 0..9
- **Hexadecimální číslice** - arabské číslice 0..9, písmena 'A'-'F' nebo 'a' - 'f'
- **Mezerové znaky** (bílé znaky) - mezera, tabelátor, konec řádku

Rezervovaná slova

V jazyce Bára mohou být použita tato klíčová slova:		
Analog	**For**	**Record**
And	**Function**	**Ref**
Array	**Global**	**Return**
Begin	**Goto**	**Step**
Binary	**If**	**Text**
Counter	**Label**	**Then**
Discrete	**Local**	**To**
Do	**Of**	**True**
Else	**Or**	**Type**
End	**Procedure**	**While**
False	**Program**	

1.2 Poznámky ve zdrojovém textu

Poznámky ve zdrojovém textu mohou být dvojího druhu:

- **Poznámky vnořené kdekoli** ve zdrojovém textu a dlouhé libovolný počet řádků - uzavírají se do složených závorek, musí vždy začínat znakem " { " a končit znakem " } ". Všechen zdrojový text uzavřený závorkami je ignorován a celá poznámka je považována za mezeru
- **Poznámky na konci řádku** - začínají vždy dvěma lomítky " // " a mají platnost pouze do konce řádku. Všechen text na řádku za těmito znaky je zcela ignorován. Tento typ poznámky nemá ukončovací znak - poznámka je automaticky ukončena koncem řádku

```
// začátek procedury
procedure Kon{ chybná poznámka }trolaMezi( hodnota, horni, dolni: analog;
porucha : binary );
local
  chyba : binary;     {poznámka O.K.}
begin
  chyba := horni < {tato poznámka je O.K.} hodnota OR hodnota < dolni;
  AlarmniPromenna := chyba AND NOT( porucha );      // spusti alarm
end;
```

```
// konec procedury
```

1.3 Identifikátory

Identifikátory se mohou skládat z písmen, číslic nebo podtržítka - znaků '_'. Na prvém místě identifikátoru nesmí být nikdy číslice. Identifikátory se nerozlišují podle velikosti písmen - identifikátory **'Ident'**, **'ident'** a **'IDENT'** jsou **shodné**.

```
_Velikost6
Proměnná
&Pomocná              // toto je příklad chybného identifikátoru
2Mezivýsledek         // a toto taky
```

1.4 Datové typy

Programovací výpočetní jazyk Bára rozeznává pět jednoduchých typu proměnných. Tyto typy jsou v úzké návaznosti na typy používané v databázích monitorovacího systému. Jednotlivé typy jsou mezi sebou vzájemně nekompatibilní a typovou konverzi je nutno uskutečnit pomocí volání konverzních funkcí 24.

Deklarace typu proměnných

Jednotlivé typy se deklarují pomocí klíčových slov shodných se jmény typu. Jsou to: Analog, Binary, Counter, Discrete a Text. Tato klíčová slova je nutno použít při deklaraci proměnných nebo v hlavičkách procedur 15 a funkcí 17.

Datový typ	*Bitů*	*Popis*	*Rozsah*
ANALOG	32	reálné (desetinné)	3.4E-38 ~ 3.4E38
BINARY	1	logické	TRUE, FALSE
COUNTER	32	celočíselné	-2 147 483 648 ~ 2 147 483 647
DISCRETE	8	celočíselné	0 ~ 255
TEXT	-	textový řetězec	max. délka 255 znaků

Analog

Tento typ je čtyřbajtové číslo s plovoucí desetinnou čárkou (v jiných jazycích například typ "float" nebo "real"). Přesnost je 6 až 7 platných číslic. Protože se jedná o číslo s plovoucí desetinnou čárkou, je vhodné si uvědomit, že doba jeho zpracování je delší než doba zpracování proměnných ostatních typů. Rychlost zpracování je také silně závislá na tom, zda je k dispozici matematický koprocesor.

Konstanty 10 tohoto typu je možno zapsat jako celé dekadické číslo nebo číslo v exponenciálním tvaru.

Binary

Typ reprezentuje pouze logické hodnoty 1 nebo 0. Pro zápis konstant tohoto typu je možno použít hodnotu **0** a **1**, případně vyhrazená klíčová slova "**True**" - nabývá hodnoty log. 1 a "**False**" - má hodnotu log. 0.

Counter

Jedná se o **celočíselný** typ délky 32 bitů se znaménkem s rozsahem od -2,147,483,648 do 2,147,483,647. Konstanty 10 tohoto typu je nutno zapsat jako celé dekadické číslo nebo hexadecimální číslo ve zmíněném rozsahu.

Discrete

Typ "**Discrete**" je obdobou typu "Counter", avšak jeho délka je 8 bitů a je bezznaménkový. Znamená to, že jeho rozsah je pouze 0 až 255.

Text

Typ "**Text**" je statický ukazatel na text. V současné verzi není možno deklarovat proměnné typu text delší než 255 znaků.

1.5 Výrazy

Výrazy se skládají z operandů 9 a operátorů 10. Vyhodnocuje se postupně podle úrovní jednotlivých operátorů a jeho výsledkem je číselná hodnota určitého typu. Části výrazu s předností ve vyhodnocování je možno uzavřít do kulatých závorek 10.

Syntakticky lze výraz znázornit takto:

```
operand < operátor operand … >
```

Není možno použít dva operátory za sebou, ale je nutno použít závorek:

```
3*-1          // chybný výraz
3*(-1)        // správný zápis
```

Příklady výrazů – priorita operátorů:

```
global
  X : counter = 1;
  Y : counter = 2;
  Z : counter = 3;
  A : analog;
  B : binary;
  C : counter;
begin
  A := 3.14;
              // výsledek 3.14
  C :=  Analog2Counter( A ) + X + Y * Z;
              // vyhodnocuje se nejprve Y * Z a k výsledku
        // se přičte X a A, výsledek 10
  B := 2 < X + ( Y AND  Z * 2 );
              // nejprve Z * 2, poté bitové AND s Y,
        // přičte se X a porovná s 2, výsledek TRUE
end.
```

1.6 Operátory

Implementovány jsou operátory **binární** - mají tedy vždy dva operandy: jeden před operátorem a jeden za operátorem. Oba operandy musí být téhož typu. Výsledný typ operátoru spolu s dovolenými typy operandů jsou uspořádány do tabulky.

Operátor	*Úroveň*	*Typy operandů*	*Typy výsledků*	
*, /	1	A, C, D	stejné jako operand	součin, podíl operandů
%	1	D, C	stejné jako operand	zbytek po celočíselném dělení operandů
+, -	2	A, C, D	stejné jako operand	součet, rozdíl operandů
AND, OR	3	B, C	stejné jako operand	logický (bitový) součin a součet operandů
>, <, =, <>, <=, >=	4	A, B, C, D	BINARY	logické porovnání operandů

Co se týče úrovní operátoru, platí tato základní pravidla:

- operand mezi dvěma operátory různých úrovní je vyhodnocen nejdříve operátorem s vyšší úrovní - např. výraz 5+3*2 se vyhodnotí jako 5+(3*2)
- operand mezi operátory stejných úrovní je předán k vyhodnocení levému operátoru - výraz 3+2+1 je totéž jako (3+2)+1
- část výrazu uzavřená do závorek je vyhodnocena jako oddělený výraz s nejvyšší úrovní

Poznámka
*Unární logický operátor NOT není implementován, je však k dispozici funkce **Not(par)** s jedním parametrem.*

Poznámka
*Unární bitový operátor INV není implementován, je však k dispozici funkce Inv**(par)** s jedním parametrem.*

1.7 Operandy

Za operandy výrazu je možno považovat:

- konstanty 10 - jedná se o jednoduché číselné konstanty
- proměnné 15 - zde je možno použít identifikátory již deklarovaných proměnných
- volání funkce 23 - zastoupen identifikátorem funkce s případnými parametry
- výraz uzavřený v závorce

```
konstanta | proměnná | volání funkce | ( výraz )
```

Pro přístup k položkám proměnných typu záznam 13 se používají znaménka '.' a pro typ pole 13 index uzavřený mezi hranaté závorky ' [ ] '.

1.8 Konstanty

Celá čísla dekadická

Skládají se ze sekvence dekadických číslic.

```
123456  0       78      -564
```

Čísla hexadecimální

Číslo se skládá ze sekvence hexadecimálních číslic jimž předchází znaky '0x'.

```
0xFFFF1111     0x01234        0xABCDEF
0xFG              // chybný zápis hexadecimálního čísla
```

Čísla v exponenciálním tvaru

Číslo v exponenciálním tvaru lze obecně zapsat jako:

```
[ +|- ] XX[.XX][E[ +|- ]XX]
```

kde:

- části uzavřené v hranatých závorkách jsou nepovinné a mohou být vypuštěny
- pro znaménka oddělená znakem '|' platí, že může být na daném místě vždy jen jedno z nich, znaky 'X' značí libovolné dekadické číslice

```
-1.235E+24     0.01    25986   +2E-02
```

Znaky

Znaková konstanta je znak uzavřený do apostrofu. Pro speciální znaky je možno použít namísto znaku zpětné lomítko s hexadecimálním vyjádřením znaku v ASCII tabulce.

```
'A'     '%'     '\x00' '\xE6'
```

Textové konstanty

Textová konstanta je libovolný text uzavřený do uvozovek. V současné verzi systému je možno používat textové konstanty maximální délky 255 znaků.

```
"Toto je textová konstanta"
```

1.9 Závorky

Části výrazu s předností ve vyhodnocování je možno uzavřít do kulatých závorek.

```
(9+3)*3
```

Obdobně je nutno uzavírat do závorek záporná čísla v případě, kdy po sobě následují dva operátory (znaménka).

```
2 * (-6)
```

2 Direktivy preprocesoru

2.1 Vložené soubory

Jazyk Bára umožňuje vkládání dalších souborů pomocí **direktiv preprocesoru**:

```
#include <jmeno_souboru>
```

případně

```
#include "jmeno_souboru"
```

Direktiva "**#include**" provede vložení zdrojového kódu z požadovaného souboru na dané místo.

Je-li jméno souboru uzavřeno ve špičatých závorkách <…>, předpokládá se, že se je jedná o soubor v adresáři se systémovými Bára Skripty monitorovacího systému ProCop.

Je-li jméno souboru uzavřeno ve uvozovkách "…", předpokládá se, že se jedná o soubor v aktuálním adresáři (případně v adresáři s projektem).

Pokus o dvojí vložení téhož souboru je automaticky rozpoznán a soubor je vložen pouze jednou.

2.2 Makra

Jazyk Bára umožňuje definovat jednoduché makra pomocí příkazu:

```
#define identifikátor_makra telo_makra
```

Pokud je v dalším zdrojovém textu nalezen identifikátor makra, pak se makro rozvine vložením těla makra na místo identifikátoru. Tímto způsobem jsou například definovány konstanty pro nejrůznější souborové operace, operace s alarmy, displeji apod.

3 Struktura programu

Každý program musí dodržovat tuto strukturu:

```
Program <název_programu> ;
  deklarace návěští
  deklarace složených typů
  deklarace globálních proměnných
  deklarace procedur
  deklarace funkcí
  tělo programu
.
```

Jednotlivé deklarace je možno mezi sebou libovolně opakovat, vyjama těla programu. Žádná z deklarací není povinná, a je možno ji vynechat, vyjma těla a tečky na konci programu.

V dalších kapitolách budou postupně popsány všechny části programu.

3.1 Deklarace návěští

Náveští jsou místa programu, na něž je možno v případě potřeby 'skočit' pomocí příkazu skoku. Skok - tedy změna posloupnosti vykonávání příkazu - je možný pouze v rámci jednoho podprogramu.

Každé návěští musí být nejprve deklarováno (v místě hlavičky podprogramu, popřípadě mezi globálními deklaracemi hlavního programu) a poté definováno v místě, kam je třeba skočit. Deklarace začíná klíčovým slovem "Label", poté následují jména návěští oddělená čárkou a seznam je ukončen středníkem. Seznam jmen návěští se může skládat z několika částí oddělených středníkem.

```
label < identifikátor_návěští <,identifikátor_návěští…>  ; …>
```

Příklad deklarace návěští:

```
label
  lab1,lab2;
  lab3;
```

Definice návěští se provádí v libovolném místě složeného příkazu. Stačí napsat pouze identifikátor návěští a za ním dvojtečku.

Syntaktický diagram vypadá takto:

```
begin
  lab1:              // identifikátor_návěští
  if( cnt < 100 ) then
    begin
      cnt := cnt + 1;
      glob := SQR( cnt );
      goto lab1;
    end;
end.
```

3.2 Deklarace složených typů

Mezi složené typy patří typy pole a záznam. Deklarace typu je uvozena klíčovým slovem "Type". Jednotlivé deklarace typů jsou ukončeny středníkem.

```
type identifikátor:
deklarace_typu;
```

Deklarace typu pole

Je možno deklarovat jednorozměrná pole jednoduchých typů i pole typů složených. Délka pole může být maximálně 255 položek. Prvky pole se indexují od 0 do (počet prvků-1). Při překročení indexu pole je generována chyba při běhu programu. Je-li potřeba deklarovat vícerozměrné pole, je třeba deklarovat pole typu pole. Maximálně je možno vytvořit třírozměrná pole.

```
array [celočíselná_konstanta] Of deklarace_typu | identifikátor_typu;
```

Deklarace typu pole začíná klíčovým slovem "**Array**". Následuje rozměr pole vyjádřený kladnou celočíselnou konstantou uzavřený do hranatých závorek. Typ prvku pole je uveden za klíčovým slovem "**Of**" a může být vyjádřen identifikátorem již existujícího typu nebo deklarací nového typu.

```
type
  TMyLinearArray : array[ 32 ] of analog;
  TMy3DArray : array[ 32 ] of array[ 10 ] of array[ 5 ] of analog;
  TMyRecordArray : array[ 32 ] of record
    a : analog;
    b : binary;
  end;
```

Deklarace typu záznam

Záznam (struktura) je složena z položek různých typů. Každá položka záznamu má svůj identifikátor unikátní v rámci tohoto záznamu. Položkou záznamu může být jednoduchý nebo složený typ.

```
record
   identifikátor položky : deklarace_typu | identifikátor_typu ;
   <
   identifikátor položky : deklarace_typu | identifikátor_typu ;
   …>
end;
```

Deklarace typu záznam je uvozena klíčovým slovem "**Record**". Následuje seznam položek oddělený středníky. Konec záznamu je označen klíčovým slovem "**End**". Jednotlivé položky se skládají z identifikátoru položky následované identifikátorem typu položky nebo deklarací typu položky.

```
type
  TMyRecord : record
     Value  : analog;
     Name   : text;
     Status : discrete;
     Vector   : TMyLinearArray;
  end;
```

Prvky složených typů nesmí být externí typy (kanály monitorovacího modulu).

Přístup k položkám složených typů

Pro přístup k položkám typu pole se používá index uložený v hranatých závorkách. Indexem musí být výraz typu "Counter". Obdobně pro přístup k položkám typu záznam se používá znaménko '.' oddělující identifikátor proměnné od identifikátoru položky záznamu. V případě vzájemně vnořených typů se používají kombinace indexace a přístupu pomocí znaménka '.'.

Pro ilustraci jsou použity typy deklarované v předchozích dvou příkladech.

```
global
  arr : TMyLinearArray;
  arr3d : TMy3DArray;
  rec : TMyRecord;
begin
  arr[ 4 ] := arr3d[ 2*1 + 3  ][ 3 ][ 0 ];
  rec.Value := rec.Vector[ 5 ];
...
```

3.3 Deklarace proměnných

Proměnné se rozlišují podle místa deklarace, způsobu uchování hodnoty a použití do tří skupin: proměnných globálních, lokálních a externích.

Deklarace globálních proměnných

Globální proměnné jsou společné pro všechny procedury a funkce programu. Jsou 'viditelné' a použitelné z kteréhokoli místa programu za vlastní deklarací proměnné. Deklarace začíná vždy klíčovým slovem Global. Poté následují řádky deklarací proměnných jednoho typu oddělené středníkem.

Každý řádek se skládá z unikátního jména nové proměnné (je-li proměnných několik, pak oddělených čárkami), poté následuje dvojtečka a identifikátor typu. Je-li požadováno, aby proměnná při prvním cyklu běhu programu měla definovanou hodnotu (tedy aby byla inicializována), je třeba doplnit deklaraci znakem '=' spolu s konstantou příslušného typu.

```
global < identifikátor < ,identifikátor …> :
                         identifikátor_typu < = konstanta > ; …>
```

```
global
  a1, a2, a3 : analog = 3.6E-1;
  c : counter;
  d, pom : discrete = 0xF;
  _bin : binary = FALSE;
```

Deklarace lokálních proměnných

Deklarace lokálních proměnných je syntakticky obdobná deklaraci proměnných globálních. Deklaraci uvozuje klíčové slova "**Local**" namísto "**Global**".

Další rozdíly jsou pouze ve vlastnostech a místě deklarace lokálních proměnných. Zatímco globální proměnné jsou přístupné k použití ze všech míst programu (za místem deklarace), lokální proměnné jsou vytvářeny jen dočasně a jsou použitelné pouze zevnitř procedury nebo funkce, v níž jsou deklarovány.

Zde je už vidět druhý rozdíl, a to v místě deklarace. Lokální proměnné se definují vždy mezi hlavičkou a tělem procedury (funkce). Přesná struktura deklarace procedur je popsána ve zvláštní kapitole.

Třetí rozdíl je v inicializaci lokálních proměnných. Lokální proměnná (je-li to požadováno) je inicializována vždy při vstupu do procedury. Je třeba mít na zřeteli, že inicializaci provádějí při každém vstupu do procedury interní funkce (shodné jako při zápisu proměnná = konstanta) a tedy zpomalují běh programu.

Externí proměnné

Externí proměnné zasluhují zvláštní pozornost . Jsou jediným místem, kde se program napojuje na proces. Externí proměnné jsou ve skutečnosti proměnné uložené v databázích V/V modulu a jejich modifikací je možno upravovat potřebné hodnoty a parametry procesu. Vzhledem k tomu, že procesní hodnoty jsou distribuovány do podřízených řídících vrstev, je nutno mít na paměti dopravní zpoždění v systému. Tím se stane, že po zápisu hodnoty trvá určitý čas, než dojde k fyzickému nastavení hodnoty. Tento čas může být až řádu jednotek minut (pro telefonní modemy).

V současné verzi není třeba externí proměnné deklarovat, je možno používat skutečné názvy řídicích proměnných. Externí proměnné jsou deklarované jako typ pole rozšířený o strukturu příznaku.

3.4 Deklarace procedur

Procedura musí být před prvním použitím (voláním) nejprve deklarována. Deklarace procedury se skládá z hlavičky se seznamem parametrů procedury, z nepovinné deklarace lokálních proměnných, návěští a vlastního těla procedury.

```
hlavička procedury
  <
    <deklarace lokálních proměnných>
    <deklarace návěští>
  …>
  tělo procedury ;
```

Hlavička procedury

Hlavička procedury začíná klíčovým slovem "**Procedure**". Poté následuje identifikátor procedury. V případě, že je třeba

deklarovat parametry procedury, je tato deklarace uzavřena do kulatých závorek. Deklarace hlavičky je ukončena středníkem.

```
procedure identifikátor_procedury < (deklarace parametrů) > ;
```

Deklarace parametrů procedury

Má-li procedura nějaké parametry, je nutno je deklarovat v hlavičce procedury. Deklarace parametrů je uzavřena kulatými závorkami.

Parametry mohou být libovolných typů, mohou být seřazeny v libovolném pořadí a každý z nich má svůj identifikátor. Deklarace parametrů jednoho typu začíná vždy seznamem identifikátorů parametrů oddělených čárkami. Následuje dvojtečka a identifikátor typu parametru. Je-li třeba deklarovat parametry dalšího typu, následuje středník a poté se deklarace opět opakuje. Je-li potřeba deklarovat parametry předávané odkazem, použijeme před seznam jmen parametrů klíčové slovo "**Ref**".

```
< < Ref > identifikátor_parametru < ,identifikátor_parametru >
        : identifikátor_typu  … >
```

Parametry procedur i funkcí jsou volány hodnotou nebo odkazem. Volání hodnotou znamená, že při volání procedur s parametry se nejprve vyhodnotí výraz, který je na místě parametru, a jeho výsledná hodnota se předá proceduře. Procedura může libovolně pracovat s parametry bez nebezpečí modifikace originálních hodnot. Při předávání parametrů odkazem se namísto hodnoty proměnné předává odkaz (adresa) na tuto proměnnou a tato proměnná může být při práci s parametrem v proceduře modifikována.

Poznámka: *Do procedury je možno předávat složené typy pouze odkazem, nikoliv hodnotou.*

Příklad

```
procedure Mocnina( ref vysl : analog; a : analog );
begin
  vysl := a * a;
end;
```

Deklarace lokálních proměnných

viz. Deklarace proměnných 15

Deklarace návěští

viz. Deklarace návěští 13

Tělo procedury

Tělo procedury je vlastně složený příkaz (tedy začíná klíčovým slovem "**Begin**" a končí klíčovým slovem "**End**"). Poté následuje středník.

Jediným rozšířením je použití příkazu návratu 20 "**Return**". Vykonáním příkazu návratu je běh procedury ihned bezpodmínečně ukončen a řízení se předá volajícímu podprogramu.

```
// Proměnné AlarmniPromenna, Teplota1, Teplota2, Stav jsou proměnné z
// databáze proměnných monitorovacího projektu.
procedure KontrolaMezi( hodnota, horni, dolni:analog; porucha:binary );
local
  chyba : binary;
begin
  chyba := ( horni < hodnota ) OR ( hodnota < dolni );
  AlarmniPromenna := chyba AND NOT( porucha );          // spustí alarm
  return;                                               // návrat z procedury
  chyba := TRUE;                                        //   toto   se   nikdy
neprovede
end;

procedure Kontroly;
begin
  KontrolaMezi( Teplota1, 50, 20, Stav );
  KontrolaMezi( Teplota2, 80, -10, Stav );
```

```
end;

// Zde začíná hlavní tělo programu
begin
  Kontroly();                                          // tělo programu
end.
```

3.5 Deklarace funkcí

Deklarace funkcí je obdobou deklarace procedur s několika odlišnostmi. Změněná je syntaxe hlavičky funkce oproti proceduře; každá funkce musí být ukončena klíčovým slovem "**Return**" následovaným návratovou hodnotou (výrazem).

V této kapitole budou popsány pouze odlišnosti oproti deklaraci procedur.

Hlavička funkce

Hlavička funkce začíná klíčovým slovem "**Function**". Poté následuje identifikátor funkce. Je-li třeba deklarovat parametry funkce, je tato deklarace uzavřena do kulatých závorek. Následuje klíčové slovo " **Of** " uvozující identifikátor návratového typu procedury. Deklarace hlavičky je ukončena středníkem.

```
function identifikátor_funkce < (deklarace parametrů) >
                                    of  identifikátor_typu ;
```

Návratová hodnota funkce

Každá funkce musí být ukončena příkazem návratu 20 začínající klíčovým slovem "Return" a následovaným výrazem 9 stejného typu jako je návratová hodnota funkce (viz. hlavička funkce - identifikátor typu za "Of"). Výraz se nejprve vyhodnotí a výsledek je předán do místa volání funkce 19 jako hodnota funkce (v příkladech např. hodnota druhé mocniny čísel nebo obsah kruhu).

Je-li ve funkci několik příkazů návratu, je vyhodnocen ten, ke kterému interpret jazyka dojde dříve.

Například:

```
function MySQR( a : analog ) of analog;
begin
  return a * a;
end;

function Obsah( r : analog ) of analog;
begin
  if r < 0 then return 0;
  return Pi() * MySQR( r );
end;
```

3.6 Tělo programu

Tělo programu je vlastně složený příkaz 21 ukončený tečkou. Části zdrojového textu za tečkou se nekompilují.

4 Jednoduché příkazy

4.1 Příkaz skoku

Pomocí příkazu skoku je možno změnit posloupnost vykonávání programu. Příkaz skoku se skládá z klíčového slova "**Goto**" za nímž následuje identifikátor návěští, kam má být přeneseno řízení.

```
goto identifikátor_návěští
```

Vykonáním příkazu skoku program provede příkaz ležící bezprostředně za místem definice návěští 13.

```
label l;
global
  a : counter;
begin
  a := 1;
l:
  a := a + 1;
  if a < 10 then
    goto l;
end.
```

4.2 Přiřazovací příkaz

Přiřazovací příkaz slouží k přiřazení hodnoty výrazu 9 proměnné. Je možno přiřazovat hodnotu libovolnému typu proměnné, avšak typ proměnné 8 musí být shodný s výsledným typem použitého výrazu.

```
identifikátor_proměnné := výraz;
```

Syntakticky je přiřazovací příkaz velmi jednoduchý. Skládá se z identifikátoru proměnné, jíž má být hodnota přiřazena, znaku přiřazení ' := ' a výrazu.

```
a := 1;
a := a + 1;
a := 300;
b := 2 * ( 1 + SQRT( a + 2 ) );
```

4.3 Příkaz volání procedur a funkcí

Příkaz volání procedur a funkcí provádí příkazy dané procedury 15 nebo funkce 17. Volaná procedura nebo funkce musí být deklarována před příkazem volání. Jedná-li se o volání funkce, lze její návratovou hodnotu použít jako operand v přiřazovacím příkazu 19, v podmínce atp.

Příkaz začíná identifikátorem volané funkce nebo procedury následovaný parametry oddělenými čárkami a uzavřenými v kulatých závorkách. Kulaté závorky jsou povinné i v případě, že se volané funkci nebo proceduře žádné parametry nepředávají.

```
identifikátor_procedury ( < parametr, …> )
```

nebo

```
identifikátor_funkce ( < parametr, …> )
```

Typ použitých parametrů musí odpovídat parametrům v deklaraci funkce. Pro případnou typovou konverzi je možno použít interní konverzní funkce 24 jazyka Bára.

```
// deklarace funkce Minimum vracející minimální hodnotu z a,b,c
function Minimum ( a, b, c : analog ) of analog
  begin
```

```
  if ( Min (a,b) = a ) then
      return Min (a,c)
    else
      return Min (b,c);
  end;
// Použití volání funkce v přiřazovacím příkazu
....
minHodnota := Minimum (x, y, z);
....
// Použití volání funkce v podmínce
....
if ( Minimum (hodnota_1,hodnota_2, Counter2Analog( hodnota_3 ) ) > 0 )
   then .....
```

4.4 Příkaz návratu

Příkaz návratu slouží k okamžitému ukončení procedury nebo funkce a návratu do volajícího podprogramu. Příkaz návratu začíná klíčovým slovem "Return". V případě, že se příkaz nachází v proceduře, následuje dále pouze středník. V případě, že se jedná o funkci, musí za klíčovým slovem "Return" následovat výraz 9, jehož typ je shodný s typem funkce. Výraz se před odchodem z funkce vyhodnotí a výsledek se předá jako výsledná hodnota funkce volajícímu podprogramu.

```
return < výraz {jen pro funkce} >
```

Každá funkce musí obsahovat alespoň jeden příkaz návratu, který zabezpečí předání výsledku funkce volajícímu podprogramu. V opačném případě není návratová hodnota funkce definována.

```
function SQR( a : analog ) : analog;
begin
  return a * a;    // navratová hodnota funkce
end;

procedure Compute( param : analog );
label l;
local
  r : analog = 2;
begin
l:
  r := SQR( r );
  if( r > param )
    return;     // návrat z procedury
  goto l;
end;
```

5 Příkazy strukturované

5.1 Příkaz složený

Příkaz složený se skládá ze sekvence jednoduchých nebo složených příkazů oddělených od sebe středníkem. Celý příkaz je uvozen klíčovým slovem "**Begin**" a ukončen klíčovým slovem "**End**". Všechny příkazy uvedené v těle složeného příkazu se provádějí sekvenčně v pořadí, v jakém jsou zapsány. Jedinou výjimku tvoří příkaz skoku 19 a příkaz návratu 20, které mohou například přeskočit několik jiných příkazů.

```
begin
  <
     příkaz skoku ;
     příkaz návratu ;
     přiřazovací příkaz ;
     podmíněný příkaz ;
     volání procedury ;
     složený příkaz
  …>
end;
```

5.2 Podmíněný příkaz

Podmíněný příkaz úplný a neúplný slouží k testování hodnot proměnných a k podmíněnému vykonávání potřebných akcí v závislosti na výsledku testu. Podmíněný příkaz úplný je složen z následující konstrukce:

```
if podmínka then příkaz_1 < else příkaz_2 >
```

kde podmínka musí být výraz 9 typu Binary, "příkaz_1" je libovolný příkaz, který se provede v případě výsledku podmínky "**True**" a "příkaz_2" se provede v případě výsledku podmínky "**False**". Neúplný podmíněný příkaz je ukončen za "příkazem_1" - postrádá tedy větev vykonávanou v případě, že výsledek podmínky je "**False**".

```
function Odmocnina( hodnota : analog; chyba : binary) of analog;
begin
  chyba := false;
  if ( hodnota < 0 ) then
    begin
      chyba := true;
      return 0;
    end
  else
    return Sqrt( hodnota );
end;
```

5.3 Smyčka typu FOR - TO - STEP

Smyčka typu "**For**" provede přiřazení počáteční hodnoty do řídící proměnné a provede porovnání řídící proměnné s výrazem uvedeným za "**To**". Vyhovuje-li řídící proměnná omezující podmínce:

```
proměnná <= výraz za TO    // pro krok > 0
proměnná >= výraz za TO    // pro krok < 0
```

pak se provede tělo cyklu. Při dalších průchodech se přičte k řídící proměnné krok (implicitně 1) a znovu se vyhodnocuje omezující podmínka. Krok je možno explicitně nastavit použitím klíčového slova "**Step**" s následující číselnou konstantou 10 (krok je možno nastavit i záporně).

```
for přiřazovací_příkaz to výraz < step konstanta > příkaz
```

Například :

```
for j := 0 to SampleCount
  begin
       // tělo cyklu
  end;
```

nebo

```
for i := 2 * Pi() to 0 step (-0.01)
  begin
       // tělo cyklu
  end;
```

5.4 Smyčka typu WHILE - DO

Smyčka typu "**While**" - "**Do**" (smyčka s podmínkou na počátku) opakovaně provádí tělo cyklu, dokud platí omezující binární podmínka na počátku smyčky. Není-li podmínka nikdy splněna, tělo cyklu se vůbec neprovede.

```
while podmínka do příkaz
```

Například:

```
function GetLastValidSample ( ref t : THTrendChannelAnalog) of analog;
local
  c, tm : counter;
begin
  tm := t.LastSampleTime;            // čas posledního vzorku
  c  := t. SampleCount;              // celkový počet vzorků
  while ( Not( t.Valid[ tm ] ) ) do  //dokud není vzorek validní
    begin
      tm := tm - t.Period;
      // ošetření případu, kdy v trendu není žádný validní vzorek
      c  := c - 1;
      if ( c < 0 ) then
        return 0;
    end;

  return  t [ tm ];
end;
```

5.5 Smyčka typu DO - WHILE

Smyčka typu "**Do**" - "**While**" (smyčka s podmínkou na konci) provede tělo cyklu a otestuje omezující binární podmínku na konci smyčky. Je-li podmínka splněna, je tělo smyčky provedeno znovu atd. Není-li podmínka nikdy splněna, tělo cyklu se provede právě jednou.

Aby bylo možno určit, není-li ukončovací klíčové slovo "**While**" vnořenou smyčkou typu "**While** - **Do**", je nutno tělo smyčky psát jako složený příkaz 21.

Ve všech typech smyček může být tělem smyčky jednoduchý příkaz nebo složený příkaz (uvozený "**Begin**" a "**End**").

```
do příkaz while podmínka
```

Například:

```
i := 10;
do
  begin
       // tělo cyklu
       i := i - 1;
  end;
while( i >= 0 );
```

6 Funkce

Funkce je možno rozdělit do třech skupin:

- matematické 23 - určené pro matematické výpočty
- konverzní 24 - pro konverzi mezi jednotlivými datovými typy
- pro animační dynamizace 24 – funkce pro jednoduché vytváření animačních dynamizací
- pro řízení běhu programu 26 – funkce pro řízení běhu programu v Bára Skriptu
- souborové funkce 26 – nejrůznější souborové funkce, otevření, čtení, zápis, zavření souboru atd.
- pro práci s technologickými displeji 28 – funkce pro přepínání, tisk, zavírání technologických displejů
- pro práci s alarmy 27 – funkce pro zápis do alarmu, událostí a systémového zápisníku
- pro práci s přístupovými oprávněními 28 – funkce pro zjišťování aktuálního uživatele, přihlášení, odhlášení
- pro práci se zvukem 29 – funkce pro přehrávání zvuku
- časové 30 - určené pro práci s datumem a časem
- ostatní 31 - výše nezařaditelné funkce

6.1 Matematické funkce

Implementované základní matematické a goniometrické funkce pracující pouze s datovým typem "**Analog**".

Funkce	*Matematicky*	*Popis*
Pi	3.1415	Ludolfovo číslo (bez parametru)
Sqr	x^2	druhá mocnina
Sqrt	-	druhá odmocnina
Exp	e^x	x-tá mocnina e
Pow	y^x	x-tá mocnina y (dva parametry)
Pow10	10^x	x-tá mocnina 10
Log	log(x)	dekadický logaritmus
Ln	ln(x)	přirozený logaritmus
Rad2Deg	-	převod radiánů na stupně
Deg2Rad	-	převod stupnů na radiány
Sin	sin(x)	sinus x (parametr v rad)
Cos	cos(x)	cosinus x (parametr v rad)
Tan	tg(x)	tangens x (parametr v rad)
Asin	Arcsin(x)	cyklometrické funkce (výsledek v rad)
Acos	Arccos(x)	cyklometrické funkce (výsledek v rad)
Atan	Arctg(x)	cyklometrické funkce (výsledek v rad)
Abs	|x|	absolutní hodnota x
Sign	Sign(x)	signum (1 pro kladné, -1 pro záporné)
Round	-	zaokrouhlení
RoundDown	-	zaokrouhlení vždy dolů
RoundUp	-	zaokrouhlení vždy nahoru
Min	Minimum(x, y)	minimum ze zadaných argumentů
Max	Maximum(x, y)	maximum ze zadaných argumentů
Select	Select(podm, x,y)	je-li výraz pod hodnoty TRUE, vrací x, jinak y
Random	Random(x)	vrací pseudonáhodné číslo v intervalu 0..(x-1)

6.2 Funkce pro animační dynamizace

Tato skupina funkcí je určena pro snadnější vytváření animačních dynamizací entit.

Funkce	*Popis*
RotateWithPeriod	Vrací hodnotu 0-360. Výsledek je typu Analog.
ScaleMoveWithPeriod	Vrací hodnotu 0-1. Výsledek je typu Analog.
BlinkWithPeriod	Vrací hodnotu true nebo false. Výsledek je tedy typu Binary.

Všechny funkce vyžadují jako parametr periodu v milisekundách.

6.3 Konverzní funkce

Konverzní funkce zajišťují datovou konverzi mezi jednotlivými typy proměnných nebo výrazů. Jsou k dispozici konverze všech datových typů mezi sebou. Název konverzní funkce je tvořen vždy názvem typu parametru, číslicí 2 a názvem typu, do něhož hodnotu konvertujeme. Funkce jsou uspořádány do tabulky:

Typ	*Analog*	*Binary*	*Counter*	*Discrete*	*Text*
Analog	-	Analog2Binary	Analog2Counter	Analog2Discrete	Analog2Text
Binary	Binary2Analog	-	Binary2Counter	Binary2Discrete	Binary2Text
Counter	Counter2Analog	Counter2Binary	-	Counter2Discrete	Counter2Text
Discrete	Discrete2Analog	Discrete2Binary	Discrete2Counter	-	Discrete2Text
Text	Text2Analog	Text2Binary	Text2Counter	Text2Discrete	-

Příklad:

```
Analog2Counter( 2 * Pi() ) - Time > 3
```

6.4 Práce s řetězci

Pro práci s řetězci jsou v systémovém hlavičkovém souboru **<string.bah>** definovány konstanty **EOS** (konec souboru – hodnota 0) a **STRING_MAX_LEN** (maximální možná délka řetězce – hodnota 255), a dále je pro přehlednější práci s řetězci definován typ char jako proměnná typu discrete.

Deklarace funkce	*Popis*
Stralloc( ) of text	Funkce alokuje paměťové místo pro řetězec dlouhý max. 255 znaků. Operace je nutná pro ostatní funkce modifikující obsah řetězce. Je nutno zapnout podporu pro řetězcové funkce !
Strfree(t: text)	Funkce uvolní paměť řetězce alokovaného funkcí stralloc.
Strcpy(dest, src: text) of text	Funkce kopíruje obsah řetězce src do řetězce dest. Řetězec dest musí být alokován pomocí funkce stralloc.
Strcat(dest, src: text) of text	Funkce připojí obsah řetězce dest k řetězci src. Řetězec dest musí být alokován pomocí funkce stralloc.
Strlen(src: text) of counter	Funkce vrací aktuální délku řetězce.
Strfindch(src: text; c: counter) of counter	Funkce vyhledá první výskyt znaku c v řetězci a vrátí jeho polohu. V případě nenalezení vrací hodnotu -1.
Strgetch(src: text; inx: counter) of char	Funkce vrací znak z řetězce uložený na zadané pozici.
Strputch(dest: text; inx: counter; c: char) of text	Funkce zapíše znak do řetězce na zadanou pozici. Řetězec dest musí být alokován pomocí funkce stralloc.
Strinsch(dest: text; inx: counter; c: char) of text	Funkce vloží znak do řetězce na zadanou pozici. Znaky za touto pozicí odsune. Řetězec dest musí být alokován pomocí funkce stralloc.
Strdelch(dest: text; inx: counter) of text	Funkce vyjme znak z řetězce ze zadané pozice. Znaky za touto pozicí odsune. Řetězec dest musí být alokován pomocí funkce stralloc.
Strrdvld(src: text) of binary	Funkce vrací TRUE, je-li řetězec v pořádku a je určen pro čtení.
Strwrvld(src: text) of binary	Funkce vrací TRUE, je-li řetězec v pořádku a je určen pro čtení nebo zápis.
IsAlpha(c: char) of binary	Funkce vrací TRUE, je-li znak písmenem (pracuje včetně písmen s diakritikou podle zadané kódové stránky Windows).
IsAlphaNumeric(c: char) of binary	Funkce vrací hodnotu výrazu (isalpha() or isnumeric()).
IsNumeric(c: char) of binary	Funkce vrací TRUE, je-li znak číslicí 0 až 9.
IsUpper(c:char) of binary	Funkce vrací TRUE, je-li znak velkým písmenem (včetně diakritiky podle nastavené kódové stránky Windows).
IsLower(c:char) of binary	Funkce vrací TRUE, je-li znak malým písmenem (včetně diakritiky podle nastavené kódové stránky Windows).
ToUpper(c:char) of char	Funkce převede znak na velké písmeno (včetně diakritiky podle nastavené kódové stránky Windows).
ToLower(c:char) of char	Funkce převede znak na malé písmeno (včetně diakritiky podle nastavené kódové stránky Windows).

Upozornění!
Ke každému volání funkce StrAlloc musí existovat právě jedno volání StrFree, jinak nedojde k dealokaci použité paměti a následně k jejímu spotřebování .

Pro práci s řetězci je dále určen systémový Bára Skript **<string.bal>**, ve kterém jsou definovány další čtyři funkce pro práci s řetězci.

Deklarace funkce	*Popis*
Strclr(s: text) of text	Funkce vymaže řetězec. Proměnná s obsahuje prázdný řetězec.
Straddch(dest: text; c: char) of text	Funkce přidá znak c na konec řetězce dest.
Strupper(s: text) of text	Funkce převede řetězec s na velká písmena.
Strlower(s: text) of text	Funkce převede řetězec s na malá písmena.

6.5 Řízení běhu programu

Deklarace funkce	*Popis*
Sleep(delay: counter);	Funkce pozastaví běh programu na danou dobu (v milisekundách). Po uplynutí této doby program pokračuje v místě volání funkce Sleep.
Suspend();	Pozastaví běh programu do dalšího spuštění dynamizací. Po novém spuštění program pokračuje v místě volání funkce Suspend.
Exit();	Funkce Exit ukončí bezpodmínečně běh programu. Při novém volání programu program začíná od začátku.

6.6 Souborové funkce

Pro práci se soubory je potřeba vložit systémový hlavičkový soubor **<files.bah>**, ve kterém jsou definovány jednak konstanty pro volání funkcí "**FileOpen**" a "**FileSeek**" a jednak se zavádí typ "**HFILE**" (identifikátor souborů) jako proměnná typu counter.

Deklarace funkce	*Popis*
IniReadString(file, section, entry: text; dest :text) of binary	Funkce přečte řetězec znaků ze souborů typu INI.
IniWriteString(file, section, entry: text; dest :text) of binary	Funkce zapíše řetězec znaků do souboru typu INI.
FileExist(file: text) of binary	Funkce vrátí TRUE v případě, jestliže zadaný soubor existuje.
FileDateTime(file: text) of counter	Funkce vrací datum a čas poslední modifikace souboru v sekundách od 1.1.1980.
FileRename(newfile, oldfile: text) of binary	Funkce přejmenuje soubor. Nový soubor musí být na stejném disku.
FileCopy(destfile, srcfile: text) of binary	Funkce zkopíruje soubor.
FileDelete(file: text) of binary	Funkce smaže soubor zadaného jména.
FileOpen(file: text; openMode: discrete) of HFILE	Funkce otevře soubor a vrátí jeho identifikátor. Identifikátor souboru je třeba používat pro následné souborové funkce. Při chybě vrací hodnotu –1.
FileClose(hfile: HFILE) of binary	Funkce zavře soubor.
FileEnd(hfile: HFILE) of binary	Funkce vrací TRUE, jestliže se ukazatel v souboru dostane na jeho konec.
FileSeek(hfile:HFILE; offset:counter; from:discrete) of counter	Funkce změní polohu ukazatele v souboru.
FileSize(hfile: HFILE) of counter	Funkce vrátí délku otevřeného souboru.
FileReadString(hfile: HFILE; dest: text) of binary	Funkce přečte řádek z textového souboru (ukončený znaky CR-LF).
FileReadBytes(hfile: HFILE; dest: text; cnt: counter) of binary	Funkce přečte zadaný počet bytů z datového souboru.
FileWriteString(hfile: HFILE; dest: text) of binary	Funkce zapíše řádek do textového souboru a ukončí jej znaky CR-LF.
FileWriteBytes(hfile: HFILE; dest: text; cnt: counter) of binary	Funkce zapíše zadaný počet bytů do datového souboru.
FileExecute(exepath, window, cmdline:text) of binary	Funkce aktivuje nebo spustí program v daném okně s danými parametry. Vrací TRUE, pokud se akce podaří

Otevření souboru

Pro úplnost jsou zde uvedeny i možné hodnoty parametru "**openMode**" pro funkci "**FileOpen**". Parametr "**OpenMode**" je orientován příznakově, takže lze použít kombinaci několika příznaků oddělených operátorem **OR**.

Typ přístupu	*Popis*
OPEN_READ	Otevře soubor pro čtení.
OPEN_WRITE	Otevře soubor pro zápis.
OPEN_RDWR	Otevře soubor pro čtení i zápis.

Typ sdílení	*Popis*
SHARE_NONE	Soubor nelze sdílet.
SHARE_READ	Soubor lze sdílet pro čtení.
SHARE_WRITE	Soubor lze sdílet pro zápis.

Další příznaky	*Popis*
CREATE_NEW	Vytvoří nový soubor. Vrátí chybu, jestliže soubor již existuje.
CREATE_OVERWRITE	Vytvoří nový soubor. Přepíše soubor, jestliže již existuje.
CREATE_EXISTING	Otevře soubor. Vrátí chybu, jestliže soubor neexistuje.
CREATE_ALWAYS	Otevře soubor vždy. Neexistoval-li, vytvoří nový.
CREATE_TRUNCATE	Otevře soubor a zkrátí jej. Vrátí chybu, jestliže soubor neexistuje.

Posun aktuálního ukazatele v souboru

Parametr "**from**" funkce "**FileSeek**" udává, vůči čemu se má posun v souboru provést.

Funkce	*Popis*
SEEK_BEGIN	Posun se provede relativně vůči začátku souboru
SEEK_CURRENT	Posun se provede relativně vůči aktuální pozici v souboru
SEEK_END	Posun se provede relativně vůči konci souboru

6.7 Práce s alarmy a událostmi

Pro práci s alarmy, událostmi a se systémovým zápisníkem je potřeba vložit systémový hlavičkový soubor **<alarms.bah>**. Obsahuje definice konstant typů, příznaků a priorit alarmu. Pro zápis alarmu, událostí slouží následující funkce:

```
SendAlarmEx( type: discrete, txt: text, style: text, source: text, level: discrete, id: text );
```

Parametry funkce

Typ (type)	*Popis*
ALR_TYPE_ALARM	Zápis bude zařazen mezi alarmy
ALR_TYPE_EVENT	Zápis bude zařazen mezi události
ALR_TYPE_LOG	Zápis bude zařazen mezi systémové události

Parametr **txt** je informační text sdělení, **style** je textové jméno stylu alarmů, s jehož vlastnostmi má být zařazen.

Úroveň (level)	*Popis*
ALR_LEVEL_OK	Zánik alarmu, přechod do normálu
ALR_LEVEL_ERROR	Vznik alarmu, přechod do chybového stavu, hlášení události

Parametr **id** má význam jen u stavových alarmů pro párování vzniku a zániku alarmů. Nezaniklé párované alarmy jsou tak zobrazovány jako trvající.

Zastaralé funkce (ponechány z důvodu zpětné kompatibility skriptů)

Následující funkce a konstanty jsou zachovány pro kompatibilitu s verzemi 2.x, dále je není vhodné používat:

```
SendAlarm( txt: text; flags: counter; priority: discrete );
SendEvent( txt: text; flags: counter; priority: discrete );
SendLogbook( txt: text; flags: counter; priority: discrete );
```

Všechny funkce zapíší text do alarmu, resp. událostí nebo systémového zápisníku.

Parametry funkcí

Příznaky (flags)	*Popis*
ALF_DISPLAY	Zobrazit alarm mezi nekvitovanými alarmy
ALF_PRINT	Tisknout alarm na tiskárně
ALF_ARCHIVE	Archivovat alarm v archívu alarmů
ALF_SIREN	Spustit sirénu
ALF_ACCEPT	Zapsat do alarmu datum a čas kvitace
ALF_STANDARDMSG	Text alarmu se nemění (jen pro interní použití)
ALF_HASPARAM	Alarm má další parametry (jen pro interní použití)

Při vzniku a zániku alarmu je pak vhodné použít předdefinované kombinace příznaků nazvané:

- **ALF_SIGNAL** - Zobrazí alarm mezi nekvitovanými alarmy, vytiskne na tiskárně, archivuje v archívu alarmů, spustí sirénu a zapíše čas kvitace alarmu (or **ALF_STANDARDMSG**)
- **ALF_NORMAL** - Zobrazí alarm mezi nekvitovanými alarmy, vytiskne na tiskárně, archivuje v archívu alarmů a zapíše čas kvitace alarmu

Priorita (priority)	*Popis*
ALP_LOW	Alarm s nízkou prioritou
ALP_MEDIUM	Alarm se střední prioritou
ALP_HIGH	Alarm s vysokou prioritou

6.8 Přístupová oprávnění

Pro zjištění aktuálně přihlášeného uživatele, přihlášení a odhlášení slouží funkce z hlavičkového souboru **<access.bah>**.

```
function UserName() of text;
function AccessLogout() of text;
function AccessLoginDlg() of text;
```

Tip!
Pro omezení některých skriptů je možné použít funkci UserName a porovnat se jménem uživatele, který má provádění skriptu povoleno.

6.9 Práce s technologickými displeji

Pro práci s technologickými displeji je potřeba vložit systémový hlavičkový soubor **<display.bah>**, obsahuje definice potřebných konstant.

```
procedure AccessDisplay( display, param: text; action: discrete );
procedure AccessDisplayParam( display, param: text; action: discrete;
userParam: text );
```

Funkce **AccessDisplay** provede s displejem požadovanou akci.

Akce	*Popis*
ACD_OPEN	Znovu otevře daný displej bez ohledu na to, je-li již otevřen.
ACD_FOCUS	Zobrazí displej (pokud není otevřen, tak jej otevře)
ACD_CLOSE	Zavře požadovaný displej
ACD_MINIMIZE	Zmenší požadovaný displej do ikony
ACD_MAXIMIZE	Zvětší displej na celou plochu aplikace Process Monitor
ACD_RESTORE	Obnoví původní rozměry displeje
ACD_PRINT	Vytiskne displej na tiskárně (je li nakonfigurována)
ACD_PRINTDLG	Zobrazí dialogové okno pro výběr tiskárny a umožní vytisknout displej
ACD_UPDATE	Překreslí požadovaný displej.
ACD_HOME	Zobrazí hlavní displej
ACD_PREV	Zobrazí předchozí displej (existuje-li)
ACD_NEXT	Zobrazí následující displej (existuje-li)
ACD_UPPER	Zobrazí nadřízený displej (existuje-li)
ACD_LOWER	Zobrazí podřízený displej (existuje-li)

Použití procedury AccessDispalyParam pro zobrazení alarmů s filtrem

display = "System Windows & Dialogs" - systémová okna a dialogy
param =
"Alarm Accept Dialog" - nekvitované alarmy
"Event Accept Dialog" - nekvitované události
"Logbook Accept Dialog" - nekvitované systémové události
"Alarm Actual Dialog" - aktuální alarmy
"Event Actual Dialog" - aktuální události
"Logbook Actual Dialog" - aktuální systémové události
"Alarm Service Window" - všechny alarmy
"Event Service Window" - všechny události
"Logbook Service Window" - všechny systémové události

action = ACD_OPEN - otevřít okno
userParam =
"/Alarm=alarmy (true/false) - zobrazovat alarmy (ano/ne)
/Event=události (true/false) - zobrazovat události (ano/ne)
/Log=systémové události (true/false) - zobrazovat systémové události (ano/ne)
/Level=úroveň alarmů (0-zánik, 1-vznik)- zobrazovat vznik/zánik
/Source=název zdroje - jen alarmy ze zdroje, zdrojů (lze použít parametr vícekrát za sebou s různými zdroji
/Style=název alarmního stylu" - jen alarmy zadaného alarmního stylu

Příklady použití

```
AccessDisplayParam( "System Windows & Dialogs", "Alarm Accept Dialog",
ACD_OPEN, "/Source=Kotelna /Style=Alarm" );
        // zobrazí nekvitované alarmy ze zdroje "Kotelna" se stylem "Alarm"

AccessDisplayParam( "System Windows & Dialogs", "Alarm Accept Dialog",
ACD_OPEN, "/Event=true /Source=Kotelna1 /Source=Kotelna2" );
        // Zobrazí nekvitované události ze zdrojů "Kotelna1" a "Kotelna2"

AccessDisplayParam( "System Windows & Dialogs", "Event Accept Dialog",
ACD_OPEN, "/Level=1" );
        // Zobrazí nekvitované události s úrovní=1
```

6.10 Práce s se zvukem

Pro práci se zvukem slouží funkce PlaySound a StopSound. Jejich deklarace naleznete v hlavičkovém souboru **<sound.bah>**.

Deklarace funkce	Popis
PlaySound(soundName: text; autoRepeat: binary)	Přehraje požadovaný zvuk. Má-li parametr autoRepeat hodnotu true, bude daný zvuk přehrávat stále dokola až do ukončení přehrávání příkazem StopSound.
StopSound(allSounds: binary)	Ukončí přehrávání zvuku. Pokud má parametr allSounds hodnotu true, vyprázdní současně frontu zvuků pro přehrání.

6.11 Práce s datem a časem

Další skupinou funkcí jsou funkce pro práci s datumem a časem. Jazyk Bára používá sekundový formát data. Určuje počet sekund od 1.1.1980. Jeho výhodou je možnost sčítání a odčítání data a času. Všechny funkce vracejí typ Counter.

Funkce	Parametr	Popis
Year	-	Vrací aktuální rok
Month	-	Vrací aktuální měsíc
Day	-	Vrací aktuální den
Hour	-	Vrací aktuální hodinu
Minute	-	Vrací aktuální minutu
Second	-	Vrací aktuální sekundu
Time	-	Vrací aktuální denní čas v sekundách od půlnoci.
Date	-	Vrací aktuální datum ve dnech od 1.1.1980
DateTime	-	Vrací aktuální datum a čas v sek. od 1.1.1980
GetSecCount	-	Vrací počet sekund od startu Windows
GetTickCount	-	Vrací počet milisekund od startu Windows
GetYear	datum	Vrací rok z data zadaného parametrem
GetMonth	datum	Vrací měsíc z data zadaného parametrem
GetDay	datum	Vrací den z data zadaného parametrem
GetHour	čas	Vrací hodinu z času zadaného parametrem
GetMin	čas	Vrací minutu z času zadaného parametrem
GetSec	čas	Vrací sekundu z času zadaného parametrem
GetDayOnWeek	GetDayOnWeek(x)	vrací den v týdnu (0-pondělí,6-neděle)
CreateDateTime	CreateDateTime(y,m,d,h,m,s)	vytvoří ze zadaného data sekundový čas

Je-li udán parametr datum nebo čas, je možno použít i funkcí vracející datum i čas společně (např. DateTime). Platí totiž rovnost:

```
DateTime() = (Date() * 86400) + Time()
```

V uvedeném případě je potřeba vynásobit hodnotu vrácenou funkcí Date (počet dní od 1.1.1980) počtem sekund za jeden den tj.

```
60 (sec/min) * 60(min/hod) * 24(hod/den) = 86400(sec/den)
```

Chci získat číslo zítřejšího dne. Volám:

```
GetDay( (Date() + 1) * 86400 )
```

Chci získat čas o hodinu vyšší, než je okamžitý:

```
Time() + 3600
```

6.12 Ostatní funkce

Ostatní nezařazené funkce jsou popsány zde.

Funkce	*Popis*
CmdLine() of text	Vrací celý obsah příkazové řádky skriptu například obsah alarmu v dynamizaci. OnAlarm
CmdLineSet(line: text)	Nastavuje obsah příkazové řádky skriptu (vhodné pro ladicí účely)
CmdLineParam(i: discrete) of text	Vrací i-tý parametr z příkazové řádky jako text (hodnoty i přepínače).
CmdLineValue(i: discrete) of text	Vrací i-tou hodnotu z příkazové řádky jako text, přeskakuje přepínače.
CmdLineOption(i: discrete) of text	Vrací i-tý přepínač z příkazové řádky jako text, přeskakuje hodnoty.
CmdLineFindOption(name: text) of text	Vyhledá a vrací jako text přepínač příkazové řádky se jménem 'name'.
CmdLineAlarm() of text	Vrací formátovaný alarm v dynamizaci OnAlarm a OnAlarmAction (vhodné pro SMS).
VarExist(variable) of binary	Vrací TRUE, pokud proměnná existuje
VarOrConst(ident) of binary	Vrací TRUE, je-li parametr proměnná či konstanta

Příklad: seznam přepínačů příkazové řádky zasílané do skriptu dynamizací OnAlarm a OnAlarmAction.

```
"Id"
"AlarmId"
"Forbidden"
"System"
"Source"
"Device"
"Text"
"Param"
"Style"
"Type"
"Priority"
"AcceptRight"
"Color"
"Time"
"WriteTime"
"Level"
"AcceptUser"
"Note"
"Mask"
```

Příklad: obsah příkazové řádky skriptu spuětěného dynamizací OnAlarm.

```
/Id=11943 /AlarmId=GEN.Sinus03 /Forbidden=0 /System="" /Source=Generator /
Device="" /Text="Porucha: GEN.Sinus0 (2.93173)!" /Param=""
/Style=Alarm /Type=0 /Priority=3 /AcceptRight={00000000-0000-0000-0000-
000000000000} /Color=255 /Time=28.08.2012-14:40:45
/WriteTime=28.08.2012-14:40:45 /Level=1 /AcceptUser="" /Note="" /Mask=3
```

Příklad: napojení RemoteDesktop po kliknutí na alarmu; dynamizace BaraScript, dynamizační podmínka OnAlarmAction.

```
global
	opt: text;
begin
	opt := CmdLineFindOption( "Source" );
	if strlen( opt ) > 0 then
		if opt = "Generator" then
			FileExecute( "mstsc.exe", "Test-W7 - Test-W7 - Remote
Desktop Connection", "e:\temp\Test-W7.rdp" );
 end.
```

Příklad: procházení pokusné příkazové řádky dynamizací typu BaraScript

```
 global
        i : counter;
         par, val : text;
begin
        CmdLineSet( "pokus /system=abc /source=George" );

        for i:= 0 to 10
        begin
                par := CmdLineOption( Counter2Discrete( i ) );
                if par = "/Source" then
                        begin
                                val := CmdLineValue( Counter2Discrete( i ) );
                                if val = "Generator" then
                                        begin
                                                // nejaka akce ....
                                        end;
                        end;
        end;
end.
```

7 Složené typy

7.1 Typ TIOChannel

Vstupně/výstupní kanály byly deklarovány jako pole hodnot některého z jednoduchých datových typů. Později byly rozšířeny o příznaky popisující jejich stav. Příznaky jsou deklarovány jako struktura základních typů a polí základních typů.

Kanál je tedy typu **Record** a je definován následovně:

```
TIOChannelXXXXX : record
  Value      : array [Array] of XXXXX;    // InOut - hlavní hodnota kanálu,
pole hodnot indexované od 0 do Array - 1
  Status     : discrete;   // Input - status kanálu - kód chyby z regulátoru
- 0=OK
  Valid: binary;     // InOut - validita hlavní hodnoty kanálu
  Array: counter;    // Input - počet prvků pole kanálu (obvykle 1, když
není pole)
  DataType   : discrete;   // Input - datový typ hlavní hodnoty kanálu
  Name : text;       // Input - jméno kanálu
  Descr: text;       // Input - popis kanálu
  TypeName   : text;       // Input - jméno typu kanálu
  Updated    : counter;    // Input - datum a čas poslední obnovy hodnoty
  Config     : text;       //  Input  -  konfigurační  řetězec  kanálu  v
regulátoru

  Lo         : analog;     // Input - minimální hodnota pro zadání, alarm
  Hi         : analog;     // Input - maximální hodnota pro zadání, alarm

  Format     : text;       // Input - formát výpisu proměnné
  Unit : text;       // Input - jednotka veličiny

  Log.Style          : text;       // Input - jméno stylu logování při zadání
hodnoty
  Alarm.Style : text;       // Input - jméno stylu alarmu při splnění alarmní
podmínky
  Alarm.State : binary;     // Input - aktuální stav splnění alarmní podmínky

  //
  // je-li trend definován - Trend.Used = true
  //   Q < 86400     - index vzorku od posledního zapsaného zpět, Q <
Trend.Count
  //   Q >= 86400    - datum a čas vzorku typu counter
  Trend.Value : array [Q] of XXXXX;      //  InOut  -  hodnota  typu  XXXXX
trendu v Q
  Trend.Valid : array [Q] of binary;     // InOut - validita vzorku trendu
v Q
  Trend.Present      : array [Q] of binary;      // InOut - vrací true, když
vzorek v Q existuje
  Trend.File   : text;                   // Input - textové jméno souboru
trendu
  Trend.Period: counter;                 //  Input  -  perioda  vzorkování
proměnné
  Trend.Count : counter;                 // Input - celkový počet zapsaných
vzorků v trendu
  Trend.Last   : counter;                // Input - datum a čas posledního
vzorku v trendu
```

```
  Trend.First : counter;                  // Input - datum a čas prvního
vzorku v trendu
  Trend.Used  : binary;                   // Input - vrací, zda je trend ke
kanálu definován
  Trend.Name  : text;                     // Input - textové jméno trendu
end;
```

kde XXXXX může být některý z typů **Analog**, **Binary**, **Counter**, **Discrete**, **Text**. V/V kanál typu TIOChannelAnalog tedy obsahuje hodnotu (Value) typu **Analog**.

Value

Příznak obsahuje hodnotu dané proměnné. Typ hodnoty je stejný jako typ kanálu. Na hodnoty proměnné se můžete odkazovat také přímo jako na kanál s vynecháním příznaku Value.

Hodnota kanálu je deklarována jako pole i v případě, že se nejedná o pole. Rozměr pole Array je v tomto případě 1. Na proměnnou se tedy lze i v tomto případě odkazovat také pomocí indexu pole jako na Proměnná[0]. Pole se indexuje od 0 po (Array - 1) včetně.

Status

Příznak Status může obsahovat dodatečné informace o proměnné, jako například překročení mezí, chybový stav čidla apod.

Array

Definuje zadanou délku pole kanálu. Pokud se jedná o samostatnou hodnotu, je Array = 1.

Příklady použití:

```
// deklarace funkce s parametrem typu analogového kanálu, musí být vždy
předávano referencí ref
function GetDaySample( ref src: TIOChannelAnalog ) of analog;

// xy jsou globální nebo lokální proměnné odpovídajících typů, c je pole
typu TIOChannelAnalog
x1 := c;                    // do x1 se přiřadí hlavní hodnota na indexu 0
if ( c.Array > 5 ) then     // pokud má pole více prvků, než 5
  begin
    x2 := c[ 5 ];           // do x2 se přiřadí hlavní hodnota na indexu 5
    x3 := c.Value[ 5 ];     // do x3 se přiřadí hlavní hodnota na indexu 5 -
totožný význam, ale plný zápis předchozího řádku
    x4 := c.Valid[ 5 ];     // do x4 se přiřadí validita vzorku na indexu 5
    x5 := c.Status;         // do x5 se přiřadí aktuální status kanálu
  end;

if (c.Trend.Used AND c.Trend.Present[ Date() ] = true ) then  // je-li trend
definován a zaznamenán vzorek o dnešní půnoci
  x6 := c.Trend.Value[ Date() ];                              // tak hodnotu
vzorku z dne3n9 p;noci p5i5a+d do x6

x7 := c.Trend.Value[ DateTime() - 3600 ];                     //      hodnota
před hodinou
x8 := ( c.Trend.Value[ 0 ] + c.Trend.Value[ 1 ] ) / 2; //      průměr      z
posledních dvou vzorků
x9 := t.Trend.Period;                                         //      perioda
trendů
xV := t.Trend.Valid[ 5 ];                                     //     validita
šestého vzorku od konce
```

7.2 Typy Desigo PX

Regulátory řady Desigo PX mají mnohem složitější logiku významu a ovládání kanálů, než jiné regulátory. Proto jsou nutné i vlastní složitější datové typy s více příznaky. Datové typy jsou definovány pro hlavní hodnoty typuAnalog, Binary, Counter, Discrete.

Kanál je tedy typu **Record** a je definován následovně:

```
TBACnetChannelXXXXX : record
  ...
  Reliability : discrete;   // Input      - informace o stavu čidla
  Manual              : binary;     // InOut      - manuální režim kanálu
  Ack          : binary;     // InOut      - kvitace poruchy kanálu
  Feedback            : analog;     // Input      - zpětné hlášení o chodu
zařízení
  OpTime              : counter;    // Input      - provozní čas zařízení
  LoLimit             : analog;     // InOut      - dolní mez, ři podkročení
může být hlášen alarm
  HiLimit             : analog;     // InOut      - horní mez, ři překročení
může být hlášen alarm
  SubstValue   : analog;     // InOut      - náhradní hodnota při poruše čidla
  Slope        : analog;     // InOut      - strmost hodnoty čidla K
(y =K*x + Q)
  Intercept           : analog;     // InOut      - posun hodnoty čidla Q
 (y =K*x + Q)
end;
```

kde XXXXX může být některý z typů **Analog**, **Binary**, **Counter**, **Discrete**, **Text**. V/V kanál typu TBACnetChanneAnalog tedy obsahuje hodnotu (Value) typu **Analog**.

7.3 Typy pro SMS modul

Příjem SMS

Typ **TLastReceivedSMS** je použit v modulu pro vysílání a příjem krátkých textových zpráv (IOSMS.IOM). Systémová proměnná **LastReceivedSMS** díky tomu obsahuje nejen text přijaté SMS ale taktéž informace o času odeslání a tel. číslo odkud byla SMS odeslána.
Typ **TLastReceivedSMS** má všechny základní příznaky jako V/V kanály rozšířené o některé další položky. Hodnota Value je deklarována jako pole typu Text o deseti prvcích.

```
TLastReceivedSMS : record
  ...
  Value      : array [10] of text;     // InOut
  ...
  Source     : array [10] of text;     // Input
  DateTime   : array [10] of counter; // Input
end;
```

Kromě standardních příznaků přibyly navíc tyto příznaky:

- **Source** - Telefonní číslo, odkud byla daná krátká textová zpráva (SMS) odeslána.
- **DateTime** - Datum a čas odeslání SMS.

Oba příznaky je možno indexovat v rozsahu 0-9. Výsledkem je tedy datum a čas + zdrojové telefonní číslo posledních deseti přijatých SMS.

Odesílání SMS

V SMS module lze přidávat kanály, které slouží pro identifikací účastníků pro příjem SMS. Každý tento kanál má navíc tyto příznaky:

```
TSendedSMS     : record
  ...
  Number              : text;       // InOut      - telefonní číslo příjemce
SMS
```

```
  NumberDesc   : text;       // InOut      - jméno příjemce SMS
  SMSGroup           : discrete;   // InOut      - skupina, do které příjemce
patří
end;
```

7.4 Příznak Manual

Některé datové typy modulů mají definován příznak "**Manual**". Příkladem může být typ "**TNitelChannel**", který je použit pro všechny proměnné V/V modulu Nitel.

```
TNitelChannelXXXXX : record
  ...
  Manual     : binary;                 // InOut
end;
```

kde XXXXX může být některý z typů Analog, Binary, Counter, Discrete. Kanál typu "**TNitelChannelAnalog**" tedy obsahuje hodnoty (Value) typu "**Analog**".

Příznak Manual znamená:

- hodnota "**True**" – proměnná je v manuálním režimu ovládání - hodnota nastavená v hlavní hodnotě kanálu se nezmění
- hodnota "**False**" – proměnná je automatickém režimu ovládání - hlavní hodnotu mění automaticky vnitřní logika regulátoru

Tip!
*Nastavením do hlavní hodnoty obvykle dojde k přepnutí příznaku "**Manual**" na hodnotu "**True**". Pro vrácení ztavu do automatického řízení regulátorem je potřeba nastavit příznak "**Manual**" zpět na hodnotu "**False**".*

8 Dodatky

8.1 Chybová hlášení

"Očekává se ')' !"
Počet pravých a levých závorek ve výrazu nebo při volání funkce neodpovídá. Zkontrolujte správnost zadání výrazu.

"Očekává se 'znak' !"
"Chyba syntaxe !"
"Neznámý identifikátor !"
Tato skupina chybových hlášení je nejčastěji zapříčiněna syntaktickou chybou při zápisu zdrojového textu. Zkontrolujte proto správnost zápisu kódu. Chyba může být také indikována v případě chybného ukončení předchozích deklarací (např. po deklaraci proměnné začneme deklarovat funkci, avšak v klíčovém slově "FUNCTION" se vytratí některé písmenko - kompilátor proto tuto zkomoleninu považuje za jméno další proměnné).

"Chyba v parametrech !"
"Chyba v typu parametru !"
"Funkce nemá žádné parametry !"
V tomto případě nastala chyba při volání procedury nebo funkce. Je chybný počet nebo typ parametrů (může být i chybně zapsán výraz na místě parametru).

"Neočekávaný konec souboru !"
Chyba je indikována v případě neočekávaného konce souboru. Program pravděpodobně není regulérně ukončen příkazem "END." (tečka je povinná). Další častou příčinou je některá neukončená poznámka (ve složených závorkách).

"Nekompatibilita typů !"
Chyba nastane při kompilaci přiřazovacího příkazu v případě, kdy nesouhlasí typy proměnné a přiřazovaného výrazu. Ujistěte se o správnosti zápisu a případně použijte konverzní funkce.

"Chyba syntaxe v deklaraci návěští !"
Zkontrolujte správnost deklarace návěští podle příručky.

"Chyba syntaxe v deklaraci proměnných !"
Vyskytla se chyba v sekci deklarace proměnných. Překontrolujte správnost deklarace podle příručky.

"Duplicitní definice návěští !"
Chyba je ohlášena, jestliže jsou nalezena dvě místa v jedné proceduře s definicí návěští shodného jména (syntax: "návěští:").

"Výsledek výrazu v podmínce musí být typu Binary !"
Výsledek výrazu v podmínce musí být vždy typu "BINARY". Zkontrolujte správnost zápisu výrazu případně použijte konverzní funkce.

"V podmínce se očekává 'THEN' !"
Nastává často v případě syntaktické chyby v podmíněném příkaze, kdy kompilátor nemůže najít klíčové slovo "THEN".

"Výsledek návratového výrazu musí být stejného typu jako funkce !"
Zde nesouhlasí typ návratového výrazu za klíčovým slovem "RETURN" s typem funkce. Překontrolujte správnost zápisu výrazu a použitých typů případně použijte konverzní funkce.

"Chyba v hlavičce procedury nebo funkce !"
Chyba v deklaraci hlavičky nebo funkce. Zkontrolujte posloupnost klíčových slov, případně závorky a středníky.

"Chybné jméno procedury nebo funkce !"
Jméno procedury se shoduje s rezervovaným slovem, je nutné je změnit.

"Chybný identifikátor !"
Použitý identifikátor není správně zapsán (chybné znaky nebo jejich posloupnost) - je třeba upravit podle dovolených pravidel.

"Duplicitní identifikátor !"
Identifikátor jména použitého v deklaraci již byl deklarován v některé předchozí části programu.

"Chybný identifikátor typu proměnné !"

Při zapisování identifikátoru jména proměnné došlo k chybě. Jediné dovolené identifikátory jsou "ANALOG", "BINARY",... , identifikátory složených typů nebo typů externích procesních proměnných.

"Příliš mnoho parametrů !"
Procedura nebo funkce může mít v této verzi maximálně 16 parametrů (lze změnit pomocí "#define").

"Chyba syntaxe v deklaraci parametrů !"
V deklaraci parametrů je syntaktická chyba. Zkontrolujte oddělovače parametrů, závorky, posloupnost jmen proměnných a jejich typů.

"Chyba v inicializační konstantě !"
Inicializační konstanta byla chybně zapsána. Podívejte se na správný zápis konstanty do kapitoly Číselné konstanty.

"Deklarované návěští není definováno !"
Návěští bylo deklarováno v hlavičce funkce, avšak nebylo určeno místo skoku.

"Chybný identifikátor typu funkce !"
Identifikátor typu funkce byl chybně zapsán. Platí obdobné jako při chybě "Chybný identifikátor typu proměnné !".

Chyby v deklaraci programových smyček

"Řídící proměnná cyklu FOR musí být typu Analog,Counter nebo Discrete!"
Řídící proměnná cyklu typu "FOR" musí být jedním z vyjmenovaných algebraických typů.

"Výraz za TO nelze porovnávat s řídící proměnnou!"
Typ omezujícího výrazu (za "TO") ve smyčce typu "FOR" musí být shodný s řídící proměnnou.

"Chyba syntaxe, očekává se TO ve smyčce typu FOR !"
Chybná syntaxe zápisu smyčky typu "FOR", nebyla nalezena povinná část zápisu smyčky - klíčové slovo "TO".

"Krok smyčky musí být stejného typu jako řídící proměnná !"
Konstanta určující krok smyčky (nepovinné) musí být stejného typu jako řídící proměnná smyčky.

"Chyba syntaxe, očekává se DO ve smyčce typu WHILE !"
Chybná syntaxe zápisu smyčky typu WHILE DO, nebyla nalezena povinná část zápisu smyčky - klíčové slovo "DO".

"Chyba syntaxe, očekává se WHILE ve smyčce typu DO !"
Chybná syntaxe zápisu smyčky typu DO WHILE, nebyla nalezena povinná část zápisu smyčky - klíčové slovo "WHILE".

Chyby při práci se složenými typy

"Typ nebyl nalezen !"
Identifikátor typu nebyl nalezen, zkontrolujte jeho jméno a syntaxi.

"Chyba syntaxe v deklaraci typů !"
"Očekává se OF v deklaraci typu pole !"
Deklarace typu neodpovídá pravidlům pro deklaraci typů popsaným v této příručce.

"Duplicitní identifikátor typu !"
Identifikátor typu již byl použit, je třeba zvolit jiný název pro typ.

"Příliš mnoho položek v deklaraci strukturovaného typu !"
Strukturovaný typ může mít v současné verzi max. 16 položek. V případě potřeby většího počtu položek, deklarujte část položek jako zvláštní typ a tento typ vnořte.

"Externí typ nesmí být položkou složeného typu !"
Položkami složeného typu smí být pouze jednoduché typy nebo uživatelem definované typy.

"Typ indexu pole musí být typu Counter !"
"Délka pole musí být v rozsahu 1 až 255 položek !"
"Index do pole je větší než délka tohoto pole !"
Index do proměnné typu pole musí být výraz typu "Counter" a musí být v rozsahu 0 <= index < rozsah, přičemž rozsah pole musí být v intervalu <1;255>.

"Očekává se '] ' !"
"Očekává se ' [' !"
Chybí otevírací nebo uzavírací závorka indexu pole. Doplňte jí nebo změňte zadání.

"Pole má příliš mnoho rozměrů !"
V současné verzi je možno vnořit pole max. do třetí úrovně - je tedy možno vytvořit maximálně třírozměrná pole.

Interní chyby

"Interní neidentifikovaná chyba !"
"Interní chyba při zápisu kódu !"

K těmto chybám by nemělo při používání nikdy dojít a jejich případný výskyt je nutno hlásit dodavateli SW.

Chyby za běhu programu

"Stack overflow !"
Chyba přetečení zásobníku. Bylo například provedeno příliš mnoho vnořených volání (rekurze).

"Division by zero !"
V programu se vyskytlo dělení nulou.

"Invalid parameter !"
Volání funkce Sleep se záporným parametrem a matematické funkce s chybným parametrem (např. odmocnina ze záporného čísla apod.)

"Bad array index !"
Odkaz na index pole větší než deklarovaná délka pole.

"Timeout occured !"
Doba běhu programu překročila definovaný maximální možný čas běhu. Do doby běhu programu se počítá pouze čas běhu programu bez přerušení funkcí "Sleep" nebo "Suspend".

"Internal error !"
Interní chyba v programu. Tato chyba by neměla nikdy nastat.

"String using error !"
Chyba při použití stringových operací. Například použití nealokovaného řetězce, zápis za max. délku řetězce (255 znaků), apod.

"User exit !"
Ukončení programu funkcí Exit.

"Using File Error !"
Volání souborové funkce s nevalidním identifikátorem souboru.

"Functions are not enabled in HW key !"
Volání funkce, která není povolena v HW klíči. Jedná se o souborové a řetězcové funkce, které je potřeba mít povoleny v HW klíči.

"External functions not installed !"
Externí funkce není instalována. Za externí funkci je považováno například funkce pro přepínání displejů, funkce pro práci s alarmy a zvukem apod. Tato chyba by neměla nikdy nastat. Může znamenat poškozenou instalaci monitorovacího systému.

8.2 Příklady uživatelských programů

Výpočet průměrné hodnoty z trendu

Následující program slouží k výpočtu průměrné hodnoty z posledních N - vzorků trendu. Vzorek trendu se zahrnuje do průměru jen v případě, že vzorek je validní.

Výpočet provádí funkce "ComputeAverage". Jako parametry se této funkci předávají reference na trendový kanál a počet vzorků pro výpočet průměru.

```
program Average;
function ComputeAverage( ref t : TIOChannelAnalog; cnt : counter )
  of analog;
local
  i, tm : counter;
  sum, c  : analog;
  period : counter;
begin
  sum := 0;
  c := 0;
  period := t.Trend.Period;              // perioda trendu
  tm := t.Trend.Last;                    // čas posledního vzorku
  for i:= 0 to cnt begin                 // z kolika vzorku počítám ??
    if( t.Trend.Valid[ tm - i * period ] ) then    // je hodnota validní ?
      begin
          sum := sum + t.Trend.Value[ tm - i * period ];     // tak přičti
hodnotu
         c := c + 1;            // a zvyš počítadlo vzorku
```

```
      end;
    end;
    if ( c = 0 ) then
      return 0        // v posledních n vzorcích není žádný validní
    else
      return sum / c;  // součet poděl počtem vzorků
end;

begin
  DBS_TMP_Average := ComputeAverage( TRND_Manual, 10 );
end.
```

Pravidelné ukládání hodnot proměnných

Program provádí vždy v danou hodinu uložení technologických proměnných **PIT_*** do proměnných databázového modulu **DB_PIT_***. V našem případě program hodnoty ukládá vždy v 5 hodin ráno. V první podmínce se testuje zda v daný den již nebyly uloženy hodnoty, a pokud ano program se ukončí. Ve druhé podmínce se testuje, zda je aktuální denní čas >5:00 hod, a pokud je podmínka splněna, jsou do proměnných **DB_PIT_*** přiřazeny hodnoty **DB_***. Na závěr je zapsán do proměnné databázového modulu **DB_LASTSAVE** aktuální čas. Dále je zajištěno ukládání hodnot databázového modulu zapsáním hodnoty true do systémové proměnné **DB_SaveData**.

```
program SaveValues;
begin
// pokud se už dnes uložilo
if ( GetDay( DB_LastSave ) = Day() ) then
  return;
// je-li po 5. hodině
// tady program nedojde, pokud už uložil - viz předchozí podmínka
if ( Hour() >= 5 ) then
  begin
    DB_Pit_Q := Pit_Q;
    DB_Pit_F := Pit_F;
    DB_Pit_P := Pit_P;
    DB_Pit_Cl := Pit_Cl;
    DB_Pit_PH := Pit_PH;
    DB_LastSave := DateTime();
    DB_SaveData := true;
  end;
end.
```

Poznámka

Pro správnou funkci programu v Monitorovacím systému ProCop je potřeba zajistit spouštění tohoto programu (jako Bára Script v dynamizacích na serveru u některého z modulů) několikrát za hodinu. Bude–li například perioda spouštění programu 3 minuty, budou hodnoty uloženy nejpozději v 5:03.

Index

- # -

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- K -

- L -

- M -

- N -

- O -

- P -

- R -

- S -

- T -

- U -

- V -

- W -

- Y -

- Z -