SKRIPTOVACÍ JAZYK PRO PROSTŘEDÍ PP2CAN

Význam skriptů **2**

Syntaxe skriptovacího jazyka **4**

Struktura skriptu *4*
Typy skriptu 5
Sekce 5
Komentáře 8

Syntaxe jazyka *8*
Všeobecný popis 8
Příkazy 12

Vestavěné funkce *15*
Konverzní funkce 16
Funkce pro práci s typem string 18
Zápis na výstup 19
Aritmetické a matematické operace 21
CAN interface 24
Datum a čas 24
Zobrazení a řízení proměnných 25
Globální proměnné 27
Další 27

Vestavěné konstanty *28*

Objekty *29*
Objekt obj_can_msg 30
Objekt obj_rs232 33
Objekty obj_vector_int a obj_vector_double 34
Objekty obj_deque_int a obj_deque_double 36
Objekt obj_csvfile 37
Objekt obj_cfgfile 38
Objekt obj_logfile 39
Objekt obj_datagrid 40
Objekt obj_datagraph 41

Interpret skriptů **42**

Prostředí interpretu *42*

Překlad skriptu a spuštění **43**

Činnost překladače *43*

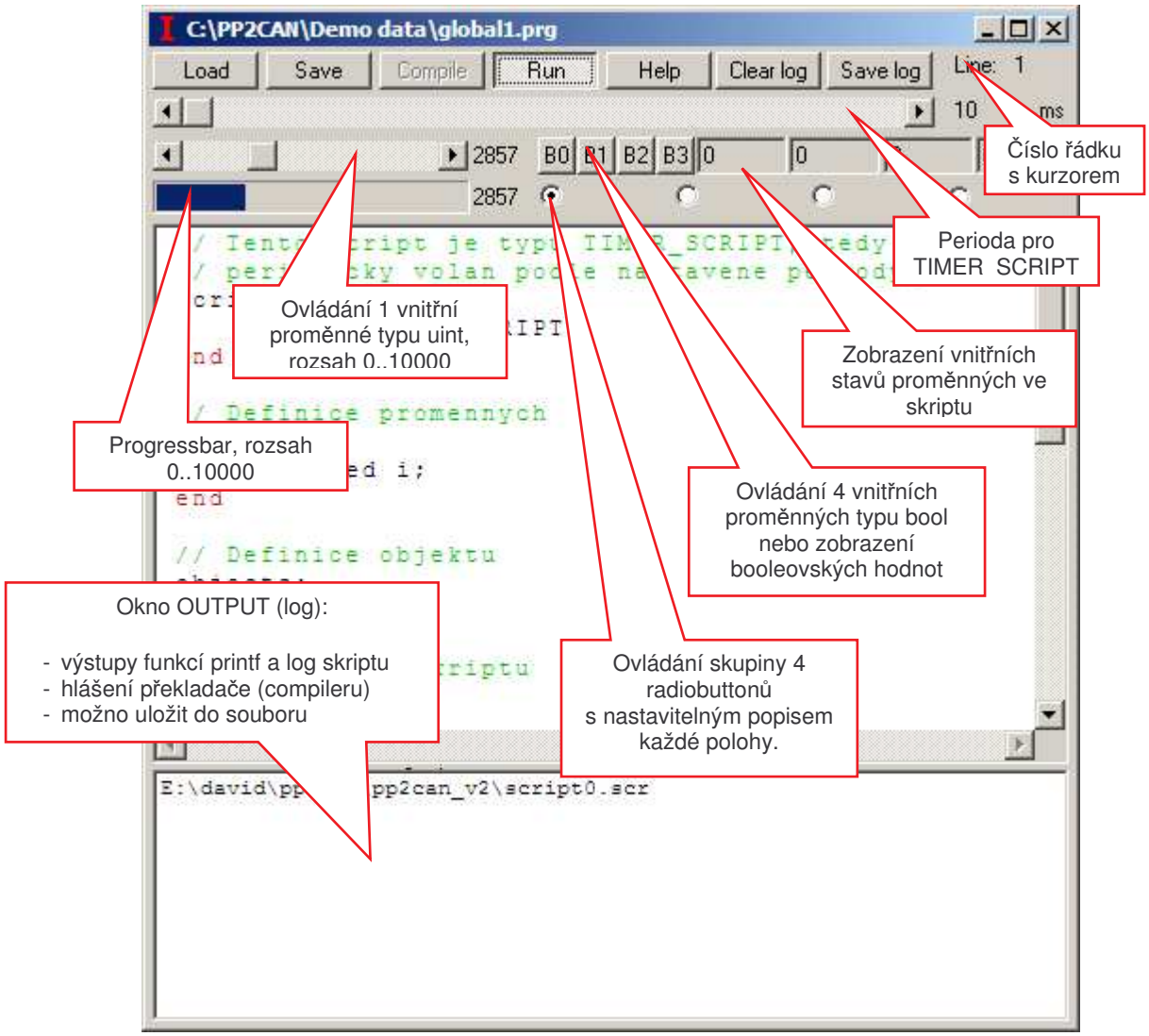
Spuštění skriptu *43*

CANLAB s.r.o.

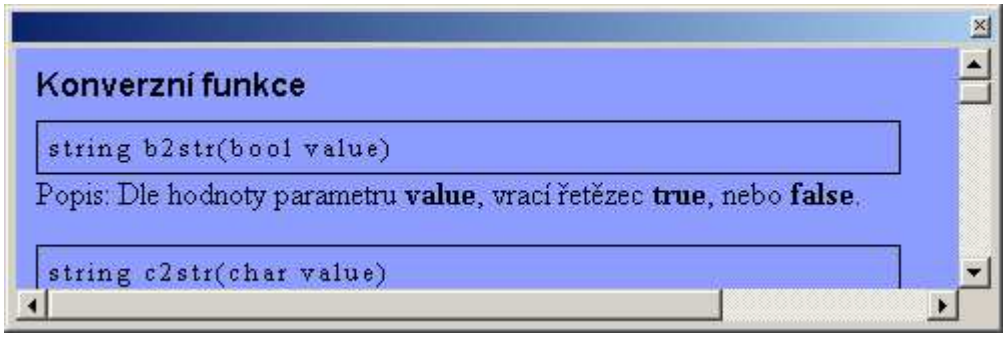
VÝZNAM SKRIPTŮ

Použití skriptů v SW PP2CAN dovoluje vytvářet uživatelům velice rychle vlastní obsluhy příchozích CANovských zpráv v případech, kdy je třeba jako reakci na příchod těchto dat generovat data na CAN sběrnici, periodicky generovat data na CAN podle zvláštních požadavjů, hlídat nejrůznější kombinace stavů z několika CAN zpráv a upozornit na ně, konfigurovat nějaké zařízení přes CAN, případně je možné skript použít i pro nejrůznější další účely.

Nástroj pro psaní a spouštění skriptů se nachází v nástrojové liště **Additional tools**, volba **CAN Script N**. V jeden okamžik lze mít aktivní 4 různé skripty.



Obr. 1: Okno CAN scriptů



Obr. 2: Okno nápovědy CAN skriptů

V adresáři s nainstalovaným SW PP2CAN (standardně C:\PP2CAN) se nalézá podadresář „**Demo data**“, který obsahuje kromě jiného i několik demonstračních skriptů s komentáři. Skripty lze testovat i bez použití HW. SW PP2CAN není vázán na nutnost připojení CAN interface. Pokud není žádný interface připojen, pracuje SW v režimu V2CAN (virtuální loopback), kdy jsou odeslané zprávy přijaty zpět jako příchozí.

Pro rychlé hledání v okně nápovědy používejte klávesovou zkratku Ctrl+F pro vyhledávání slov.

SYNTAXE SKRIPTOVACÍHO JAZYKA

Celá následující kapitola je věnována popisu jednoduchého programovacího jazyka, který je základem pro psaní skriptů pro SW PP2CAN. Syntaxe jazyka je často velmi analogická s programovacím jazykem C, avšak s jistými omezeními. Bude následovat popis všech aspektů jazyka, jeho příkazů, implementačních vlastností, ale i vestavěných funkcí a objektů.

STRUKTURA SKRIPTU

Nejprve se budeme zabývat strukturou samotného textového souboru představujícího námi napsaný a překladačem nepřeložený skript. Skripty jsou strukturovány do několika sekcí jejichž význam a syntaxe se liší dle typu sekce.

Každý skript obsahuje povinné sekce **script**, **init** a **body**. Jejich význam bude diskutován později. A dále nepovinné sekce **func_n**, **variables** a **objects** obsahující především deklarace proměnných a objektů, které mají ve skriptu vždy globální charakter. Přesnější popis bude opět následovat.

Sekce je vždy uvozena jejím jménem a dvojtečkou a následuje její tělo. Konec sekce je vyznačen klíčovým slovem **end**.

Pořadí sekcí není přímo určeno, avšak například deklarace proměnných by měly být uvedeny před samotným kódem skriptu (sekce **init, body**).

PŘÍKLAD STRUKTURY SKRIPTU:

```
// Vlastnosti skriptu
script:
    type = CAN_FUNCTION;
end

// Promenne
variables:
    string str;
    int i;
end

// Objekty
objects:
    obj_can_msg can_msg;
end

// Inicilizacni cast
init:
    str = "initialized";
end
```

```
// Vlastní telo skriptu
body:
    prints(str);
    print_endl();
end
```

Typy skriptu

Definice typu skriptu je uvedena v sekci **script**, která deklaruje jeho specifické vlastnosti, a spočívá v přiřazení typu do proměnné **type**. Typy skriptů jsou odvozeny od jejich použití.

DEFINOVÁNY JSOU TYTO TYPY:

CAN_FUNCTION
Volá se při příchodu zprávy z CANu.

TIMER_SCRIPT
Volá se periodicky dle nastavené periody (nečte příchozí zprávu).

CAN_FUNCTION_AND_TIMER
Volá se při příchodu zprávy z CANu i dle nastavené periody. K rozlišení o jaké volání se jedná slouží funkce is_timer_call a is_can_function_call.

PŘÍKLAD SCRIPT SEKCE:

```
Viz kapitolka Sekce script.
```

Sekce

Následují popisy jednotlivých sekcí skriptu.

SEKCE SCRIPT

V sekci **script** se definují základní vlastnosti skriptu. Prozatím obsahuje pouze volbu typu skriptu. Všechna nastavení se provádí přiřazením zvolené hodnoty proměnné předem daného jména.

MOŽNÉ PROMĚNNÉ:

- **type** … typ skriptu (CAN_FUNCTION, TIMER_SCRIPT)

PŘÍKLAD SCRIPT SEKCE:

```
script:
    type = CAN_FUNCTION;
end
```

SEKCE VARIABLES

Obsahem sekce jsou deklarace proměnných základních typů. Všechny takto deklarované proměnné mají globální charakter a lze je tedy užívat v následujících sekcích **init, shutdown** a **body,** sekcích se samotným kódem skriptu.

Deklarace se řídí syntaxí známou z jazyka C. Přehled všech základních typů bezprostředně následuje.

ZÁKLADNÍ TYPY:

- **bool** - nabývá hodnot **true** nebo **false**
- **char** - znak (případně hodnota v rozsahu 0-255, tedy jeden byte)
- **unsigned** - celé číslo bez znaménka (velikost 4 byty)
- **int** - celé číslo (4 byty)
- **float** - desetinné číslo (4 byty)
- **double** - desetinné číslo (8 bytu)
- **string** - řetězec jako celek, není možný přístup obdobný poli (viz. jazyk C)

STRUKTURA SEKCE:

```
variables:
    // syntaxe: type variable1[, variable2, …];
    int i, j;
    string str;
end
```

SEKCE OBJECTS

Deklarace proměnných objektového typu jsou uvedeny právě v této sekci. Proměnné objektového typu slouží k zapouzdření složitějších strukturovaných typů. Jejich vlastnosti lze měnit přístupem k atributům objektu. Těmito atributy jsou především *metody* a *proměnné*. Všechny typy objektů jsou definovány předem, jako vlastnost překladače dané verze, a nelze je definovat na uživatelské úrovni, například tak, jak to známe v jazyce C++.

Popisu implementovaných objektových typů je věnována jedna z následujících kapitol, zároveň je však nutné podotknout, že výčet objektových typů se neustále rozšiřuje v závislosti na schopnostech překladače a interpretu. Názvy všech objektových typů jsou uvozeny předložkou **obj_**.

STRUKTURA SEKCE:

```
objects:
    // syntaxe: type object1[, object2, …];
    obj_can_msg msg;
end
```

SEKCE INIT

Dostáváme se k sekci, která je naplněna kódem, tedy vlastními příkazy jazyka. Všechny příkazy uvedené v této sekci jsou považovány za inicializační a jsou provedeny pouze jednou a to ve chvíli, kdy interpret skript načte a spustí.

Sekce tedy slouží především k inicializaci deklarovaných proměnných a případně i k dalším úkonům inicializačního charakteru.

STRUKTURA SEKCE:

```
init:
    // příkazy zakončené středníkem
    …
end
```

SEKCE SHUTDOWN

Podobně jako je kód sekce INIT vykonán při startu skriptu, je kód sekce shutdown vykonán při ukončení skriptu je li tato sekce definována.

Sekce je vhodná v případech, kdy skript v sekci INIT enabluje nějaké zařízení, při ukončení je pak možné v této sekci zařízení disablovat.

STRUKTURA SEKCE:

```
shutdown:
    // příkazy zakončené středníkem
    …
end
```

SEKCE BODY

Stejně jako předchozí, obsahuje samotný výkonný kód skriptu. Její tělo obsahuje jednotlivé příkazy oddělené středníkem. Na rozdíl od **init** sekce, je tento kód často volán opakovaně. Například v *CAN_FUNCTION skriptu* bude sekce **body** provedena vždy při příchodu CAN zprávy.

S touto vlastností přímo souvisí globální charakter proměnných a objektů, protože mezi dvěmi voláními kódu **body** sekce se hodnoty proměnných neztrácejí a jsou od předešlého provedení sekce zachovány.

STRUKTURA SEKCE:

```
body:
    // příkazy zakončené středníkem
    …
end
```

SEKCE FUNC_N (N=0 .. 9)

Poslední typ sekce. Jedná se o jednoduchou implementaci funkcí bez parametrů a návratových hodnot. Ty je však možné nahradit proměnnými, které jsou platné pro celý skript. Skript může obsahovat až 10 těchto funkcí, které slouží ke zjednodušení v případě opakujících se částí kódu. Během volání funkcí je interpretem kontrolována hloubka vnoření. Je li detekováno vnoření větší než 10, je vnucena interpretru instrukce STOP a provádění skriptu je ukončeno.

STRUKTURA SEKCE:

```
func_0:
    // příkazy zakončené středníkem
    …
end

func_1:
    // příkazy zakončené středníkem
    …
end
…
…
```

Volání funkce se provádí takto:

```
func(0);    // Zavolá funkci 0, tedy vykoná příkazy mezi func_0 a příslušným end.
func(1);    // Zavolá funkci 1, tedy vykoná příkazy mezi func_1 a příslušným end.
```

Komentáře

Komentáře jsou standardní výbavou každého jazyka a ve skriptech jsou obsaženy také. Jejich charakter je naprosto shodný s komentáři ve známém jazyce C/C++. Text zapsaný jako komentář je při zpracování skriptu překladačem vynechán.

KOMENTÁŘE:

```
// toto je komentář až do konce řádku
/* komentář až do odvolání */
```

SYNTAXE JAZYKA

Nyní se budeme věnovat především obsahu **init** a **body** sekcí. Následuje tedy popis příkazů a popis práce s proměnnými základních a objektových typů.

Všeobecný popis

Jak již bylo nastíněno, syntaxe jazyka je převzata z jazyka C/C++. Jejich vlastnosti a chování je intuitivní vzhledem ke znalostem C/C++. Všechny příkazy jsou tradičně zakončeny

středníkem a klíčová slova musí být psána malými písmeny, neboť překladač rozlišuje velká a malá písmena.

Identifikátory proměnných musí začínat písmenem a následně mohou obsahovat všechna písmena abecedy, včetně číslic a znaku '_'.

KONSTANTY

Pod pojmem konstanta myslíme konstantní hodnotu určitého typu zapsanou přímo v kódu skriptu. Nejedná se tedy o konstantní proměnnou.

ČÍSELNÉ KONSTANTY TYPU UNSIGNED:

Zapisují se jako prostá celočíselná hodnota.

PŘÍKLAD:

```
u = 123456;        // konstanta typu unsigned
```

ČÍSELNÉ KONSTANTY TYPU FLOAT:

Klasické desetinné číslo, případně s kladným či záporným exponentem zapsaným bezprostředně za oddělovačem exponentu **e**. Desetinná tečka je v tomto případě nutná, včetně minimálně jedné číslice za ní.

PŘÍKLADY:

```
f = 123.456;       // konstanty typu float
f = 123.0;
f = 123.456e5;
f = 123.0e-5;
f = 123.0e+5;

f = 123.;          // toto nejsou konstanty typu float
f = 123.e5;
```

ŘETĚZCOVÉ KONSTANTY (STRING):

Tvoří je řetězec libovolných znaků uzavřený mezi dojité uvozovky. Všechny výskyty dvojznaku **\t** v řetězci jsou nahrazeny *tabulátorem* a dvojznak **\n** je nahrazen znakem *konec řádku*.

PŘÍKLAD:

```
str = "abcdef";    // konstanty typu string
str = "zakonceny radek\n";
str = "123\t253";
```

ZNAKOVÉ KONSTANTY TYPU CHAR:

První možností jejich zápisu je znak uzavřený mezi jednoduché uvozovky, druhou možností je zápis hexadecimální hodnoty znaku ve formátu **0x??**. Opět se provádí, obdobně jako u řetězců, náhrada dvojznaků **\t** a **\n** jednim odpovídajícím znakem.

PŘÍKLADY:

```
c = 'a';          // konstanty typu char
c = '\n';
c = 0x52;
c = 0xff;
c = 0xAF;
```

Poznámka: na všechny operace s konstantami (přiřazení apod.) se vztahují implicitní typové konverze prováděné překladačem (viz. dále).

IMPLICITNÍ TYPOVÁ KONVERZE

Ve všech přiřazovacích příkazech a ve volání funkcí či metod jsou prováděny standardní typové konverze z nižších základních typů na vyšší.

ZÁKLADNÍ TYPY – VZESTUPNÁ TYPOVÁ KONVERZE:

- **bool**
- **char**
- **unsigned**
- **int**
- **float**
- **double**

Dle uvedené tabulky vidíme, že překladač je při překladu schopen automaticky doplnit implicitní konverze z typu **bool** na **int**, **int** na **float** apod. Implementována je i jedna speciální konverze typu **char** na **string**.

Při práci s typem char je třeba používat konstanty (při inicializaci nebo výpočtu) v hexadecimálním tvaru, tedy takto:

char c;

c=0x1E;

c= c+0x2A;

Případně je možné pracovat s typem unsigned/int a výsledek konvertovat na char pomocí funkcí u2c/i2c (viz. dále).

Záporná čísla jsou zadávány takto:

```
double d;

d=0-1.3;
```

nebo

```
double d;

d=minusd(1);

int i;

i=minusi(i)
```

OPERÁTORY

Pro práci s proměnnými základních typů jsou definovány některé důležité operátory, jejich výčet je uveden v následujících tabulkách.

ARITMETICKÉ OPERÁTORY:

Typ proměnné	**Povolené operátory**
char, int, unsigned, float, double	+, -, *, /
char, int, unsigned	% (modulo)
int, float, double	- (unární mínus)

LOGICKÉ OPERÁTORY (VÝSLEDEK TYPU BOOL):

Typ proměnné	**Povolené operátory**
char, unsigned, int, float, double, string	>, >=, <, <=
bool, char, unsigned, int, float, double, string	==, !=
bool, char, unsigned, int, float, double	! (logická negace), && (logický and), || (logický or)

Poznámka: všechny logické operátory vrací výsledek typu **bool**, kdežto aritmetické operátory vrací výsledek příslušného typu, na který byl operátor aplikován. Binární operátor lze aplikovat pouze na proměnné shodného typu, překladač však provádí implicitní typové konverze.

Příkazy

Každý z uvedených a implementovaných příkazů musí být zakončen středníkem. Použití následujících příkazů je omezeno pouze na sekce **init** a **body**.

VÝRAZY

Před uvedením prvního příkazu si musíme nejprve definovat pojem výraz. Každý výraz je určen svou hodnotou a jejím typem. Výraz může obsahovat opět proměnné, všechny povolené operátory, volání funkcí vracejících hodnotu a konstanty. Další nedílnou součástí výrazů jsou i *kulaté závorky* **()** s jejich klasickým významem pro vyhodnocování výrazů. Výrazy jsou vyhodnocovány zleva doprava, ovšem s důrazem na priority jednotlivých operátorů (viz. tabulka).

PRIORITA OPERÁTORŮ (SESTUPNĚ SEŘAZENO):

&&, ||
==, !=
<, <=, >, >=
+, -
*, /, %
- (unární mínus), !
( )

PŘIŘAZENÍ

Slouží k přiřazení hodnoty výrazu do proměnné.

SYNTAXE:

```
variable = expression;
```

PŘÍKLADY:

```
a = 123;
b = a * 2 + b;
c = (b - a) / 2;
```

VOLÁNÍ FUNKCE

Každá funkce provádí operaci s ní spojenou. Funkce nelze vytvářet na uživatelské úrovni, ale lze volat pouze tzv. vestavěné funkce. Jejich úplný výčet si uvedeme později. Důležité je však

rozdělení na funkce s návratovou hodnotou a funkce bez návratové hodnoty (tedy spíše procedury). Funkce s návratovou hodnotou lze používat pouze ve výrazech a naopak volání procedury musí být samostatný příkaz ukončený středníkem. Překladač v tomto případě není benevolentní a kontrolu správného použití funkcí tvrdě vyžaduje.

Volání funkce se provede zápisem jména funkce a do kulatých závorek uvedeme jednotlivé její parametry oddělené čárkami. Počet a typ parametrů je pro každou funkci specifický a je třeba ho dodržet. Samozřejmě mohou existovat i funkce bez parametrů.

SYNTAXE:

```
function_name(parameters)
```

PŘÍKLADY:

```
prints("abc");
result = shr(u, 5);
```

PŘÍKAZ IF-ELSE

První z příkazů, který rozděluje chod programu do větví. Na základě pravdivosti podmínky je provedena pouze jedna z větví a to ta, pro kterou je podmínka pravdivá. Syntaxe příkazu je stejná jako v jazyce C, avšak s omezením, že tělo každé větve *musí* být uzavřeno do množinových závorek **{}**.

SYNTAXE:

```
if ( expression is true ) {
    …
}
[ else if ( expression2 is true ) {
    …
} ]
[ else {
    …
} ]
```

PŘÍKLADY:

```
if ( a == 123 ) {
    prints("123");
}
else {
    prints("not 123");
}
```

CYKLUS WHILE

Opět jeden z klasických příkazu jazyka C, nutnost uzavřít opakovanou sekci kódu do závorek však platí stále a bude platit i pro následující příkaz **for**.

SYNTAXE:

```
while ( expression is true ) {
    …
}
```

PŘÍKLAD:

```
a = 1;
while ( a <= 10 ) {
    printi(a);
    print_endl();
    a = a + 1;
}
```

CYKLUS FOR

Omezení cyklu typu for spočívá především v nutnosti uvést jeho podmínku, počáteční přiřazení hodnot proměnných a také přiřazení, která se budou provádět při každém cyklu. Jak je již naznačeno, lze v první a třetí části hlavičky for uvést pouze přiřazovací příkazy oddělené čárkou. Nutnost použití množinových závorek zůstává.

SYNTAXE:

```
for ( assign1 [, assign2, …] ; while expression is true ; assign1 [, assign2, …] ) {
    …
}
```

PŘÍKLAD:

```
for ( a = 1; a <= 10; a = a + 1 ) {
    printi(a);
    print_endl();
}
```

PŘÍKAZ SWITCH

A konečně poslední implementovaný příkaz. Jedná se o switch, tedy příkaz pro větvení kódu, se syntaxí převzatou z C. Hlavička obsahuje výraz jehož hodnota je porovnávána postupně se všemi výrazy v hlavičkách jednotlivých větví, rovnají-li se je větev provedena. Provedena je vždy pouze první platná větev, přičemž každá větev *musí* být uzavřena *klíčovým slovem* **break** a středníkem.

Poslední větev příkazu může být i klasická **default:** větev. Která je provedena v případě, že pro žádnou jinou nebyla uvedená podmínka splněna.

SYNTAXE:

```
switch ( expression ) {
    [ case expression1 [, expression2, …]:
        …
```

```
            break; ]
    …
    [ default:
            …
            break; ]
}
```

PŘÍKLAD:

```
switch ( a ) {
    case 123:
            prints("123");
            break;
    case 123 * 2:
            prints("246");
            break;
    default:
            prints("unknown");
            break;
}
```

Protože datový typ string je vestavěn jako základní datový typ, je možná i tato konstrukce:

```
switch ( s ) {     // s je typu string
    case "123":
            prints("123");
            break;
    case "246":
            prints("246");
            break;
    default:
            prints("unknown");
            break;
}
```

VESTAVĚNÉ FUNKCE

Funkce definované pro použití ve skriptech jsou pevně dány svými parametry a návratovou hodnotou, počet parametru je nutné dodržet vždy, typy parametru je nutné dodržet také, ovšem můžeme brát v potaz implicitní typové konverze.

Překladač vyžaduje správné použití funkcí s návratovou hodnotou a bez návratové hodnoty (procedury). Není **možné „zahazovat“ návratovou hodnotu** tak, jak jsme zvyklí z C.

Ve výčtu je u každé funkce uveden počet a typ parametrů, návratová hodnota (případně void, jedná-li se o proceduru), to vše v syntaxi C. Dále pak textový popis funkce.

Konverzní funkce

```
string b2str(bool value)
```

Popis: Dle hodnoty parametru **value**, vrací řetězec **true**, nebo **false**.

```
string c2str(char value)
```

Popis: Konverze znaku na řetězec.

```
string i2str(int value)
```

Popis: Konverze celočíselné hodnoty na řetězec.

```
string i2str_radix(int value, int radix)
```

Popis: Konverze celočíselné hodnoty na řetězec včetně zadaní číselné soustavy.

```
string u2str(unsigned value)
```

Popis: Konverze bezznaménkové celočíselné hodnoty na řetězec.

```
string f2str(float value)
```

Popis: Konverze desetinného čísla na řetězec.

```
string d2str(double value)
```

Popis: Konverze desetinného čísla na řetězec.

```
string f2str(float value)
```

Popis: Konverze desetinného čísla na řetězec.

```
int str2i(string value)
```

Popis: Konverze celého čísla zapsaného v řetězci na jeho číselnou hodnotu.

```
unsigned str2u(string value)
```

Popis: Konverze celého čísla zapsaného v řetězci na jeho číselnou hodnotu.

```
char str2c(string value)
```

Popis: Konverze celého čísla zapsaného v řetězci na jeho číselnou hodnotu.

```
bool str2b(string value)
```

Popis: Konverze booleovského typu (0-1,n-y,n-a) zapsaného v řetězci na jeho číselnou hodnotu. Rozhodující je pouze první znak řetězce.

```
float str2f(string value)
```

Popis: Konverze desetinného čísla zapsaného v řetězci na jeho číselnou hodnotu.

```
double str2d(string value)
```

Popis: Konverze desetinného čísla zapsaného v řetězci na jeho číselnou hodnotu.

```
int truncf(float value)
```

Popis: Vrací celou část desetinného čísla.

```
int truncd(double value)
```

Popis: Vrací celou část desetinného čísla.

```
int roundf(float value)
```

Popis: Zaokrouhlení desetinného čísla.

```
int roundd(double value)
```

Popis: Zaokrouhlení desetinného čísla.

```
char i2c(int value)
```

Popis: Přetypování hodnoty **int** na **char**. Hodnota **value** by měla být v rozsahu 0-255.

```
char u2c(unsigned value)
```

Popis: Přetypování hodnoty **unsigned** na **char**. Hodnota **value** by měla být v rozsahu 0-255.

```
unsigned i2u(int value)
```

Popis: Přetypování hodnoty **int** na **unsigned**. Hodnota **value** by měla být kladné celé číslo.

```
char bcd2dec (char value)
```

Popis: Konvertuje bajt s BCD kódem na hodnotu.

```
char dec2 bcd (char value)
```

Popis: Konvertuje bajt s hodnotou (0..99) na bajt s BCD kódem.

```
char get_bfu (unsigned value, int byte_index)
```

Popis:Funkce vrací zadaný bajt ze slova typu unsigned (index v rozsahu 0..3).

```
char get_bfi (unsigned value, int byte_index)
```

Popis:Funkce vrací zadaný bajt ze slova typu int (index v rozsahu 0..3).

```
char bitsetc(char value, int index)
char bitseti(int value, int index)
char bitsetu(unsigned value, int index)
```

Popis: Nastavuje v zadané hodnotě zvoleného typu (char, int, unsigned int) bit zadaný jako druhý parametr na 1. Takto modifikovaná hodnota je pak vrácena jako návratová hodnota.

```
char bitclrc(char value, int index)
char bitclri(int value, int index)
char bitclru(unsigned value, int index)
```

Popis: Nastavuje v zadané hodnotě zvoleného typu bit zadaný jako druhý parametr na 1. Takto modifikovaná hodnota je pak vrácena jako návratová hodnota.

```
bool bitestc(char value, int index)
bool bitesti(int value, int index)
bool bitestu(unsigned value, int index)
```

Popis: Testuje hodnotě která je zadána jako první parametr hodnotu bitu, který je zadán druhým parametrem. Má li bit hodnotu 1, vrací true, jinak vrací false.

Funkce pro práci s typem string

```
int str_length (string str)
```

Popis: Funkce vrací délku textu ve stringu, který je zadán jako první parametr.

```
int str_find (string str, string substr)
```

Popis: Funkce vrací pozici podřetězce substr v řetězci str. Není li podřetězec nalezen, vrací -1.

```
int str_find_from (string str, int startpos, string substr)
```

Popis: Funkce vrací pozici podřetězce substr v řetězci str. Hledání je zahájeno od pozice startpos. Není li podřetězec nalezen, vrací -1.

```
int str_find_first_of (string str, int startpos, string charset)
```

Popis: Funkce vrací pozici prvního výskytu některého ze znaků uvedených v charset. Hledání je zahájeno od pozice startpos. Není li znak nalezen, vrací -1.

```
int str_find_first_not_of (string str, int startpos, string charset)
```

Popis: Funkce vrací pozici prvního výskytu některého znaku, který se nenachází v charset. Hledání je zahájeno od pozice startpos. Není li znak nalezen, vrací -1.

```
void str_replace(string str, int startpos, int max, string source)
```

Popis: Funkce překopíruje do zadaného stringu str od pozice startpos maximálně max znaků ze stringu source.

```
char str_get (string str, int pos)
```

Popis: Funkce vrací znak na pozici pos. Je li zadaná pozice mimo řetězec, vrací 0.

```
int str_insert (string str, int pos, char character)
```

Popis: Funkce na pozici pos stringu str nastaví znak character.

```
void str_erase(string str, int startpos, int length)
```

Popis: Funkce vymaže ze stringu znaky počínaje pozicí startpos o délce length.

Zápis na výstup

První skupina funkci **print** provede ihned výpis na **OUTPUT.** Druha skupina funkci **log** zapisuje data do interniho textového bufferu. Výpis na **OUTPUT** se provede až zavoláním funkce log_endl().

```
void printb(bool value)
```

Popis: Zapíše hodnotu **value** do okna OUTPUT dialogu. V tomto případě tedy **true** či **false**.

```
void printc(char value, bool type)
```

Popis: Zapíše hodnotu **value** do okna OUTPUT dialogu. Je-li nastaveno **type**, pak bere hodnotu jako znak, v opačném případě jako hodnotu 0-255.

```
void prints(string value)
```

Popis: Zapíše řetězec do okna OUTPUT dialogu.

```
void printi(int value)
```

Popis: Zapíše hodnotu typu int do okna OUTPUT dialogu.

```
void printu(unsigned value)
```

Popis: Zapíše hodnotu typu unsigned do okna OUTPUT dialogu.

```
void printf(float value)
```

Popis: Zapíše hodnotu desetinného čísla do okna OUTPUT dialogu.

```
void printd(double value)
```

Popis: Zapíše hodnotu desetinného čísla do okna OUTPUT dialogu.

```
void logb(bool value)
```

Popis: Zapíše hodnotu **value** (**true**, **false**) do textového bufferu.

```
void logc(char value, bool type)
```

Popis: Zapíše hodnotu **value** do textového bufferu. Je-li nastaveno **type**, pak bere hodnotu jako znak, v opačném případě jako hodnotu 0-255.

```
void logc_hex(char value)
```

Popis: Zapíše hodnotu **value** do textového bufferu v hexadecimálním tvaru. Hodnota bajtu je zapsána vždy dvojmístně, např. 0 jako 00, 255 jako FF.

```
void logc_bin(char value)
```

Popis: Zapíše hodnotu **value** do textového bufferu v binárním tvaru. Hodnota bajtu je zapsána vždy jako předpona b následovaná 8 bity. Například 16 jako 0b00001111.

```
void logs(string value)
```

Popis: Zapíše řetězec do textového bufferu.

```
void logi(int value)
```

Popis: Zapíše hodnotu do textového bufferu.

```
void logu(unsigned value)
```

Popis: Zapíše hodnotu do textového bufferu.

```
void logf(float value)
```

Popis: Zapíše hodnotu desetinného čísla do textového bufferu.

```
void logd(double value)
```

Popis: Zapíše hodnotu desetinného čísla do textového bufferu.

```
void log_time()
```

Popis: Zapíše do textového bufferu aktuální čas. Vhodné volat na začátku zápisu informací o nějaké události či činnosti.

```
void log_endl()
```

Popis: Zapíše obsah textového bufferu do okna OUTPUT. Obsah textového bufferu je vymazán.

```
void log_endl_color(char color)
```

Popis: Zapíše obsah textového bufferu do okna OUTPUT. Obsah textového bufferu je vymazán. Pro zobrazení lze zvolit jednu ze 4 barev. 0 – cerna, 1 – cervená, 2 – zelená, 3 – modrá.

```
void log_clr()
```

Popis: Vymaže obsah textového bufferu.

```
int log_length()
```

Popis: Vrací aktuální velikost textu v textovém bufferu.

```
int log_max()
```

Popis: Vrací maximální velikost textu, který lze umístit do textového bufferu.

```
void clear_output()
```

Popis: Vymaže obsah okna OUTPUT.

Aritmetické a matematické operace

```
char shlc(char value, int count)
```

Popis: Vrací hodnotu **value** bitově posunutou o **count** bitů vlevo.

```
int shli(int value, int count)
```

Popis: Vrací hodnotu bitově posunutou vlevo o count bitů.

```
unsigned shlu(unsigned value, int count)
```

Popis: Vrací hodnotu bitově posunutou vlevo o count bitů.

```
char shrc(char value, int count)
```

Popis: Vrací hodnotu **value** bitově posunutou o **count** bitů vpravo.

```
int shri(int value, int count)
```

Popis: Vrací hodnotu bitově posunutou vpravo o count bitů..

```
unsigned shru(unsigned value, int count)
```

Popis: Vrací hodnotu bitově posunutou vpravo o count bitů.

```
char andc(char a, char b)
```

Popis: Vrací logický součin obou hodnot.

```
int andi(int a, int b)
```

Popis: Vrací logický součin obou hodnot.

```
unsigned andu(unsigned a, unsigned b)
```

Popis: Vrací logický součin obou hodnot.

```
char orc(char a, char b)
```

Popis: Vrací logický součet obou hodnot.

```
int ori(int a, int b)
```

Popis: Vrací logický součet obou hodnot.

```
unsigned oru(unsigned a, unsigned b)
```

Popis: Vrací logický součet obou hodnot.

```
char xorc(char a, char b)
```

Popis: Vrací XOR obou hodnot.

```
int xori(int a, int b)
```

Popis: Vrací XOR obou hodnot.

```
unsigned xoru(unsigned a, unsigned b)
```

Popis: Vrací XOR obou hodnot.

```
double sin(double a)
```

Popis: Vrací vypočtenou hodnotu funkce SINUS. Parametr je zadáván v radiánech.

```
double cos(double a)
```

Popis: Vrací vypočtenou hodnotu funkce COSINUS. Parametr je zadáván v radiánech.

```
double tan(double a)
```

Popis: Vrací vypočtenou hodnotu funkce TANGENT.

```
double sinh(double a)
```

Popis: Vrací vypočtenou hodnotu funkce HYPERBOLICKÝ SINUS.

```
double cosh(double a)
```

Popis: Vrací vypočtenou hodnotu funkce HYPERBOLICKÝ COSINUS.

```
double tanh(double a)
```

Popis: Vrací vypočtenou hodnotu funkce HYPERBOLICKÝ TANGENT.

```
double asin(double a)
```

Popis: Vrací vypočtenou hodnotu funkce ARKUS SINUS.

```
double acos(double a)
```

Popis: Vrací vypočtenou hodnotu funkce ARKUS COSINUS.

```
double atan(double a)
```

Popis: Vrací vypočtenou hodnotu funkce ARKUS TANGENT.

```
double log(double a)
```

Popis: Vrací vypočtenou hodnotu logaritmu (základ e) .

```
double log10(double a)
```

Popis: Vrací vypočtenou hodnotu logaritmu (základ 10) .

```
double exp(double a)
```

Popis: Vrací exponenciální hodnotu parametru a.

```
double sqrt(double a)
```

Popis: Vrací vypočtenou druhou odmocninu.

```
double pow(double a, double b)
```

Popis: Vrací mocninu a na b (=a^b).

```
double poly2(double x, double a, double c, double c)
```

Popis: Vrací vypočtenou hodnotu dle vzorce = $a*x^2 + b*x+c$.

```
data_type min?(data_type x1, data_type x2)
```

Popis: Vrací menší hodnotu ze dvou zadaných.

Datové typy	char - Název funkce	minc
	int	mini
	unsigned	minu
	float	minf
	double	mind

```
data_type max?(data_type x1, data_type x2)
```

Popis: Vrací větší hodnotu ze dvou zadaných.

Datové typy	char - Název funkce	maxc
	int	maxi
	unsigned	maxu
	float	maxf
	double	maxd

```
double linear_interpolation(double x0, double y0, double x1, double y1, double
x)
```

Popis: Vypočte lineárně interpolovanou hodnotu mezi dvěma body:

$$y = y_0 + (x - x_0)\frac{y_1 - y_0}{x_1 - x_0}$$

```
double deg2rad(double a)
```

Popis: Přepočet ze stupňů na radiány (funkce sin, cos a další požadují parametr v radiánech).

```
double rad2deg(double a)
```

Popis: Přepočet z radiánů na stupně.

CAN interface

```
unsigned x2can_tec()
```

Popis: Vrací hodnotu registru TEC (Transmit Error Counter) CAN řadiče. Čtení tohoto registru musí být povoleno v Options v SW PP2CAN.

```
unsigned x2can_rec()
```

Popis: Vrací hodnotu registru REC (Receive Error Counter) CAN řadiče. Čtení tohoto registru musí být povoleno v Options v SW PP2CAN.

```
unsigned x2can_txbl()
```

Popis: Vrací počet CAN zpráv, které čekají na odeslání na CAN sběrnici.

```
unsigned x2can_rxbl()
```

Popis: Vrací počet CAN zpráv přijatých z CANu a které čekají na zpracování.

```
void x2can_reset(int value)
```

Popis: Funkce slouží k nastavení požadované komunikační rychlosti CAN sběrnice.

Hodnoty value:	0-10k	1-20k	2-33.3k
	3-50k	4-62.5k	5-83.3k
	6-100k	7-125k	8-250k
	9-500k	10-1M	

Datum a čas

Varianta s UTC: UTC je zkratka anglického výrazu Coordinated Universal Time, koordinovaný světový čas. Někdy je nazýván také Zulu time, označován písmenem Z za časovým údajem. UTC je základem systému občanského času, jednotlivá časová pásma jsou definována svými odchylkami od UTC. UTC je jako základ systému měření času nástupcem GMT (Greenwich Mean Time – greenwichský střední čas) a v neformálním vyjadřování je s ním někdy zaměňován. Na rozdíl od GMT, který udává čas platný v časovém pásmu základního poledníku, který je založen na rotaci Země, je UTC založen na atomových hodinách, tzn. je na rotaci Země nezávislý.

```
int time_h(), int time_utc_h()
```

Popis: Vrací aktuální čas – hodina. Varianta UTC vrací Coordinated Universal Time, koordinovaný světový čas

```
int time_m(),int time_utc_m()
```

Popis: Vrací aktuální čas - minuta.

```
int time_s(),int time_utc_s()
```

Popis: Vrací aktuální čas - sekunda.

```
int time_ms(), int time_utc_ms()
```

Popis: Vrací aktuální čas - milisekundy.

```
int date_y(),int date_utc_y()
```

Popis: Vrací aktuální datum - rok.

```
int date_m(),int date_utc_m()
```

Popis: Vrací aktuální datum - měsíc.

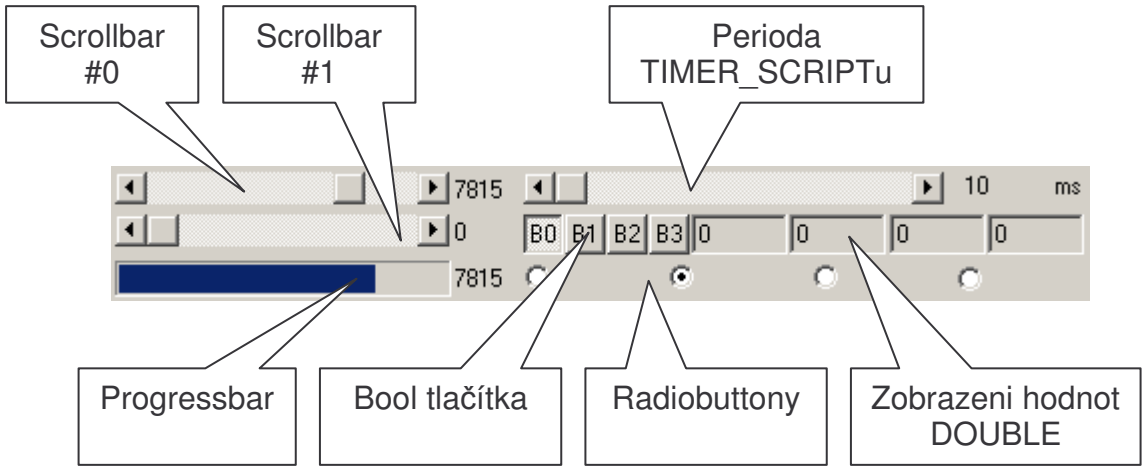
```
int date_d(),int date_utc_d()
```

Popis: Vrací aktuální datum - den.

Zobrazení a řízení proměnných

Okno dialogu skriptu dovoluje zobrazit stav několika proměnných ve skriptu. Zobrazit lze 4 proměnné typu double a až 4 proměnné typu bool a řídit jeden prvek progressbar s rozsahem 0..10000. Ovládat pak lze 2 proměnné typu unsigned hodnotou 0..10000 nebo až 4 proměnné typu bool a skupinu 4 radiobuttonu. Funkce pro čtení a modifikaci těchto proměnných mají předponu ui (user interaction).

Taktéž lze použít pro řízení hodnot přečtených z případného připojeného joysticku.




```
bool ui_get_b0 (), bool ui_get_b1 (), bool ui_get_b2 (), bool ui_get_b3 ()
```

Popis: Vrací hodnotu typu bool zvoleného tlačítka.

```
void ui_set_b0 (bool state), void ui_set_b1 (bool state)
void ui_set_b2 (bool state), void ui_set_b3 (bool state)
```

Popis: Nastavuje hodnotu zvoleného tlačítka (například pro signalizaci booleovských stavů).

```
void ui_set_r (int state)
```

Popis: Nastavuje hodnotu skupiny 4 radio-buttonů (hodnota 0..3).

```
void ui_set_name_r (int radio_button, string name)
```

Popis: Nastavuje popisek zadaného radiobuttonu.

```
int ui_get_r (void)
```

Popis: Vrací nastavení skupiny 4 radio-buttonů (hodnota 0..3).

```
void ui_set_p (unsigned state)
```

Popis: Nastavuje hodnotu progressbaru. Rozsah hodnot 0..10000.

```
void ui_set_d0 (double state), void ui_set_d1 (double state)
void ui_set_d2 (double state), void ui_set_d3 (double state)
```

Popis: Nastavuje hodnotu typu double pro zobrazení v jednom ze 4 oken.

```
unsigned ui_get_val0 (),unsigned ui_get_val1 ()
```

Popis: Vrací hodnotu typu unsigned scroolbaru. Pracovní rozsah scroolbaru je 0 až 10000.

```
void ui_set_timer_script_period (unsigned)
```

Popis: Funkce dovoluje ze skriptu nastavit periodu generování TIMER_SCRIPT. U skriptů jejichž funkce vyžaduje nastavit žádanou periodu je vhodné volat tuto funkci v sekci init. Po spuštění skriptu se pak upraví nastavení periody na ovládacím prvku automaticky na požadovanou hodnotu bez nutnosti přednastavení periody uživatelem před spuštěním.

```
unsigned ui_get_timer_script_period ()
```

Popis: Funkce dovoluje ze skriptu zjistit periodu generování TIMER_SCRIPTu.

```
int ui_joy_x (int joystick_id), int ui_joy_y (int joystick_id)
int ui_joy_z (int joystick_id), int ui_joy_r (int joystick_id)
int ui_joy_u (int joystick_id), int ui_joy_v (int joystick_id)
```

Popis: Vrací hodnotu polohy osy joysticku. Rozsah hodnot je vhodné ověřit vždy pro konkrétní joystick. Jako parametr je zadán index joysticku, 0 pro první joystick, 1 pro druhý joystick. Lze také použít konstant JOYSTICK1 a JOYSTICK2.

```
bool ui_joy_b (int joystick_id, int button_number)
```

Popis: Vrací hodnotu zadaného tlačítka joysticku.

```
int ui_joy_b ()
```

Popis: Vrací počet připojených joysticku.

Globální proměnné

Současná verze SW PP2CAN dovoluje provozovat současně 4 aktivní skripty. Tyto skripty mohou vykonávat zcela nezávislou činnost nebo navzájem spolupracovat. K jejich spolupráci pak lze využít sdílených-globálních proměnných. Jedná se o 3 pole po 128 prvků typu INT, DOUBLE a BOOL. K zápisu a čtení do/z těchto proměnných se využívá této skupiny funkcí:

```
bool global_bool_get (unsigned index);
int global_int_get (unsigned index);
double global_double_get (unsigned index);
```

Popis: Funkce čte z pole globálních proměnných hodnotu na zadaném indexu pole. Rozsah indexu je 0..127, tedy celkem 128 proměnných každého typu.

```
void global_bool_set (unsigned index bool value);
void global_int_set (unsigned index, int value);
void global_double_set (unsigned index, double value);
```

Popis: Funkce zapisuje do pole globálních proměnných hodnotu na zadaném indexu pole. Rozsah indexu je 0..127, tedy celkem 128 proměnných každého typu.

Další

```
void stop()
```

Popis: Funkce přeruší vykonávání skriptu.

```
void wait(int value)
```

Popis: Pozastaví provádění skriptu po dobu **value** milisekund. Funkci je doporučeno používat jen v nutných případech. Při velkém množství příchozích dat její použití může dojít k zahlcení programu.

```
unsigned get_run_counter()
```

Popis: Vrací hodnotu počtu spuštění body sekce skriptu.

```
void msg_box(string value)
```

Popis: Zobrazí MessageBox s textem zadaným jako parametr.

```
void msg_beep()
```

Popis: Krátké zvukové upozornění počítače.

```
string file_dialog_load (string fileext)
```

Popis: Otevře dialog pro výběr souboru k načtení dat. Parametr fileext slouží k zadání přípony souboru.

```
string file_dialog_save (string fileext)
```

Popis: Otevře dialog pro výběr souboru k uložení dat. Parametr fileext slouží k zadání přípony souboru.

```
bool is_can_function_call()
```

Popis: Funkce vrací true pokud jde o volání skriptu od příchozí CAN zprávy. Využívá se u skriptu CAN_FUNCTION_AND_TIMER k rozlišení, zda byl skript spuštěn po přijetí CAN zprávy nebo od uplynulé periody.

```
bool is_timer_call()
```

Popis: Funkce vrací true pokud jde o volání skriptu od uplynulé periody volání skriptu. Využívá se u skriptu CAN_FUNCTION_AND_TIMER k rozlišení, zda byl skript spuštěn po přijetí CAN zprávy nebo od uplynulé periody.

VESTAVĚNÉ KONSTANTY

Překladač obsahuje několik vestavěných konstant. Ve skriptu pak lze používat místo hodnot názvy těchto konstant:

M_PI	(double)3.14159265358979323846)	PI
M_PI_2	(double)1.57079632679489661923)	½ PI
M_PI_4	(double)0.78539816339744830961 6)	¼ PI
M_E	(double)2.71828182845904523536)	e
M_SQRT2	(double)1.41421356237309504880)	odmocnina čísla 2
M_SQRT1_2	(double)0.70710678118654752440 1)	½ odmocniny čísla 2
M_INT16_MIN	(int)-32768	
M_INT16_MAX	(int)32767)	
M_INT32_MIN	(int)-2147483648	
M_INT32_MAX	(int)2147483647	
M_UINT16_MIN	(unsigned int)0	
M_UINT16_MAX	(unsigned int)65535	

M_UINT32_MIN (unsigned int)0

M_UINT32_MAX (unsigned int)4294967295

M_FLOAT_MIN (float)1.175494351e-38

M_FLOAT_MAX (float)3.402823466e+38

M_DOUBLE_MIN (double)2.2250738585072014e-308

M_DOUBLE_MAX (double)1.7976931348623158e+308

Pro funkci log_endl_color:

COLOR_RED (unsigned char)0x01

COLOR_GREEN (unsigned char)0x02

COLOR_BLUE (unsigned char)0x03

Indexy joysticků :

JOYSTICK1 (int)0

JOYSTICK2 (int)1

OBJEKTY

Pojem objekt není ve skriptech z programátorského plnohodnotný. Jedná se pouze o zapouzdření skupiny *dat základních typů* a definici *množiny operací (metod)* schopných s těmito daty pracovat. Každý objekt je definován svým typem, přičemž typy objektů jsou pevně definovány překladačem a interpretem, není tedy možné definovat nový typ objektu na uživatelské úrovni ve skriptu. Všechny prvky objektů se nazývají *atributy*. Atributem může tedy být proměnná základního typu, nebo metoda.

K proměnným a metodám „uvnitř“ objektu přistupujeme pomocí *kvalifikačního operátoru* tečka. Neboli s pomocí tečkové notace známé z jazyka C (práce se strukturou apod.).

PŘÍKLADY:

```
objekt.promenna = 123;
result = objekt.metoda(1, 2);
```

Objekty používané skriptem je nutné deklarovat v sekci *objects*, kde je každé proměnné objektového typu (objektu) přiřazen jeho typ.

STRUKTURA SEKCE:

```
objects:
    // syntaxe: type object1[, object2, …];
```

```
    obj_can_msg msg;
end
```

Následuje popis funkcí, datových složek a zejména pak metod jednotlivých objektových typů. Názvy všech typů jsou uvozeny předložkou *_obj*.

Objekt obj_can_msg

Práce s CAN zprávami. Odesílaní na port apod.

DATOVÉ SLOŽKY:

`unsigned id1`

Popis: Identifikátor zprávy (11 bitová část).

`unsigned id2`

Popis: Druhá část identifikátoru zprávy (18 bitová část).

`bool stext`

Popis: Příznak zda zpráva má standardní (11 bitový) identifikátor – false, nebo rozšířený (29 bitový identifikátor) - true.

`bool rtr`

Popis: Příznak zda zpráva má nastaven příznak RTR, true-ano, false-ne.

`int size`

Popis: Počet datových bytů zprávy (0-8).

METODY:

`void send()`

Popis: Odeslání CAN zprávy na CAN sběrnici.

`unsigned get_ID29()`

Popis: Vrací identifikátor zprávy v 29 bitovém formátu.

`void set_ID29(unsigned)`

Popis: Nastavuje identifikátor zprávy (id1 a id2) zadaný jako parametr v 29 bitovém formátu.

`char data(int i)`

Popis: Vrací hodnotu **i**-tého datového bytu zprávy.

`void set_data(int i, char byte)`

Popis: Nastavení hodnoty **i**-tého datového bytu zprávy.

```
void set_bool(int i, bool value)
```

Popis: Zápis hodnoty typu **bool** na pozici **i**-tého datového bytu (zapisuje se 1 bit).

```
bool get_bool(int i)
```

Popis: Vrací hodnotu typu **bool** zapsanou na pozici **i**-tého bytu zprávy.

```
void set_int16(int i, int value), void set_int16_be(int i, int value)
```

Popis: Zápis 16 bitové hodnoty **int** s počátkem na **i**-tém bytu zprávy. Varianta _be provádí změnu endianu.

```
int get_int16(int i), int get_int16_be(int i)
```

Popis: Vrací 16 bitovou **int** hodnotu. Varianta _be provádí změnu endianu.

```
void set_int32(int i, int value), void set_int32_be(int i, int value)
```

Popis: Zápis 32 bitové hodnoty **int**. Varianta _be provádí změnu endianu.

```
int get_int32(int i), int get_int32_be(int i)
```

Popis: Přečte 32 bitovou hodnotu **int**. Varianta _be provádí změnu endianu.

```
void set_uint16(int i, unsigned value), void set_uint16_be(int i, unsigned value)
```

Popis: Zápis 16 bitové hodnoty **unsigned** s počátkem na i-tém bytu zprávy. Varianta _be provádí změnu endianu.

```
unsigned get_uint16(int i), unsigned get_uint16_be(int i)
```

Popis: Vrací 16 bitovou **unsigned** hodnotu. Varianta _be provádí změnu endianu.

```
void set_uint32(int i, unsigned value), void set_uint32_be(int i, unsigned value)
```

Popis: Zápis 32 bitové hodnoty **unsigned**. Varianta _be provádí změnu endianu.

```
unsigned get_uint32(int i), unsigned get_uint32_be(int i)
```

Popis: Přečte 32 bitovou hodnotu **unsigned**. Varianta _be provádí změnu endianu.

```
void set_float(int i, float value)
```

Popis: Zápis 32 bitové **float** hodnoty.

```
float get_float(int i)
```

Popis: Čte 32 bitovou hodnotu **float**.

```
void set_double(int i, double value)
```

Popis: Zápis 64 bitové **double** hodnoty.

```
double get_double(int i)
```

Popis: Vrací přečtenou 64 bitovou **double** hodnotu.

```
void set_bit(int i, bool value)
```

Popis: Nastavuje zadaný bit v datové části zprávy (8 datových bajtů, parametr index bitu 0-63).

```
bool get_bit(int i)
```

Popis: Vrací hodnotu bitu z datové části zprávy (8 datových bajtů, parametr index bitu 0-63).

```
void set_string(int first, int length, string str)
```

Popis: Nastavuje string str do CANovské zprávy od pozice first, nastaveno je length znaků.

```
void get_string(int first, int length, string str)
```

Popis: Do stringu str nastaví string z CANovské zprávy od pozice first, o délce length znaků.

```
void load()
```

Popis: Načtení obsahu zprávy, která vyvolala spuštění skriptu typu CAN_FUNCTION. U skriptu typu TIMER_SCRIPT nemá načtení význam.

```
void print()
```

Funkce vytiskne do okna OUTPUT obsah zprávy. Data a identifikátory jsou zobrazeny dekadicky.

Příklad výpisu:

ST ID 111-8: RTR
Zpráva má 11 bitový identifikátor a je typu RTR, DLC je nastaven na 8.

EXT ID 111-123456(29221440)-8: RTR
Zpráva má 29 bitový identifikátor, uvedena je 11 bitová část, následuje 18 bitová a v závorce je pak uveden 29 bitový formát identifikátoru. Zpráva je typu RTR, DLC je nastaven na 8.

ST ID 111-8: 0,0,0,0,0,0,0,0
Zpráva má 11 bitový identifikátor. Zpráva má 8 datových bajtů, následují hodnoty datových bajtů.

EXT ID 111-123456(29221440)-8: 0,0,0,0,0,0,0,0
Zpráva má 29 bitový identifikátor, uvedena je 11 bitová část, následuje 18 bitová a v závorce je pak uveden 29 bitový formát identifikátoru. Zpráva má 8 datových bajtů, následují hodnoty datových bajtů

```
void print_hex()
```

Funkce vytiskne do okna OUTPUT obsah zprávy. Data a identifikátory jsou zobrazeny hexadecimalně.

Příklad výpisu:

ST ID 6F-8: RTR

EXT ID 6F-1E240(1BDE240)-8: 01,02,03,04,0E,20,40,FF

Funkce get_datový_typ a set_ datový_typ je nutno používat s ohledem na délku požadovaných dat. Například set_uint32 nastavuje 32 bitové slovo, a proto index bajtu může mít hodnotu 0 .. 4.

Objekt obj_rs232

Práce se sériovým portem.

METODY:

```
bool open(unsigned value port_number, unsigned baud_rate)
```

Funkce otevře a inicializuje zadaný sériový port zadanou rychlostí. Úspěšnost otevření portu indikuje návratová hodnota true.

```
bool open_ex(unsigned value port_number, unsigned baud_rate, unsigned
byte_size, unsigned parity, unsigned stop_bits)
```

Funkce otevře a inicializuje zadaný sériový port zadanou rychlostí. Navíc je možné specifikovat počet bitů v přenášeném bajtu, paritu a počet stop-bitů. Úspěšnost otevření portu indikuje návratová hodnota true.

byte_size	4-8
parity	0-4 = no,odd,even,mark,space
stop_bits	0,1,2 = 1, 1.5, 2

```
void close()
```

Funkce uzavře sériový port.

```
bool is_ok()
```

Funkce vrací true, byl li port úspěšně otevřen.

```
bool data_received ()
```

Funkce vrací true pokud byla ze sériového portu přijata data a je možné je číst.

```
unsigned char get ()
```

Funkce vrací bajt přijatý ze sériového portu.

```
void send (unsigned char data)
```

Funkce odesílá zadaný bajt na sériový port.

```
void send_log ()
```

Funkce odesílá obsah textového bufferu log na sériový port. Obsah textového bufferu je zachován.

Objekty obj_vector_int a obj_vector_double

Jedná se o objekt, který představuje zjednodušený kontajner typu vector. Tento kontejner dovoluje pracovat s polem dat typu int nebo double.

```
int size ()
```

Vrací počet prvků ve vektoru..

```
void clear ()
```

Volání vymaže všechny prvky v kontajneru.

```
void erase ()
```

Volání vymaže prvky kontejneru který je aktuálně vybrán funkcemi first a next.

```
void push_back (typ_dat value)
```

Funkce vloží novou hodnotu jako poslední prvek kontejneru.

```
typ_dat pop_back (void)
```

Funkce vrací hodnotu posledního vloženého prvku. Prvek je voláním zároveň odstraněn.

```
void insert(typ_dat value)
```

Funkce vloží hodnotu prvku na pozici v kontejneru která je vybrána funkcemi first a next. (nastaven vnitřní iterátor) Původní prvek a prvky po něm následující jsou odsunuty za tento prvek.

```
typ_dat at (int index)
```

Funkce vrací hodnotu prvku jehož pozice je zadána hodnotou index.

```
typ_dat get ()
```

Funkce vrací hodnotu prvku jehož pozice je vybrána funkcemi first a next – tedy na který ukazuje vnitřní iterátor..

```
typ_dat first ()
```

Funkce vrací hodnotu prvního prvku v kontajneru. Nastavuje vnitřní iterátor na jeho polohu.

```
bool next_exist ()
```

Funkce true, pokud v kontejneru po volání funkce first existuje další prvek ke čtení.

```
typ_dat next ()
```

Funkce vrací hodnotu dalšího prvku v kontajneru. Nastavuje vnitřní iterátor na jeho polohu.

```
void sort ()
```

Volání funkce setřídí prvky v kontejneru od prvku s nejmenší hodnotou po prvek s největší hodnotou.

```
typ_dat sum ()
```

Funkce vrací součet hodnot všech prvků v kontajneru.

```
void min ()
```

Funkce nastavuje vnitřní iterátor na polohu prvku s nejmenší hodnotou.

```
void max ()
```

Funkce nastavuje vnitřní iterátor na polohu prvku s největší hodnotou.

```
typ_dat min_val ()
```

Funkce vrací hodnotu prvku s nejnižší hodnotou.

```
typ_dat max_val ()
```

Funkce vrací hodnotu prvku s největší hodnotou.

```
void fill (typ_dat value)
```

Funkce nastavuje všechny prvky kontejneru na zadanou hodnotu.

```
int count (int value)                                    jen varianta kontejneru int
```

Funkce vrací počet prvků se zadanou hodnotou.

```
int count (double min, double max)                  jen varianta kontejneru double
```

Funkce vrací počet prvků v zadaném rozsahu hodnot včetně hodnot min a max.

```
typ_dat xor ()                                           jen varianta kontejneru int
```

Funkce vrací XOR všech hodnot v kontajneru. Tato metoda je vhodná například pro výpočet kontrolního součtu při práci s RS232.

```
void tostring (string str)                    jen varianta kontejneru int
```

Funkce převede dolní bajty prvků vektoru na znaky řetězce který je zadán jako parametr.

Objekty obj_deque_int a obj_deque_double

Jedná se o objekt, který představuje zjednodušený kontajner typu deque (obousměrná fronta). Tento kontejner dovoluje pracovat s frontou dat typu int nebo double.

```
int size ()
```

Vrací počet prvků ve vektoru..

```
void clear ()
```

Volání vymaže všechny prvky v kontajneru.

```
void erase ()
```

Volání vymaže prvky kontejneru který je aktuálně vybrán funkcemi first a next.

```
void push_back (typ_dat value)
```

Funkce vloží novou hodnotu jako poslední prvek kontejneru.

```
void push_front (typ_dat value)
```

Funkce vloží novou hodnotu jako první prvek kontejneru.

```
typ_dat pop_back (void)
```

Funkce vrací hodnotu posledního prvku v kontajneru. Prvek je voláním zároveň odstraněn.

```
typ_dat pop_front (void)
```

Funkce vrací hodnotu prvního prvku v kontajneru. Prvek je voláním zároveň odstraněn.

```
void insert(typ_dat value)
```

Funkce vloží hodnotu prvku na pozici v kontejneru která je vybrána funkcemi first a next (nastaven vnitřní iterátor). Původní prvek a prvky po něm následující jsou odsunuty za tento prvek.

```
typ_dat get ()
```

Funkce vrací hodnotu prvku jehož pozice je vybrána funkcemi first a next – tedy na který ukazuje vnitřní iterátor.

```
typ_dat at (int index)
```

Funkce vrací hodnotu prvku jehož pozice je zadána hodnotou index.

```
typ_dat first ()
```

Funkce vrací hodnotu prvního prvku v kontajneru. Nastavuje vnitřní iterátor na jeho polohu.

```
bool next_exist ()
```

Funkce vrací true, pokud v kontejneru po volání funkce first existuje další prvek ke čtení.

```
typ_dat next ()
```

Funkce vrací hodnotu dalšího prvku v kontajneru. Nastavuje vnitřní iterátor na jeho polohu.

```
void tostring (string str)                    jen varianta kontejneru int
```

Funkce převede dolní bajty prvků fronty na znaky řetězce který je zadán jako parametr.

Od verze překladače 1.10 včetně došlo k nahrazení kontejnerové metody *get* za metodu *at*. Metoda *get* dostala jiný význam.

Objekt obj_csvfile

Objekt zprostředkovává možnost práce se soubory csv. Tyto soubory mohou obsahovat různá tabulková data apod. Jedná se o textové soubory ve tvaru uvedeném v následujícím příkladu:

1,112,1,zapnuto
2,200,7,vypnuto
3,300,7,porucha 1
4,200,1,porucha 2

Tyto soubory lze snadno importovat do programu Excel nebo opačně data z Excelu exportovat do tohoto formátu.

METODY:

```
void  read (string file)
```

Funkce přečte soubor zadaný jako parametr.

```
void  write (string file)
```

Funkce zapíše data do souboru zadaného jako parametr.

```
void  set (unsigned row, unsigned col, string value)
```

Funkce zapíše data na zadaný sloupec a řádek zadanou hodnotu.

```
void get (unsigned row, unsigned col, string value)
```

Funkce čte data ze zadaného sloupce a řádku.

Funkce set a get pracují s hodnotou která má textový tvar. Pro konverzi na číslo je nutno využít příslušné konverzní funkce jako f2str, i2str, str2f, str2i a další uvedené v kapitole Konverzní funkce.

```
unsigned size_r ()
```

Funkce vrací počet řádků souboru csv.

```
unsigned size_c ()
```

Funkce vrací počet sloupců souboru csv.

```
void reserve (unsigned rows, unsigned cols, string value)
```

Funkce rezervuje v souboru patřičné místo je li vytvářen nový soubor nebo zvětšován načtený soubor .

Objekt obj_cfgfile

Objekt zprostředkovává možnost práce se soubory cfg/ini. Tyto soubory mohou obsahovat různá konfigurační data a nastavení. Jedná se o textové soubory ve tvaru uvedeném v následujícím příkladu:

```
[sekce_1]
par1=1
par2=y
par3=data
par4=1.35

[sekce_2]
par1=3
par2=n
par3=no_data
par4=13.1

[sekce_3]
par1=1.5673
par2=341
```

Sekce mohou mít parametry stejných jmen. Parametry mohou být typu INT, UNSIGNED, FLOAT, BOOL (1,0,y,n,yes,no) a STRING.

```
void set_name (string file)
```

Funkce nastavuje konfigurační soubor ze kterého nebo do kterého budou zapisovány data-parametry.

```
int read_int (string section, string item, string default_value);
unsigned read_unsigned (string section, string item, string default_value);
float read_float (string section, string item, string default_value);
bool read_bool (string section, string item, string default_value);
string read_string (string section, string item, string default_value);
```

Funkce čtou parametr s názvem zadaným v item z příslušné sekce section. Není li takto pojmenovaný parametr nalezen, je vrácena hodnota default_value která je zadávána jako text.

```
void write_int (string section, string item, int value);
void write_unsigned (string section, string item, unsigned value);
void write_float (string section, string item, float value);
void write_bool (string section, string item, bool value);
void write_string (string section, string item, string value);
```

Funkce zapisují parametr pod názvem zadaným v item do příslušné sekce section.

Objekt obj_logfile

Objekt zprostředkovává možnost zápisu do textových souborů, například logovacích informací.

```
void set_name (string file)
```

Funkce nastavuje textový/logovací soubor do kterého se budou zapisovat informace.

```
void write_log (void)
```

Funkce zapisuje obsah logovaciho bufferu do souboru. Za řetězec vložen a zapsán zna konce řádku.

```
void write_string (string str)
```

Funkce zapisuje zadaný string do souboru.

```
void write_string_endl (string str)
```

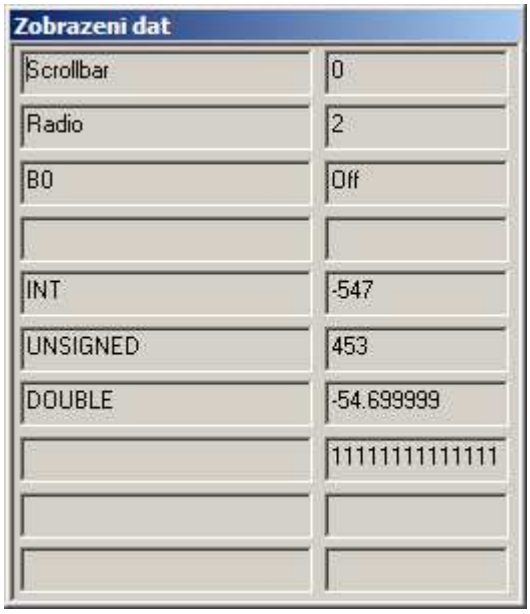
Funkce zapisuje zadaný string do souboru. Za string je vložen a zapsán znak konce řádku.

```
void close (void)
```

Funkce uzavře logovací soubor.

Objekt obj_datagrid

Tento objekt má podobné použití jako některé funkce ui_xxx. Je určen k výpisu veličin a informací ze skriptu. Na rozdíl od funkcí ui_xxx dovoluje vypisovat více informací a je možné používat více těchto objekt.



Obr. 3 Okno objektu obj_datagrid.

Okno dovoluje nastavit zobrazovaný název veličiny a vlastní veličinu samostatně. Veličin může být maximálně 10 s indexem 0..9.

```
void open(string name)
```

Funkce otevře okno objektu datagrid. Parametr name dovoluje nastavit název tohoto okna.

```
void set_name(int index,string name)
```

Funkce nastaví název zadané veličiny.

```
void set_value_string(int index,string value)
```

Funkce nastaví zadanou veličinu. Veličina je zadána jako string.

```
void set_value_int(int index,int value)
```

Funkce nastaví zadanou veličinu. Veličina je zadána jako int.

```
void set_value_unsigned(int index,unsigned value)
```

Funkce nastaví zadanou veličinu. Veličina je zadána jako unsigned.

```
void set_value_double(int index,double value)
```

Funkce nastaví zadanou veličinu. Veličina je zadána jako double.

```
void update(void)
```

Po zavolání této metody dojde k zobrazené hodnot zadaných funkcemi set_name a set_value_xxx v dialogu.

Objekt obj_datagraph

Tento objekt dovoluje v jednoduché formě zobrazit data formou grafu. Zobrazovaná data musejí být v rozsahu 0..60000. Rozsah lze v tomto rozmezí upravovat. Současně mohou být zobrazeny v jednom okně objektu 4 grafy / veličiny.

Obr. 4 Okno objektu obj_datagraph.

```
void open(string name)
```

Funkce otevře okno objektu datagraph. Parametr name dovoluje nastavit název tohoto okna.

```
void set_value(int index,int value)
```

Funkce nastaví hodnotu zadané veličiny. Veličina je zadána jako int.

```
void set_min(int value)
```

Funkce nastaví hodnotu minima grafu (rozsah a popisek). Rozsah v rozmezí 0..59999.

```
void set_max(int value)
```

Funkce nastaví hodnotu maxima grafu (rozsah a popisek). Rozsah v rozmezí 1..60000.

```
void update(void)
```

Po zavolání této metody dojde k překreslení grafu v dialogu.

```
void show_as_bar (int index,bool bar)
```

Funkce dovoluje nastavit sloupcový typ grafu.

INTERPRET SKRIPTŮ

Skripty přeložené do jejich binární podoby mohou být provedeny interpretem. Interpret je integrován do SW PP2CAN a je schopen provést kód přeloženého skriptu.

PROSTŘEDÍ INTERPRETU

Několik poznámek věnujeme charakteristice prostředí interpretu. Interpret obsahuje paměťová místa pro pomocné proměnné využívané skriptem. Pomocné proměnné přiděluje překladač a nejsou jinak přístupné. Obsah těchto proměnných se po provedení skriptu nemaže.

Paměť alokovaná pro proměnné deklarované uživatelem v rámci skriptu jsou přímou součástí skriptu a jejich hodnoty mezi jeho opakovaným spuštěním zůstávají zachovány! Inicializaci hodnot (například počítadlo spuštění skriptu) pak provádí *init* sekce skriptu.

Důležitou vlastností interpretu je i schopnost jednoduché detekce nekonečné smyčky v kódu skriptu na základě časového limitu pro jeho provedení. Při překročení limitu je skript násilně ukončen.

Inicializační část skriptů je provedena při startu skriptu. V této části je vhodné inicializovat proměnné (není li je třeba inicializovat při každém spuštění skriptu), identifikátory objektů CAN zpráv pokud není třeba je měnit za běhu, vypsat informaci o běhu skriptu a podobně.

Je li třeba zobrazovat nějaké vnitřní stavy skriptu nebo získaná data, je možné zapisovat data do OUTPUT okna, případně využít funkcí ui_xxx nebo objekt obj_datagrid. Tyto funkce lze také použít pro řízení chování skriptu.

PŘEKLAD SKRIPTU A SPUŠTĚNÍ

Překladač je implementován v samostatném programu Compiler.exe. Překlad souborů do jejich binární podoby je nutný pro interpret skriptů. Překladač se vola na editovaný skript po stisku tlačítka **Compile**. Chybová hlášení nebo hlášení o připravenosti se vypisují do okna OUTPUT.

ČINNOST PŘEKLADAČE

Co překladač se skriptem provádí? Prochází text skriptu a kontroluje syntaxi jeho obsahu (správný zápis příkazů), zároveň kontroluje sémantiku, což znamená, že ověřuje například existence deklarací proměnných, počty parametrů funkcí apod. Hlášení o nalezených chybách vypisuje na standardní výstup. Součást tohoto výstupu je informace o řádku, na kterém byla chyba nalezena. Proběhne-li veškerá kontrola bez problémů, přeloží překladač skript do tzv. bytecodu, obdoba jakéhosi assembleru, proveditelného interpretem.

Přímá interpretace skriptů je nemyslitelná zejména z důvodů časové náročnosti syntaktických a sémantických kontrol.

SPUŠTĚNÍ SKRIPTU

Je li skript přeložen (hlášení Script ready), je možné jej spustit. Jeho spuštění se provede aktivací tlačítka **Run** v okně CAN Scriptu. Tímto okamžikem je skript vyvolán vždy při příchodu zprávy z CANu pokud se jedná o skript CAN_FUNCTION, nebo je spouštěn pravidelně podle nastavené periody a to v případě, že se jedná o skript typu TIMER_SCRIPT. Případně je možné zvolit skript CAN_FUNCTION_AND_TIMER který se spouští v obou případech.

Funkce print a log mohou za běhu skriptu zapisovat data do okna OUTPUT. Obsah tohoto okna lze uložit do textového souboru. Okno OUTPUT tak lze využít pro výpis dat / měřených / vypočtených hodnot a následně hodnoty uložit pro pozdější zpracování / analýzu do textového souboru. Vhodně formátovaný textový soubor je možné načíst například do programu Excel apod.