

Skriptovací jazyk pro prostředí PP2CAN

Uživatelská příručka

Překladač a interpret verze 3.00

CANLAB s.r.o. · www.canlab.cz

Obsah

Obsah.....	2
Úvod.....	4
Význam skriptů	4
První skript.....	6
Struktura skriptu.....	7
Typy skriptů	7
Proměnné a objekty	7
Komentáře	8
Datové typy a konstanty.....	9
Zápis konstant.....	9
Hexadecimální a binární zápis	9
Implicitní konverze	10
Přetypování.....	10
Výrazy a operátory.....	11
Přiřazení.....	11
Příkazy.....	13
if, else if, else	13
while	13
for	13
switch.....	13
Ukončení skriptu.....	13
Funkce.....	14
Uživatelské funkce	14
Sekce func_0 až func_9	14
Vestavěné funkce	15
Konverzní funkce	15
Funkce pro práci s typem string	17
Zápis na výstup	18
Aritmetické a matematické operace	19
CAN interface.....	22
Datum a čas	22
Zobrazení a řízení proměnných	23
Globální proměnné.....	25
Další funkce.....	26
Vestavěné konstanty	27
Objekty	34
Přiřazení objektů.....	34
obj_can_msg.....	35
obj_lin	38

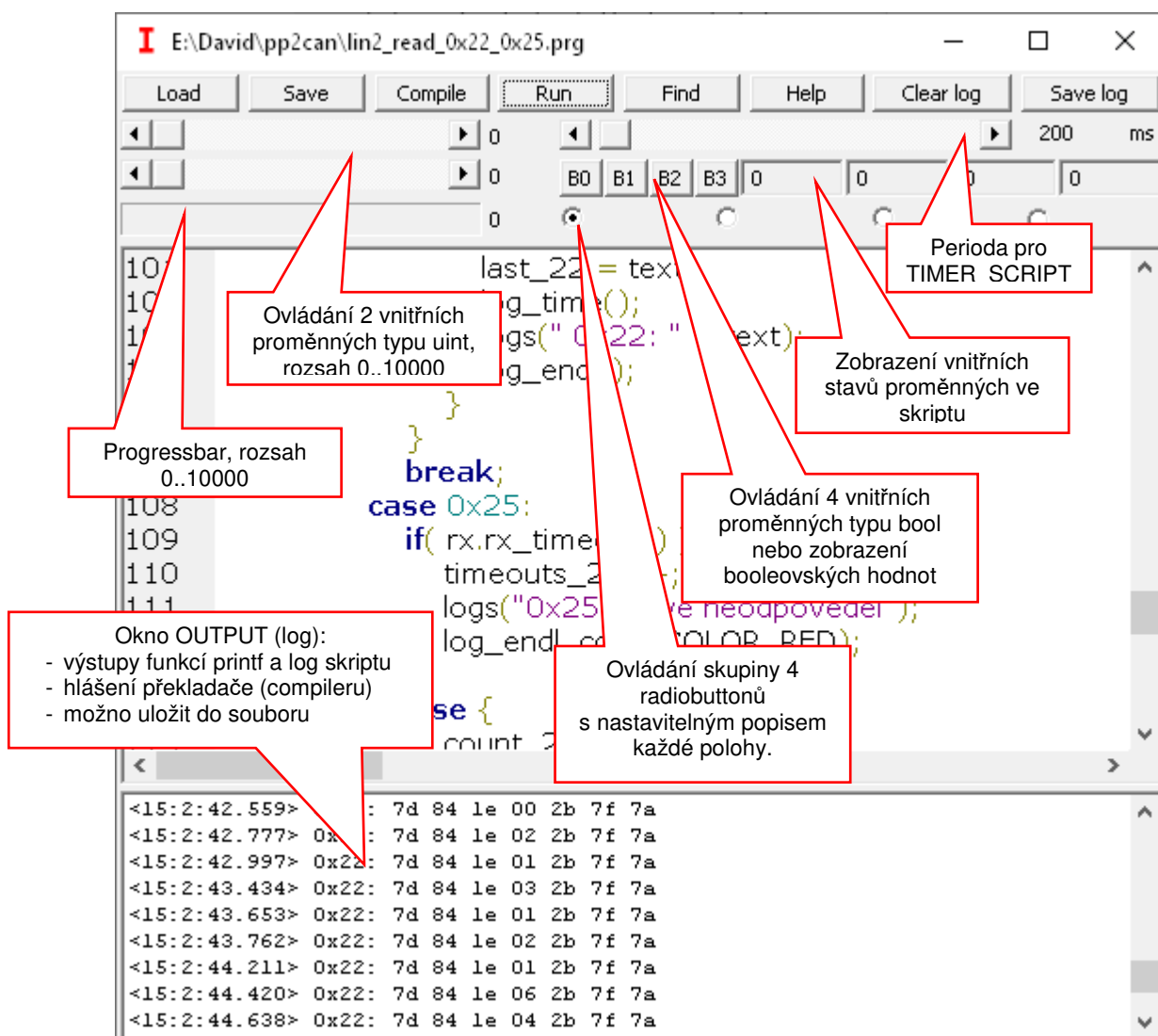
obj_rs232	39
obj_vector_int	40
obj_vector_double	42
obj_deque_int	44
obj_deque_double	45
obj_timepoint	46
obj_csvfile	46
obj_cfgfile	47
obj_logfile	47
obj_datagrid	48
obj_dataset	49
obj_datagraph	50
obj_datagauge	50
obj_socket_server	51
obj_socket_client	52
obj_pid	52
obj_memory	53
obj_dllfnc	54
obj_uds	56
obj_j1939	58
Přenos dlouhých dat (UDS, J1939)	60
Odesílání	60
Příjem	60
Protokoly obj_j1939	60
Příklad: UDS čtení DID (0x22)	60
Příklad: J1939 odeslání přes RTS/CTS a BAM	61
LIN	63
Prostředí interpretu	64
Překladač	65
Ukázkové a testovací skripty	66
Samokontrolní testy	66
Ukázky a dvojice skriptů	66
Seznam skriptů	66

Úvod

Význam skriptů

Skripty v programu PP2CAN umožňují rychle vytvářet vlastní obsluhu CAN komunikace: reagovat na přijaté zprávy, periodicky vysílat data, hlídat kombinace stavů z více zpráv a upozorňovat na ně, konfigurovat zařízení přes CAN, komunikovat přes UDS nebo J1939, pracovat s LIN, se sériovým portem, s TCP/UDP a se soubory. Skriptovací jazyk vychází z jazyka C a je doplněn o vestavěné funkce a objekty pro práci s CAN zprávami, kontejnery dat, okny pro zobrazení a soubory.

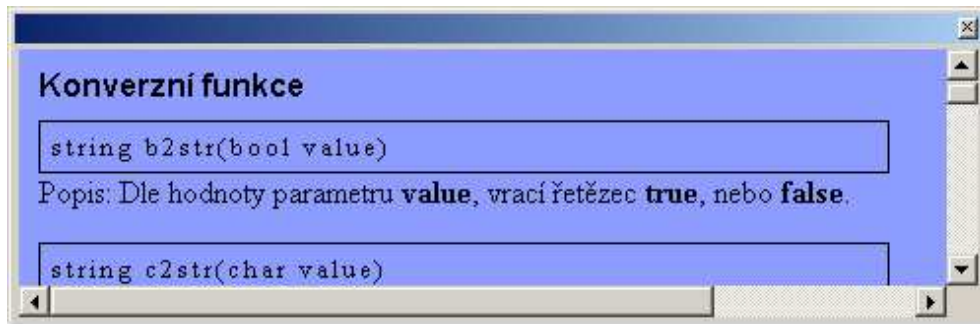
Nástroj pro psaní a spouštění skriptů je v nástrojové liště Additional tools, volba CAN Script N. Současně mohou být aktivní 4 skripty, které mohou sdílet data přes globální proměnné.



Obr. 1: Okno CAN skriptu

Prvek okna	Význam
Load, Save	Načtení a uložení zdrojového textu skriptu (.prg).
Compile	Překlad skriptu. Chyby nebo hlášení Script ready se vypíší do okna OUTPUT.
Run	Spuštění a zastavení přeloženého skriptu.
Help	Okno nápovědy s popisem jazyka, funkcí a objektů.
Clear log, Save log	Vymazání a uložení obsahu okna OUTPUT.

Prvek okna	Význam
Perioda (ms)	Perioda volání skriptu časovačem (TIMER_SCRIPT a skripty AND_TIMER).
Dva posuvníky	Hodnoty 0 až 10000 čitelné funkcemi ui_get_val0 a ui_get_val1.
Progressbar	Ukazatel 0 až 10000 nastavovaný funkcí ui_set_p.
Tlačítka B0 až B3	Čtyři logické hodnoty (ui_get_b0 ... ui_set_b3).
Čtyři pole hodnot	Zobrazení hodnot double funkcemi ui_set_d0 až ui_set_d3.
Přepínače	Skupina 4 přepínačů s popisky (ui_get_r, ui_set_r, ui_set_name_r).
Okno OUTPUT	Výstup funkcí print a log a hlášení překladače; lze uložit do souboru.



Obr. 2: Okno nápovědy CAN skriptů

V okně nápovědy lze hledat klávesovou zkratkou Ctrl+F. Skripty lze zkoušet i bez převodníku: bez připojeného rozhraní pracuje PP2CAN v režimu V2CAN (virtuální smyčka), ve kterém se odeslané zprávy vracejí jako přijaté.

K příručce patří sada komentovaných ukázkových a testovacích skriptů (kapitola Ukázkové a testovací skripty). Na příslušné skripty odkazují poznámky „Ukázka:“ v jednotlivých kapitolách.

První skript

Následující skript vysílá každých 100 ms zprávu s identifikátorem 0x123, ve které je 16bitový čítač, a do okna OUTPUT vypisuje každé desáté odeslání.

```
script:
    type = TIMER_SCRIPT;          // body se vola periodicky
end

variables:
    unsigned counter;
end

objects:
    obj_can_msg msg;
end

init:
    counter = 0;
    msg.id1 = 0x123;
    msg.size = 2;
    ui_set_timer_script_period(100);
end

body:
    counter++;
    msg.set_uint16(0, counter);
    msg.send();
    if( counter % 10 == 0 ) {
        logs("Odeslano: ");
        logu(counter);
        log_endl();
    }
end
```

- Tlačítkem Compile se skript přeloží; v okně OUTPUT se objeví Script ready nebo seznam chyb s čísly řádků.
- Tlačítkem Run se skript spustí: provede se sekce init a pak se periodicky volá sekce body.
- Dalším stiskem Run se skript zastaví (provede se sekce shutdown, pokud existuje).

Struktura skriptu

Skript je textový soubor (přípona .prg) rozdělený do sekcí. Každá sekce začíná svým jménem s dvojtečkou a končí klíčovým slovem end. Povinná je sekce script; ostatní sekce se použijí podle potřeby. Na pořadí sekcí nezáleží, deklarace proměnných a objektů je ale vhodné uvádět na začátku.

Sekce	Význam
script:	Vlastnosti skriptu, zejména typ: type = TIMER_SCRIPT;
variables:	Deklarace globálních proměnných základních typů.
objects:	Deklarace objektů (CAN zpráva, vektor, okno, soubor ...).
init:	Kód provedený jednou při spuštění skriptu.
body:	Hlavní kód, provádí se při každém volání skriptu.
shutdown:	Kód provedený jednou při zastavení skriptu.
func_0: až func_9:	Jednoduché funkce bez parametrů, volání func(N);
function ... end	Uživatelská funkce s parametry a návratovou hodnotou.

Typy skriptů

Typ skriptu určuje, kdy PP2CAN volá sekci body.

Typ	Volání sekce body
CAN_FUNCTION	Pro každou přijatou CAN zprávu; zprávu načte metoda load() objektu obj_can_msg.
TIMER_SCRIPT	Periodicky podle periody v okně skriptu nebo ui_set_timer_script_period().
CAN_FUNCTION_AND_TIMER	Pro přijaté CAN zprávy i periodicky; rozlišení funkcemi is_can_function_call() a is_timer_call().
DATA_GENERATOR_SCRIPT	Volá generátor dat PP2CAN pro generovanou zprávu.
LIN_FUNCTION	Pro každou přijatou LIN zprávu; zprávu načte metoda load() objektu obj_lin.
LIN_FUNCTION_AND_TIMER	Pro přijaté LIN zprávy i periodicky; rozlišení funkcemi is_lin_function_call() a is_timer_call().

Proměnné a objekty

Proměnné se deklarují v sekci variables, objekty v sekci objects, vždy ve tvaru typ jméno1, jméno2;. Jsou globální: platí ve všech sekcích a funkcích a mezi voláními sekce body si zachovávají hodnotu. Při spuštění skriptu mají proměnné hodnotu 0, řetězce jsou prázdné a logické proměnné mají hodnotu false.

```
variables:
    int i, j;
    double voltage;
    string name;
end

objects:
    obj_can_msg rx_msg, tx_msg;
    obj_vector_int buffer;
end
```

Identifikátory začínají písmenem a mohou obsahovat písmena, číslice a podtržítka. Rozlišují se malá a velká písmena; klíčová slova se píší malými písmeny.

Komentáře

```
// komentár do konce radku  
/* komentár pres  
   více radku */
```

Datové typy a konstanty

Typ	Velikost	Rozsah / význam
bool	1 bajt	true, false
char	8 bitů	0 až 255 (bajt nebo znak)
unsigned	32 bitů	0 až 4294967295
int	32 bitů	-2147483648 až 2147483647
unsigned64	64 bitů	0 až 18446744073709551615
int64	64 bitů	-9223372036854775808 až 9223372036854775807
float	32 bitů	desetinné číslo, asi 7 platných číslic
double	64 bitů	desetinné číslo, asi 15 platných číslic
string	max. 1023 znaků	textový řetězec; delší výsledek se zkrátí

Zápis konstant

Zápis	Typ	Příklad
celé číslo	unsigned, nad 32 bitů unsigned64	100, 4294967296
desetinné číslo (s tečkou)	float	1.5, 2.5e-3, 1.0e10
0x..., hex(...)	char do 0xFF, unsigned do 0xFFFFFFFF, jinak unsigned64	0xFF, 0x18EA0B66, hex(ABC)
0b..., bin(...)	jako hexadecimální	0b1010, bin(11110000)
znak v apostrofech	char	'A', '\n', '\t'
řetězec v uvozovkách	string	"text", "a\tb", "\"", "\\"
true, false	bool	

- Záporné číslo se zapíše unárním minus: -1, -2.5. Konstanta 1 je typu unsigned, výraz -1 je typu int.
- Desetinné konstanty jsou typu float. Pro výpočet s plnou přesností double stačí, aby byl jeden operand typu double, např. (double)1 / 3.
- Celočíselnou konstantu 0 až 255 lze přímo přiřadit do proměnné char nebo předat jako parametr char (c = 0;). Jiné hodnoty typu unsigned do char vyžadují přetypování.
- Ve skriptu jsou k dispozici vestavěné konstanty, např. M_PI, M_INT32_MAX, COLOR_RED (kapitola Vestavěné konstanty).

Ukázka: T01_data_types_constants.prg

Hexadecimální a binární zápis

Celočíselnou hodnotu lze zapsat hexadecimálně nebo binárně dvěma způsoby: s předponou (0x..., 0b...) nebo zápisem hex(...) a bin(...). Velikost písmen hexadecimálních číslic nehraje roli. Délka není omezena na bajt; konstanta může mít až 64 bitů.

```

c = 0xA5;           // char
c = bin(10100101);  // char, stejná hodnota 0xA5
u = 0x1234ABCD;     // unsigned
u = hex(DEADBEEF);  // unsigned
u = 0b1010 | 0x80;  // bitové operace s konstantami
u1 = 0x123456789ABCDEF0; // unsigned64
i = -0x10;          // int -16
msg.set_ID29(0x18FEF100); // 29bitový identifikátor

```

Hodnota konstanty	Typ konstanty	Lze přiřadit do
0x00 až 0xFF	char	char, unsigned, int, unsigned64, int64, float, double
0x100 až 0xFFFFFFFF	unsigned	unsigned, int, unsigned64, int64, float, double
nad 0xFFFFFFFF	unsigned64	unsigned64, int64, double

- Binární konstanty se řídí stejnými pravidly podle své hodnoty (0b11111111 je char, 0b100000000 je unsigned).
- Přiřazení většího typu do menšího je chyba překladu: `c = 0x1FF`; ani `u = 0x100000000`; nelze. Hodnotu lze zúžit přetypováním: `c = (char)0x1FF`;
- Přiřazení do typu se znaménkem převezme hodnotu bitově: `i = 0xFFFFFFFF`; dá -1, `l = 0x8000000000000000`; dá nejmenší int64.
- Hexadecimálně a binárně lze hodnoty i vypisovat: `logc_hex`, `logc_bin`, `logu_hex`, `logu_bin`, `logu64_hex`, `logu64_bin`.
- Řetězec s hexadecimálním číslem převede na hodnotu funkce `from_hex("1A2B")`; hexadecimální data přijímají metody `data_from_hex` (`obj_can_msg`) a `from_hex_string` (`obj_vector_int`).

Implicitní konverze

Typy jsou seřazeny `bool < char < unsigned < int < float < unsigned64 < int64 < double`. Ve výrazu s operandy různých typů se menší typ automaticky převede na větší; float s 64bitovým celým číslem dává double a int s unsigned64 dává int64. Do string lze automaticky převést char.

Přiřazení a předání parametru funkci dovolí jen převod na větší typ. Převod na menší typ (např. double do int) je chyba překladu a vyžaduje přetypování.

Pozor: při porovnání int s unsigned se obě strany převedou na int. Hodnota unsigned nad 2147483647 je pak záporná. Řešením je porovnání v int64: `(int64)u > i`. Ukázka: [T03_implicit_conversions.prg](#)

Přetypování

Zápis (typ)výraz převede hodnotu na zadaný typ, i na menší. Přetypování má vyšší prioritu než aritmetické operátory: `(int)d * 2` znamená `((int)d) * 2`.

- Celé číslo na menší celé číslo: horní bity se oříznou, např. `(char)300 = 44`, `(char)-1 = 255`.
- Desetinné číslo na celé: desetinná část se usekne, `(int)-3.9 = -3`; hodnoty mimo rozsah se omezí na minimum nebo maximum typu, NaN dá 0.
- Na bool: výsledek je true, je-li hodnota různá od nuly.
- Na string lze přetypovat jen char: `(string)c`. Čísla na text převádějí funkce `i2str`, `d2str` a další.

```
c = (char)i;           // nejnižší bajt
i = (int)d;            // useknutí desetinné části
l = (int64)u * 4;       // výpočet v 64 bitech
d = (double)a / b;      // dělení v double
```

Ukázka: [T04_explicit_casts.prg](#)

Výrazy a operátory

Výraz se skládá z proměnných, konstant, volání funkcí s návratovou hodnotou, operátorů a závorek. Operátory jsou v tabulce seřazeny od nejvyšší priority; operátory se stejnou prioritou se vyhodnocují zleva doprava.

Operátory	Význam	Typy operandů
()	závorky, volání funkce	
- ! ~ (typ)	unární minus, logická negace, bitová negace, přetypování	minus: čísla; !: bool; ~: celá čísla
* / %	násobení, dělení, zbytek po dělení	% jen celá čísla
+ -	sčítání a spojení řetězců, odčítání	čísla; + také string a char
<< >>	bitový posun vlevo a vpravo	celá čísla
< > <= >=	porovnání	čísla, znaky, řetězce
== !=	rovnost, nerovnost	všechny základní typy
&	bitový součin (AND)	celá čísla
^	bitový exkluzivní součet (XOR)	celá čísla
	bitový součet (OR)	celá čísla
&&	logický součin a součet (stejná priorita)	bool

- Operátory && a || mají stejnou prioritu a vyhodnocují se zleva: `a || b && c` znamená `(a || b) && c`. Při jejich kombinaci vždy použijte závorky.
- Celočíselné dělení useká směrem k nule (`-7 / 2 = -3`), zbytek má znaménko dělece (`-7 % 2 = -1`). Celočíselné dělení nulou vypíše hlášení `Division by zero` a výsledek je 0.
- Celočíselné typy při přetečení přecházejí přes rozsah (dvojkový doplněk), např. `M_INT32_MAX + 1 = -2147483648`.
- Výpočet probíhá v typu operandů: `char + char` se počítá v 8 bitech, `unsigned * unsigned` ve 32 bitech. Pro větší rozsah přetypujte jeden operand, např. `(int64)a * b`.
- U operátorů `~`, `<< a >>` se `char` rozšíří na `int`, takže `c << 8` neztratí horní bity.
- Porovnání řetězců probíhá podle abecedy a rozlišuje velká a malá písmena.

Ukázky: [T02_arithmetic.prg](#), [T06_bitwise_operators.prg](#), [T07_comparison_logic.prg](#), [T20_expressions_priority.prg](#)

Přiřazení

Zápis	Význam
<code>x = výraz;</code>	přiřazení
<code>x += výraz; -= *= /= %=</code>	složené přiřazení: <code>x = x + výraz</code> atd.
<code>x &= výraz; = ^= <= >=</code>	složené bitové přiřazení
<code>x++; x--; ++x; --x;</code>	zvýšení a snížení o 1

- Přiřazení je příkaz, nikoli výraz (nelze psát `a = b = 0;`).
- U složeného přiřazení se výsledek převede zpět na typ proměnné, proto funguje i `char c; c += 1;`.
- Složené přiřazení funguje i pro proměnné objektů (`msg.size += 2;`) a řetězce (`s += "abc"; s += c;`).
- `x++` a `x--` lze použít jako příkaz i v hlavičce cyklu `for`, `++x` a `--x` jen jako samostatný příkaz.

Ukázka: [T05_compound_assignment.prg](#)

Příkazy

Příkazy končí středníkem. Těla příkazů if, else, while a for se vždy uzavírají do složených závorek, i když obsahují jediný příkaz.

if, else if, else

```
if( temperature > 90 ) {
    prints("ALARM");
}
else if( temperature > 75 ) {
    prints("WARNING");
}
else {
    prints("OK");
}
```

while

```
i = 1;
while( i <= 10 ) {
    sum += i;
    i++;
}
```

for

V první a třetí části hlavičky for mohou být pouze přiřazení oddělená čárkou.

```
for( i = 0, j = 10; i < j; i++, j-- ) {
    logi(i);
    log_endl();
}
```

switch

Výraz switch může být číslo, znak nebo řetězec. Za case lze uvést více hodnot oddělených čárkou. Každá větev case i default musí končit příkazem break; (případně return ve funkci); default musí být poslední. Řetězce se porovnávají celé a rozlišují velká a malá písmena. break ukončí jen switch, nikoli okolní cyklus.

```
switch( command ) {
    case "start", "run":
        state = 1;
        break;
    case "stop":
        state = 0;
        break;
    default:
        prints("unknown command");
        break;
}
```

Jazyk nemá příkazy continue ani break pro cykly; cyklus se ukončí podmínkou. Ukázky: T08_control_flow.prg, T17_switch_details.prg

Ukončení skriptu

Funkce stop() ukončí běh skriptu (provede se sekce shutdown). Ukázka: T21_body_calls.prg

Funkce

Uživatelské funkce

```
function double average(double a, double b)
    return (a + b) / 2;
end

function divide(int x, int y, ref int quotient, ref int remainder)
    quotient = x / y;
    remainder = x % y;
end

function int next_id()
    static int counter;
    counter++;
    return counter;
end
```

- Definice: function [typ | void] jméno([ref] typ parametr, ...), tělo končí slovem end. Bez typu nebo s void jde o funkci bez návratové hodnoty.
- Funkce může být kdekoli v souboru a lze ji volat i před její definicí. Funkce s hodnotou se volá ve výrazu, funkce bez hodnoty jako samostatný příkaz.
- Parametry se předávají hodnotou a převedou se na typ parametru jako při přiřazení. Parametr ref je výstupní: po návratu se hodnota zkopíruje zpět, argumentem proto musí být proměnná stejného typu.
- Lokální proměnné se deklarují na začátku těla a při každém volání mají hodnotu 0. Proměnná static si hodnotu ponechá mezi voláními. Lokální jména zakrývají globální proměnné stejného jména.
- return výraz; vrátí hodnotu, return; ukončí funkci bez hodnoty.
- Funkce mohou volat jiné funkce do hloubky 10; rekurze (ani nepřímá) není dovolena a interpret při ní skript zastaví.
- Parametry jsou základních typů. Objekty jsou globální a funkce je používá přímo jménem.
- Výsledek funkce nebo metody s návratovou hodnotou se musí použít (přiřadit nebo použít ve výrazu).

Ukázky: T09_user_functions.prg, T19_edge_cases.prg

Sekce func_0 až func_9

Jednoduché funkce bez parametrů a návratové hodnoty pracující s globálními proměnnými. Volají se příkazem func(N);, kde N může být i proměnná. Vnoření je omezeno na 10 úrovní.

```
body:
    func(0);
end

func_0:
    counter++;
end
```

Ukázka: T10_func_sections.prg

Vestavěné funkce

U každé funkce je uvedena deklarace v syntaxi jazyka C: typ výsledku (void = bez výsledku), jméno a typy parametrů. Funkce s výsledkem se používá ve výrazu, funkce bez výsledku jako samostatný příkaz.

Konverzní funkce

Převody mezi typy a mezi čísly a řetězci, práce s bity a bajty. Pro převod mezi číselnými typy lze použít také přetypování (typ)výraz.

string b2str(bool value)

Podle hodnoty parametru value vrací řetězec true nebo false.

string c2str(char value)

Převod znaku na řetězec.

string i2str(int value)

Převod celé číslo na řetězec.

string i2str_radix(int value, int radix)

Převod celé číslo na řetězec v zadané číselné soustavě (např. 2, 16).

string u2str(unsigned value)

Převod bezznaménkové celé číslo na řetězec.

string f2str(float value)

Převod desetinné číslo na řetězec.

string d2str(double value)

Převod desetinné číslo na řetězec.

int str2i(string value)

Převod celé číslo zapsané v řetězci na číselnou hodnotu.

unsigned str2u(string value)

Převod celé číslo zapsané v řetězci na číselnou hodnotu.

char str2c(string value)

Převod celé číslo zapsané v řetězci na číselnou hodnotu typu char.

bool str2b(string value)

Převod logickou hodnotu zapsanou v řetězci. Rozhoduje první znak: 1, y, Y, a, A znamená true, cokoli jiného false.

float str2f(string value)

Převod desetinné číslo zapsané v řetězci na číselnou hodnotu.

double str2d(string value)

Převod desetinné číslo zapsané v řetězci na číselnou hodnotu.

unsigned from_hex(string value)

Převod hexadecimální číslo zapsané v řetězci (např. "1A2B") na číselnou hodnotu.

int truncf(float value)

Vrací celou část desetinného čísla.

int truncd(double value)

Vrací celou část desetinného čísla.

int roundf(float value)

Zaokrouhlí desetinné číslo.

int roundd(double value)

Zaokrouhlí desetinné číslo.

char i2c(int value)

Převede int na char (nejnižší bajt).

char u2c(unsigned value)

Převede unsigned na char (nejnižší bajt).

unsigned i2u(int value)

Převede int na unsigned.

char i642c(int64 value)

Převede int64 na char (nejnižší bajt).

char u642c(unsigned64 value)

Převede unsigned64 na char (nejnižší bajt).

int i642i(int64 value)

Převede int64 na int (nejnižších 32 bitů).

unsigned u642u(unsigned64 value)

Převede unsigned64 na unsigned (nejnižších 32 bitů).

char bcd2dec(char value)

Převede bajt v kódu BCD na hodnotu (např. 0x25 na 25).

char dec2bcd(char value)

Převede hodnotu 0 až 99 na bajt v kódu BCD.

char get_bfu(unsigned value, int byte_index)

Vrací zadaný bajt (index 0 až 3) ze slova typu unsigned.

char get_bfi(int value, int byte_index)

Vrací zadaný bajt (index 0 až 3) ze slova typu int.

char bitsetc(char value, int index)

Vrací hodnotu s bitem index nastaveným na 1. Varianty: bitseti (int), bitsetu (unsigned), bitseti64 (int64), bitsetu64 (unsigned64).

int bitseti(int value, int index)

Viz bitsetc.

unsigned bitsetu(unsigned value, int index)

Viz bitsetc.

int64 bitseti64(int64 value, int index)

Viz bitsetc.

unsigned64 bitsetu64(unsigned64 value, int index)

Viz bitsetc.

char bitclrc(char value, int index)

Vrací hodnotu s bitem index nastaveným na 0. Varianty: bitclri, bitclru, bitclri64, bitclru64.

int bitclri(int value, int index)

Viz bitclrc.

unsigned bitclru(unsigned value, int index)

Viz bitclrc.

int64 bitclri64(int64 value, int index)

Viz bitclrc.

unsigned64 bitclru64(unsigned64 value, int index)

Viz bitclrc.

bool bitestc(char value, int index)

Vrací true, má-li bit index hodnotu 1. Varianty: bitesti, bitestu, bitesti64, bitestu64.

bool bitesti(int value, int index)

Viz bitestc.

bool bitestu(unsigned value, int index)

Viz bitestc.

bool bitesti64(int64 value, int index)

Viz bitestc.

bool bitestu64(unsigned64 value, int index)

Viz bitestc.

Ukázky: T12_string_conversions.prg, T14_bit_functions_bcd.prg, T18_more_functions.prg

Funkce pro práci s typem string

Pozice znaků v řetězci se číslují od 0. Funkce, které řetězec mění (str_insert, str_erase, str_replace, str_remove...), mění přímo proměnnou předanou jako první parametr.

int str_length(string str)

Vrací délku textu v řetězci.

int str_find(string str, string substr)

Vrací pozici podřetězce substr v řetězci str, není-li nalezen, vrací -1.

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

Jako str_find, hledání začíná na pozici startpos.

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

Vrací pozici prvního výskytu některého ze znaků charset od pozice startpos, jinak -1.

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

Vrací pozici prvního znaku od startpos, který není v charset, jinak -1.

int str_replace(string str, int startpos, int count, string source)

Nahradí v řetězci str count znaků od pozice startpos řetězcem source. Vrací 1, pokud nahrazení proběhlo, nebo 0, je-li startpos mimo řetězec.

char str_get(string str, int pos)

Vrací znak na pozici pos, je-li pozice mimo řetězec, vrací 0.

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

Přepíše znak na pozici pos řetězce str; pozice musí být uvnitř řetězce.

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

Vymaže z řetězce length znaků od pozice startpos.

int str_row(string text, string row, int pos)

Do row zkopíruje řádek textu začínající na pozici pos (do znaku konce řádku). Vrací pozici začátku dalšího řádku, nebo -1, není-li další řádek. Vhodné pro procházení víceřádkového textu.

int str_item(string text, string item, int pos, char separator)

Jako str_row, položky jsou odděleny znakem separator (např. čárkou).

void str_remove_nonprintable(string str)

Odstraní z řetězce znaky mimo rozsah ASCII 0 až 127.

void str_remove_diacritics(string str)

Nahradí znaky s diakritikou znaky bez diakritiky.

Ukázky: T11_strings.prg, T12_string_conversions.prg

Zápis na výstup

Funkce print zapíše hodnotu ihned do okna OUTPUT. Funkce log zapisují do interního textového bufferu (max. 512 znaků), jehož obsah se do okna OUTPUT vypíše funkcí log_endl().

void prints(string value)

Vypíše řetězec.

void printb(bool value)

Vypíše true nebo false.

void printc(char value, bool type)

Vypíše hodnotu jako znak (je-li type true) nebo jako číslo 0 až 255.

void printi(int value)

Vypíše hodnotu typu int.

void printu(unsigned value)

Vypíše hodnotu typu unsigned.

void printi64(int64 value)

Vypíše hodnotu typu int64.

void printu64(unsigned64 value)

Vypíše hodnotu typu unsigned64.

void printf(float value)

Vypíše desetinné číslo.

void printd(double value)

Vypíše desetinné číslo.

void print_endl()

Vypíše prázdný řádek.

void logs(string value)

Přidá řetězec do bufferu.

void logb(bool value)

Přidá true nebo false do bufferu.

void logc(char value, bool type)

Přidá hodnotu jako znak (je-li type true) nebo jako číslo 0 až 255.

void logc_hex(char value)

Přidá bajt hexadecimálně, vždy dvojmištně (00 až FF).

void logc_bin(char value)

Přidá bajt binárně jako znak b a 8 bitů, např. 15 jako b00001111.

void logi(int value)

Přidá hodnotu typu int.

void logu(unsigned value)

Přidá hodnotu typu unsigned.

void logu_hex(unsigned value)

Přidá hodnotu typu unsigned hexadecimálně.

void logu_bin(unsigned value)

Přidá hodnotu typu unsigned binárně (b a 32 bitů).

void logi64(int64 value)

Přidá hodnotu typu int64.

void logu64(unsigned64 value)

Přidá hodnotu typu unsigned64.

void logu64_hex(unsigned64 value)

Přidá hodnotu typu unsigned64 hexadecimálně.

void logu64_bin(unsigned64 value)

Přidá hodnotu typu unsigned64 binárně (b a 64 bitů).

void logf(float value)

Přidá desetinné číslo.

void logd(double value)

Přidá desetinné číslo.

void log_time()

Přidá aktuální čas ve tvaru <h:m:s.ms>. Vhodné na začátek záznamu o události.

void log_endl()

Vypíše obsah bufferu do okna OUTPUT a buffer vymaže.

void log_endl_color(char color)

Jako log_endl, text se zobrazí barvou 0 černá, 1 červená (COLOR_RED), 2 zelená (COLOR_GREEN), 3 modrá (COLOR_BLUE).

void log_clr()

Vymaže obsah bufferu.

int log_length()

Vrací aktuální délku textu v bufferu.

int log_max()

Vrací maximální délku textu v bufferu.

void clear_output()

Vymaže okno OUTPUT.

Ukázky: T18_more_functions.prg

Aritmetické a matematické operace

Goniometrické funkce pracují v radiánech. Bitové funkce (shl, shr, and, or, xor) dávají stejný výsledek jako operátory << >> & | ^.

char shlc(char value, int count)

Vrací hodnotu posunutou o count bitů vlevo. Varianty: shli (int), shlu (unsigned), shlu64.

int shli(int value, int count)

Viz shlc.

unsigned shlu(unsigned value, int count)

Viz shlc.

unsigned64 shlu64(unsigned64 value, int count)

Viz shlc.

char shrc(char value, int count)

Vrací hodnotu posunutou o count bitů vpravo. Varianty: shri, shru, shru64.

int shri(int value, int count)

Viz shrc.

unsigned shru(unsigned value, int count)

Viz shrc.

unsigned64 shru64(unsigned64 value, int count)

Viz shrc.

char andc(char a, char b)

Vrací bitový součin (AND). Varianty: andi, andu, andu64.

int andi(int a, int b)

Viz andc.

unsigned andu(unsigned a, unsigned b)

Viz andc.

unsigned64 andu64(unsigned64 a, unsigned64 b)

Viz andc.

char orc(char a, char b)

Vrací bitový součet (OR). Varianty: ori, oru, oru64.

int ori(int a, int b)

Viz orc.

unsigned oru(unsigned a, unsigned b)

Viz orc.

unsigned64 oru64(unsigned64 a, unsigned64 b)

Viz orc.

char xorc(char a, char b)

Vrací bitový exkluzivní součet (XOR). Varianty: xori, xoru, xoru64.

int xori(int a, int b)

Viz xorc.

unsigned xoru(unsigned a, unsigned b)

Viz xorc.

unsigned64 xoru64(unsigned64 a, unsigned64 b)

Viz xorc.

double sin(double a)

Sinus, parametr v radiánech.

double cos(double a)

Kosinus, parametr v radiánech.

double tan(double a)

Tangens, parametr v radiánech.

double sinh(double a)

Hyperbolický sinus.

double cosh(double a)

Hyperbolický kosinus.

double tanh(double a)

Hyperbolický tangens.

double asin(double a)

Arkus sinus (výsledek v radiánech).

double acos(double a)

Arkus kosinus (výsledek v radiánech).

double atan(double a)

Arkus tangens (výsledek v radiánech).

double log(double a)

Přirozený logaritmus.

double log10(double a)

Dekadický logaritmus.

double exp(double a)Exponenciální funkce e^a .**double sqrt(double a)**

Druhá odmocnina.

double pow(double a, double b)Mocnina a na b .**double poly(double x, double a, double b, double c)**Hodnota polynomu $a*x*x + b*x + c$.**double linear_interpolation(double x0, double y0, double x1, double y1, double x)**Lineární interpolace hodnoty v bodě x mezi body $[x0, y0]$ a $[x1, y1]$.**double deg2rad(double a)**

Převod stupňů na radiány.

double rad2deg(double a)

Převod radiánů na stupně.

char minc(char x1, char x2)

Vrací menší ze dvou hodnot. Varianty: mini, minu, mini64, minu64, minf, mind.

int mini(int x1, int x2)

Viz minc.

unsigned minu(unsigned x1, unsigned x2)

Viz minc.

int64 mini64(int64 x1, int64 x2)

Viz minc.

unsigned64 minu64(unsigned64 x1, unsigned64 x2)

Viz minc.

float minf(float x1, float x2)

Viz minc.

double mind(double x1, double x2)

Viz minc.

char maxc(char x1, char x2)

Vrací větší ze dvou hodnot. Varianty: maxi, maxu, maxi64, maxu64, maxf, maxd.

int maxi(int x1, int x2)

Viz maxc.

unsigned maxu(unsigned x1, unsigned x2)

Viz maxc.

int64 maxi64(int64 x1, int64 x2)

Viz maxc.

unsigned64 maxu64(unsigned64 x1, unsigned64 x2)

Viz maxc.

float maxf(float x1, float x2)

Viz maxc.

double maxd(double x1, double x2)

Viz maxc.

int minusi(int value)

Vrací hodnotu s opačným znaménkem.

double minusd(double value)

Vrací hodnotu s opačným znaménkem.

int random(int min, int max)

Vrací náhodné číslo v rozsahu min až max-1.

Ukázky: T13_math_functions.prg, T14_bit_functions_bcd.prg

CAN interface

unsigned x2can_tec()

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

unsigned x2can_rec()

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

unsigned x2can_txbl()

Vrací počet CAN zpráv čekajících na odeslání.

unsigned x2can_rxbl()

Vrací počet přijatých CAN zpráv čekajících na zpracování.

void x2can_reset(int value)

Nastaví komunikační rychlost CAN: 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.

bool lin_config(int port, int mode, unsigned speed, int version)

Nastaví LIN port 1 nebo 2 převodníku USB2LIN: režim LIN_MASTER, LIN_SLAVE nebo LIN_SCAN, rychlost v Bd (např. 19200) a verzi LIN_VER_13 nebo LIN_VER_20. Vrací true při úspěchu; při chybném parametru nebo jiném převodníku vypíše hlášení a vrací false.

Datum a čas

Funkce vracejí místní čas, varianty _utc koordinovaný světový čas (UTC). Pro měření časových intervalů použijte objekt obj_timepoint.

int time_h()

Aktuální hodina. Varianta time_utc_h.

int time_m()

Aktuální minuta. Varianta time_utc_m.

int time_s()

Aktuální sekunda. Varianta time_utc_s.

int time_ms()

Aktuální milisekundy. Varianta time_utc_ms.

int time_utc_h()

Viz time_h.

int time_utc_m()

Viz time_m.

int time_utc_s()

Viz time_s.

int time_utc_ms()

Viz time_ms.

int date_y()

Aktuální rok. Varianta date_utc_y.

int date_m()

Aktuální měsíc. Varianta date_utc_m.

int date_d()

Aktuální den. Varianta date_utc_d.

int date_utc_y()

Viz date_y.

int date_utc_m()

Viz date_m.

int date_utc_d()

Viz date_d.

Ukázky: T16_time_timepoint.prg

Zobrazení a řízení proměnných

Okno skriptu umožňuje zobrazit 4 hodnoty typu double a 4 hodnoty typu bool, řídit progressbar (0 až 10000), číst dva posuvníky (0 až 10000), 4 tlačítka a skupinu 4 přepínačů. Funkce mají předponu ui (user interaction). Lze číst i joystick a stav kláves.

bool ui_get_b0()

Vrací stav tlačítka 0. Varianty ui_get_b1, ui_get_b2, ui_get_b3.

bool ui_get_b1()

Viz ui_get_b0.

bool ui_get_b2()

Viz ui_get_b0.

bool ui_get_b3()

Viz ui_get_b0.

void ui_set_b0(bool state)

Nastaví stav tlačítka 0 (např. pro signalizaci). Varianty ui_set_b1 až ui_set_b3.

void ui_set_b1(bool state)

Viz ui_set_b0.

void ui_set_b2(bool state)

Viz ui_set_b0.

void ui_set_b3(bool state)

Viz ui_set_b0.

int ui_get_r()

Vrací vybraný přepínač ze skupiny (0 až 3).

void ui_set_r(int state)

Vybere přepínač ve skupině (0 až 3).

void ui_set_name_r(int radio_button, string name)

Nastaví popisek zadaného přepínače.

void ui_set_p(unsigned state)

Nastaví progressbar (0 až 10000).

void ui_set_d0(double state)

Zobrazí hodnotu v okně 0. Varianty ui_set_d1 až ui_set_d3.

void ui_set_d1(double state)

Viz ui_set_d0.

void ui_set_d2(double state)

Viz ui_set_d0.

void ui_set_d3(double state)

Viz ui_set_d0.

unsigned ui_get_val0()

Vrací hodnotu posuvníku 0 (0 až 10000).

unsigned ui_get_val1()

Vrací hodnotu posuvníku 1 (0 až 10000).

void ui_set_val0(unsigned value)

Nastaví hodnotu posuvníku 0.

void ui_set_val1(unsigned value)

Nastaví hodnotu posuvníku 1.

void ui_set_timer_script_period(unsigned period)

Nastaví periodu volání skriptu časovačem v ms. Vhodné volat v sekci init, ovládací prvek se nastaví automaticky.

unsigned ui_get_timer_script_period()

Vrací periodu volání skriptu časovačem.

int ui_joy_x(int joystick_id)

Vrací polohu osy X joysticku (0 první, 1 druhý; konstanty JOYSTICK1, JOYSTICK2). Varianty pro osy y, z, r, u, v. Rozsah hodnot závisí na joysticku.

int ui_joy_y(int joystick_id)

Viz ui_joy_x.

int ui_joy_z(int joystick_id)

Viz ui_joy_x.

int ui_joy_r(int joystick_id)

Viz ui_joy_x.

```
int ui_joy_u(int joystick_id)
```

Viz ui_joy_x.

```
int ui_joy_v(int joystick_id)
```

Viz ui_joy_x.

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

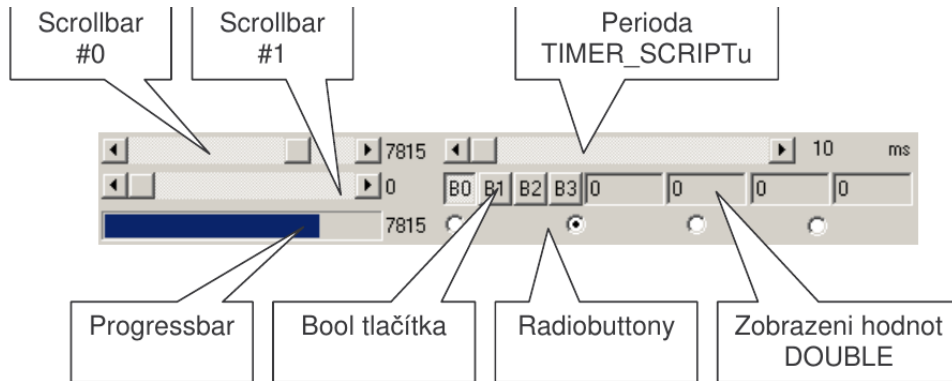
Vrací stav zadaného tlačítka joysticku.

```
int ui_joy_get_num()
```

Vrací počet joysticků podporovaných systémem.

```
bool ui_key_state(int key)
```

Vrací true, je-li stisknuta klávesa zadaná virtuálním kódem (konstanty VK_...).



Obr. 3: Ovládací prvky okna skriptu

Ukázka: D01_ui_controls.prg

Globální proměnné

PP2CAN může provozovat současně 4 skripty. Ke spolupráci mohou využít sdílená pole po 128 prvcích typu bool, int, unsigned, double a string (index 0 až 127).

```
bool global_bool_get(unsigned index)
```

Čte globální proměnnou typu bool.

```
int global_int_get(unsigned index)
```

Čte globální proměnnou typu int.

```
unsigned global_unsigned_get(unsigned index)
```

Čte globální proměnnou typu unsigned.

```
double global_double_get(unsigned index)
```

Čte globální proměnnou typu double.

```
string global_string_get(unsigned index)
```

Čte globální proměnnou typu string.

```
void global_bool_set(unsigned index, bool value)
```

Zapíše globální proměnnou typu bool.

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

Zapíše globální proměnnou typu int.

```
void global_unsigned_set(unsigned index, unsigned value)
```

Zapíše globální proměnnou typu unsigned.

```
void global_double_set(unsigned index, double value)
```

Zapíše globální proměnnou typu double.

void global_string_set(unsigned index, string value)

Zapíše globální proměnnou typu string.

Ukázky: T15_global_variables.prg, P01a_global_producer.prg, P01b_global_consumer.prg

Další funkce**void stop()**

Ukončí provádění skriptu.

void func(int index)

Spustí sekci func_0 až func_9 se zadaným číslem.

void wait(unsigned value)

Pozastaví skript na value milisekund. Používejte jen v nutných případech, při velkém množství dat může dojít k zahlcení.

unsigned get_run_counter()

Vrací počet provedení sekce body.

void msg_box(string value)

Zobrazí MessageBox se zadaným textem.

void msg_beep()

Krátké zvukové upozornění.

string file_dialog_load(string fileext)

Otevře dialog pro výběr souboru k načtení s příponou fileext a vrací cestu (při zrušení mezeru).

string file_dialog_save(string fileext)

Otevře dialog pro výběr souboru k uložení a vrací cestu.

void command(string cmd, string output)

Spustí příkaz operačního systému cmd a jeho textový výstup uloží do output.

bool is_can_function_call()

Vrací true, byl-li skript spuštěn přijatou CAN zprávou (CAN_FUNCTION_AND_TIMER).

bool is_timer_call()

Vrací true, byl-li skript spuštěn časovačem.

bool is_lin_function_call()

Vrací true, byl-li skript spuštěn přijatou LIN zprávou (LIN_FUNCTION_AND_TIMER).

Ukázky: T21_body_calls.prg

Vestavěné konstanty

Konstanty lze ve skriptu použít místo hodnot. Konstanta má přednost před proměnnou stejného jména.

Znaky a řetězce

Název	Hodnota
ENDL	(unsigned char)"\n"
TAB	(unsigned char)"\t"
ENDLS	"\n"
TABS	"\t"

Barvy pro log_endl_color

Název	Hodnota
COLOR_RED	(unsigned char)0x01
COLOR_GREEN	(unsigned char)0x02
COLOR_BLUE	(unsigned char)0x03

Indexy joysticků

Název	Hodnota
JOYSTICK1	(int)0
JOYSTICK2	(int)1

Matematické konstanty

Název	Hodnota
M_PI	(double)3.14159265358979323846
M_PI_2	(double)1.57079632679489661923
M_PI_4	(double)0.785398163397448309616
M_E	(double)2.71828182845904523536
M_SQRT2	(double)1.41421356237309504880
M_SQRT1_2	(double)0.707106781186547524401

Rozsahy typů

Název	Hodnota
M_INT16_MIN	(int)-32768
M_INT16_MAX	(int)32767
M_INT32_MIN	(int)-2147483648
M_INT32_MAX	(int)2147483647
M_INT64_MIN	LLONG_MIN
M_INT64_MAX	LLONG_MAX
M_UINT16_MIN	(unsigned int)0
M_UINT16_MAX	(unsigned int)65535

Název	Hodnota
M_UINT32_MIN	(unsigned int)0
M_UINT32_MAX	(unsigned int)4294967295
M_UINT64_MIN	0
M_UINT64_MAX	ULLONG_MAX
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

LIN

Název	Hodnota
LIN_MASTER	(int)0
LIN_SLAVE	(int)1
LIN_SCAN	(int)2
LIN_VER_13	(int)13
LIN_VER_20	(int)20
LIN_PERIOD_OFF	(unsigned int)0
LIN_ONE_SHOT	(unsigned int)0xFFFF

Adresování obj_uds

Název	Hodnota
ISOTP_NORMAL	(int)0
ISOTP_EXTENDED	(int)1
ISOTP_MIXED	(int)2

Kódy chyb obj_uds a obj_j1939

Název	Hodnota
TP_OK	(int)0
TP_ERR_SEQUENCE	(int)1
TP_ERR_UNEXPECTED	(int)2
TP_ERR_SIZE	(int)3
TP_ERR_ABORT	(int)4
TP_ERR_CONFIG	(int)5
TP_ERR_FRAME	(int)6

Volací konvence pro obj_dllfnc

Název	Hodnota
ABI_STDCALL	(unsigned int)0
ABI_THISCALL	(unsigned int)1

Název	Hodnota
ABI_FASTCALL	(unsigned int)2
ABI_MS_CDECL	(unsigned int)3
ABI_PASCAL	(unsigned int)4
ABI_REGISTER	(unsigned int)5

Typy parametrů pro obj_dllfnc

Název	Hodnota
PARAM_VOID	(unsigned int)0
PARAM_UINT8	(unsigned int)1
PARAM_SINT8	(unsigned int)2
PARAM_UINT16	(unsigned int)3
PARAM_SINT16	(unsigned int)4
PARAM_UINT32	(unsigned int)5
PARAM_SINT32	(unsigned int)6
PARAM_UINT64	(unsigned int)7
PARAM_SINT64	(unsigned int)8
PARAM_FLOAT	(unsigned int)9
PARAM_DOUBLE	(unsigned int)10
PARAM_POINTER	(unsigned int)11

Virtuální kódy kláves pro ui_key_state

Název	Hodnota
VK_LBUTTON	(int)0x01
VK_RBUTTON	(int)0x02
VK_CANCEL	(int)0x03
VK_MBUTTON	(int)0x04
VK_BACK	(int)0x08
VK_TAB	(int)0x09
VK_CLEAR	(int)0x0C
VK_RETURN	(int)0x0D
VK_SHIFT	(int)0x10
VK_CONTROL	(int)0x11
VK_MENU	(int)0x12
VK_PAUSE	(int)0x13
VK_CAPITAL	(int)0x14
VK_ESCAPE	(int)0x1B
VK_CONVERT	(int)0x1C
VK_NONCONVERT	(int)0x1D
VK_ACCEPT	(int)0x1E
VK_MODECHANGE	(int)0x1F

Název	Hodnota
VK_SPACE	(int)0x20
VK_PRIOR	(int)0x21
VK_NEXT	(int)0x22
VK_END	(int)0x23
VK_HOME	(int)0x24
VK_LEFT	(int)0x25
VK_UP	(int)0x26
VK_RIGHT	(int)0x27
VK_DOWN	(int)0x28
VK_SELECT	(int)0x29
VK_PRINT	(int)0x2A
VK_EXECUTE	(int)0x2B
VK_SNAPSHOT	(int)0x2C
VK_INSERT	(int)0x2D
VK_DELETE	(int)0x2E
VK_HELP	(int)0x2F
VK_0	(int)0x30
VK_1	(int)0x31
VK_2	(int)0x32
VK_3	(int)0x33
VK_4	(int)0x34
VK_5	(int)0x35
VK_6	(int)0x36
VK_7	(int)0x37
VK_8	(int)0x38
VK_9	(int)0x39
VK_A	(int)0x41
VK_B	(int)0x42
VK_C	(int)0x43
VK_D	(int)0x44
VK_E	(int)0x45
VK_F	(int)0x46
VK_G	(int)0x47
VK_H	(int)0x48
VK_I	(int)0x49
VK_J	(int)0x4A
VK_K	(int)0x4B
VK_L	(int)0x4C
VK_M	(int)0x4D
VK_N	(int)0x4E
VK_O	(int)0x4F

Název	Hodnota
VK_P	(int)0x50
VK_Q	(int)0x51
VK_R	(int)0x52
VK_S	(int)0x53
VK_T	(int)0x54
VK_U	(int)0x55
VK_V	(int)0x56
VK_W	(int)0x57
VK_X	(int)0x58
VK_Y	(int)0x59
VK_Z	(int)0x5A
VK_LWIN	(int)0x5B
VK_RWIN	(int)0x5C
VK_APPS	(int)0x5D
VK_SLEEP	(int)0x5F
VK_NUMPAD0	(int)0x60
VK_NUMPAD1	(int)0x61
VK_NUMPAD2	(int)0x62
VK_NUMPAD3	(int)0x63
VK_NUMPAD4	(int)0x64
VK_NUMPAD5	(int)0x65
VK_NUMPAD6	(int)0x66
VK_NUMPAD7	(int)0x67
VK_NUMPAD8	(int)0x68
VK_NUMPAD9	(int)0x69
VK_MULTIPLY	(int)0x6A
VK_ADD	(int)0x6B
VK_SEPARATOR	(int)0x6C
VK_SUBTRACT	(int)0x6D
VK_DECIMAL	(int)0x6E
VK_DIVIDE	(int)0x6F
VK_F1	(int)0x70
VK_F2	(int)0x71
VK_F3	(int)0x72
VK_F4	(int)0x73
VK_F5	(int)0x74
VK_F6	(int)0x75
VK_F7	(int)0x76
VK_F8	(int)0x77
VK_F9	(int)0x78
VK_F10	(int)0x79

Název	Hodnota
VK_F11	(int)0x7A
VK_F12	(int)0x7B
VK_F13	(int)0x7C
VK_F14	(int)0x7D
VK_F15	(int)0x7E
VK_F16	(int)0x7F
VK_F17	(int)0x80
VK_F18	(int)0x81
VK_F19	(int)0x82
VK_F20	(int)0x83
VK_F21	(int)0x84
VK_F22	(int)0x85
VK_F23	(int)0x86
VK_F24	(int)0x87
VK_NUMLOCK	(int)0x90
VK_SCROLL	(int)0x91
VK_LSHIFT	(int)0xA0
VK_RSHIFT	(int)0xA1
VK_LCONTROL	(int)0xA2
VK_RCONTROL	(int)0xA3
VK_LMENU	(int)0xA4
VK_RMENU	(int)0xA5
VK_BROWSER_BACK	(int)0xA6
VK_BROWSER_FORWARD	(int)0xA7
VK_BROWSER_REFRESH	(int)0xA8
VK_BROWSER_STOP	(int)0xA9
VK_BROWSER_SEARCH	(int)0xAA
VK_BROWSER_FAVORITES	(int)0xAB
VK_BROWSER_HOME	(int)0xAC
VK_VOLUME_MUTE	(int)0xAD
VK_VOLUME_DOWN	(int)0xAE
VK_VOLUME_UP	(int)0xAF
VK_MEDIA_NEXT_TRACK	(int)0xB0
VK_MEDIA_PREV_TRACK	(int)0xB1
VK_MEDIA_STOP	(int)0xB2
VK_MEDIA_PLAY_PAUSE	(int)0xB3
VK_LAUNCH_MAIL	(int)0xB4
VK_LAUNCH_MEDIA_SELECT	(int)0xB5
VK_LAUNCH_APP1	(int)0xB6
VK_LAUNCH_APP2	(int)0xB7

Objekty

Objekty sdružují data a operace nad nimi (metody). Typy objektů jsou pevně dané (názvy začínají obj_) a deklarují se v sekci objects. K datovým složkám a metodám se přistupuje tečkou.

```
objects:
    obj_can_msg msg;
end

init:
    msg.id1 = 0x100;           // datova slozka
    msg.size = 8;
    msg.send();               // metoda bez navratove hodnoty
    n = msg.get_uint16(0);     // metoda s navratovou hodnotou
end
```

Metody next_frame a receive objektů obj_uds a obj_j1939 přebírají jako parametr objekt obj_can_msg.

Objekt	Účel
obj_can_msg	CAN a CAN FD zpráva: sestavení, odeslání, příjem, signály
obj_lin	LIN zpráva a položka LIN tabulky (USB2LIN)
obj_uds, obj_j1939	přenos dlouhých dat: ISO-TP (UDS), J1939-21/22
obj_vector_int, obj_vector_double	pole čísel s algoritmy (řazení, statistika, CRC, Base64)
obj_deque_int, obj_deque_double	obousměrná fronta (FIFO, zásobník)
obj_memory	blok paměti (např. pro funkce v DLL)
obj_timepoint	měření časových intervalů
obj_rs232	sériový port
obj_socket_server, obj_socket_client	TCP a UDP komunikace
obj_csvfile, obj_cfgfile, obj_logfile	soubory CSV, konfigurační (ini) a textové záznamy
obj_datagrid, obj_dataset, obj_datagraph, obj_datagauge	okna pro zobrazení a zadávání hodnot, graf, budík
obj_pid	PID regulátor
obj_dllfnc	volání funkce z knihovny DLL

Přiřazení objektů

Přiřazení a = b; mezi objekty kopíruje nebo převádí data. Podporované kombinace:

Cíl = zdroj	Výsledek
obj_can_msg = obj_can_msg	kopie celé zprávy (ID, příznaky, port, data)
obj_vector_int = obj_can_msg	datové bajty zprávy (obsah vektoru se nahradí)
obj_deque_int = obj_can_msg	datové bajty se přidají na konec fronty
obj_can_msg = obj_vector_int	bajty z vektoru (max. 64), size podle počtu; nad 8 bajtů fd = true
vektor = vektor / deque	kopie prvků; do vektoru se obsah nahradí
deque = vektor / deque	prvky se přidají na konec fronty
int <-> double kontejnery	převod hodnot; double na int useknutím
obj_memory = obj_vector_int / obj_can_msg	alokuje blok a zkopíruje bajty
obj_vector_int / obj_deque_int = obj_rs232, socket	přijátá data
obj_rs232, socket = obj_vector_int / obj_deque_int	odeslání dat

Cíl = zdroj	Výsledek
obj_datagraph = kontejner	hodnoty do grafu (čára 0)
obj_csvfile = kontejner	hodnoty do aktivního řádku tabulky
obj_can_msg = obj_uds / obj_j1939	další rámec k odeslání
obj_uds / obj_j1939 = obj_can_msg	předání přijatého rámce
obj_uds / obj_j1939 <-> obj_vector_int	data přenosu

Přiřazení objektu do sebe sama nic nemění. Po přiřazení ukazuje vnitřní iterátor cílového kontejneru na první prvek.

Ukázky: C06_object_assignment.prg, C07_object_assignment_matrix.prg

obj_can_msg

Práce s CAN zprávou: sestavení a odeslání zprávy, načtení přijaté zprávy (skript CAN_FUNCTION). Datová část má 64 bajtů (CAN FD), u klasického CAN se používá prvních 8. Metody get_/set_ pro vícebajtové hodnoty používají index prvního bajtu; index musí odpovídat délce dat (např. set_uint32 nejvýše 60).

Datové složky

unsigned id1

Identifikátor zprávy (11bitová část).

unsigned id2

Druhá část identifikátoru (18bitová část) u rozšířeného identifikátoru.

int size

Počet datových bajtů (DLC), 0 až 8, u CAN FD až 64.

bool stext

false = standardní (11bitový) identifikátor, true = rozšířený (29bitový).

bool rtr

Příznak RTR.

int port

Číslo portu (CAN kanálu) pro odeslání, u přijaté zprávy port příjmu. Výchozí hodnota 1.

bool fd

Zpráva CAN FD.

bool brs

CAN FD: přepnutí přenosové rychlosti (bit rate switch).

Metody

void send()

Odešle zprávu na CAN sběrnici.

void load()

Načte zprávu, která vyvolala spuštění skriptu typu CAN_FUNCTION (CAN_FUNCTION_AND_TIMER). U volání časovačem nemá význam.

unsigned get_ID29()

Vrací identifikátor ve 29bitovém tvaru.

void set_ID29(unsigned id)

Nastaví id1 a id2 z identifikátoru ve 29bitovém tvaru.

void set_ID29_hex(string id)

Jako set_ID29, identifikátor je zadán hexadecimálním řetězcem.

char data(int i)

Vrací i-tý datový bajt.

void set_data(int i, char byte)

Nastaví i-tý datový bajt.

void data_from_hex(string hex)

Nastaví datové bajty z hexadecimálního řetězce (např. "0102A0FF") a podle počtu bajtů nastaví size.

void set_all(char b0, char b1, char b2, char b3, char b4, char b5, char b6, char b7)

Nastaví prvních 8 datových bajtů.

void set_alli(int b0, int b1, int b2, int b3, int b4, int b5, int b6, int b7)

Jako set_all, hodnoty se zadávají jako int (použije se nejnižší bajt).

void fill(char value)

Nastaví všech 64 datových bajtů na zadanou hodnotu.

void fill_data(char value)

Stejně jako fill.

void set_bool(int i, bool value)

Zapíše hodnotu bool do i-tého bajtu (0 nebo 1).

bool get_bool(int i)

Čte hodnotu bool z i-tého bajtu.

void set_bit(int i, bool value)

Nastaví bit datové části (index bitu 0 až 63).

bool get_bit(int i)

Vrací bit datové části (index bitu 0 až 63).

void set_int16(int i, int value)

Zapíše 16bitovou hodnotu se znaménkem od i-tého bajtu. Varianta _be pracuje s opačným pořadím bajtů (big endian).

int get_int16(int i)

Čte 16bitovou hodnotu se znaménkem. Varianta _be pracuje s opačným pořadím bajtů (big endian).

void set_int16_be(int i, int value)

Viz set_int16.

int get_int16_be(int i)

Viz get_int16.

void set_uint16(int i, unsigned value)

Zapíše 16bitovou hodnotu bez znaménka. Varianta _be pracuje s opačným pořadím bajtů (big endian).

unsigned get_uint16(int i)

Čte 16bitovou hodnotu bez znaménka. Varianta _be pracuje s opačným pořadím bajtů (big endian).

void set_uint16_be(int i, unsigned value)

Viz set_uint16.

unsigned get_uint16_be(int i)

Viz get_uint16.

void set_int32(int i, int value)

Zapíše 32bitovou hodnotu se znaménkem. Varianta _be pracuje s opačným pořadím bajtů (big endian).

int get_int32(int i)

Čte 32bitovou hodnotu se znaménkem. Varianta `_be` pracuje s opačným pořadím bajtů (big endian).

void set_int32_be(int i, int value)

Viz `set_int32`.

int get_int32_be(int i)

Viz `get_int32`.

void set_uint32(int i, unsigned value)

Zapíše 32bitovou hodnotu bez znaménka. Varianta `_be` pracuje s opačným pořadím bajtů (big endian).

unsigned get_uint32(int i)

Čte 32bitovou hodnotu bez znaménka. Varianta `_be` pracuje s opačným pořadím bajtů (big endian).

void set_uint32_be(int i, unsigned value)

Viz `set_uint32`.

unsigned get_uint32_be(int i)

Viz `get_uint32`.

void set_float(int i, float value)

Zapíše 32bitovou hodnotu float.

float get_float(int i)

Čte 32bitovou hodnotu float.

void set_double(int i, double value)

Zapíše 64bitovou hodnotu double.

double get_double(int i)

Čte 64bitovou hodnotu double.

void set_value(unsigned64 value, int start_bit, int length, bool big_endian, bool is_signed)

Zapíše signál libovolné délky (1 až 64 bitů) do prvních 8 bajtů dat. Počáteční bit je nejnižší bit signálu; u pořadí Motorola (`big_endian = true`) pokračují vyšší bity do bajtů s nižším indexem (v DBC se u Motorola uvádí nejvyšší bit).

unsigned64 get_value(int start_bit, int length, bool big_endian, bool is_signed)

Přečte signál libovolné délky od zadaného bitu. U signálu se znaménkem je hodnota rozšířena znaménkem (výsledek lze přetypovat na `int64`).

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

Zapíše `length` znaků řetězce do dat zprávy od bajtu `first`.

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

Do `str` uloží `length` bajtů dat od bajtu `first`.

unsigned get_timestamp()

Časová značka přijaté zprávy.

unsigned get_time_h()

Čas přijetí zprávy - hodina.

unsigned get_time_m()

Čas přijetí zprávy - minuta.

unsigned get_time_s()

Čas přijetí zprávy - sekunda.

unsigned get_time_ms()

Čas přijetí zprávy - milisekundy.

void print()

Vypíše zprávu do okna OUTPUT dekadicky, např. ST ID 111-8: 0,0,0,0,0,0,0,0 nebo EXT ID 111-123456(29221440)-8: RTR (11bitová část, 18bitová část, v závorce 29bitový tvar, DLC).

void print_hex()

Jako print, hodnoty hexadecimálně.

Signály (set_value, get_value)

Metody set_value a get_value zapisují a čtou signál libovolné délky 1 až 64 bitů v prvních 8 bajtech dat. Počáteční bit je vždy nejnižší bit signálu (bit 0 = bit 0 bajtu 0, bit 8 = bit 0 bajtu 1). Intel (big_endian = false): další bity rostou směrem k vyšším bajtům. Motorola (big_endian = true): vyšší bity pokračují do bajtů s nižším indexem. Signál 12 bitů od bitu 8 v pořadí Motorola tedy zabírá celý bajt 1 (dolních 8 bitů) a bity 0 až 3 bajtu 0. Pozor: v souborech DBC se u signálů Motorola uvádí počáteční bit jako nejvyšší bit.

CAN FD

Zpráva CAN FD má fd = true a až 64 datových bajtů; platné délky nad 8 bajtů jsou 12, 16, 20, 24, 32, 48 a 64. brs zapne vyšší rychlost v datové fázi. Metody get_/set_ pro bajty, čísla, text a bity pracují s celými 64 bajty. data_from_hex a přiřazení vektoru s více než 8 bajty nastaví fd = true automaticky.

Ukázky: 001_can_message_build.prg, 007_can_fd_messages.prg, 008_can_memory_timepoint_more.prg, P04a/P04b (CAN FD vysílač a přijímač)

obj_lin

LIN zpráva pro převodník USB2LIN: zápis položky LIN tabulky (master/slave) a načtení přijaté LIN zprávy ve skriptu LIN_FUNCTION. Viz kapitola LIN.

Datové složky**unsigned index**

Index položky tabulky (0 až 31).

unsigned id

LIN identifikátor (0 až 63).

int size

Počet datových bajtů (0 až 8).

int port

LIN port (1 nebo 2).

unsigned timeout

Čas čekání na odpověď.

unsigned period

Perioda vysílání položky v režimu master: 0 vypnuto (LIN_PERIOD_OFF), 0xFFFF jednou (LIN_ONE_SHOT).

bool rx

true = data se přijímají (odpověď jiného uzlu), false = data odesílá převodník.

unsigned checksum

Kontrolní součet přijaté zprávy.

unsigned flags

Příznaky přijaté zprávy (bit 0x80 = vypršel čas odpovědi).

unsigned error_code

Kód chyby přijaté zprávy.

Metody

void set()

Zapiše položku tabulky (index, id, size, period, timeout, rx a data) do převodníku. Při chybném indexu, délce nebo portu vypíše hlášení.

char data(int i)

Vrací i-tý datový bajt (0 až 7).

void set_data(int i, char byte)

Nastaví i-tý datový bajt (0 až 7).

void load()

Načte LIN zprávu, která vyvolala spuštění skriptu LIN_FUNCTION (LIN_FUNCTION_AND_TIMER): id, size, port, data, checksum, flags, error_code a čas přijetí.

bool rx_timeout()

Vrací true, pokud u přijaté zprávy vypršel čas odpovědi.

unsigned get_timestamp()

Časová značka přijaté zprávy.

unsigned get_time_h()

Čas přijetí - hodina.

unsigned get_time_m()

Čas přijetí - minuta.

unsigned get_time_s()

Čas přijetí - sekunda.

unsigned get_time_ms()

Čas přijetí - milisekundy.

Ukázka: D05_lin_master.prg

obj_rs232

Práce se sériovým portem.

Metody

bool open(unsigned port_number, unsigned baud_rate)

Otevře zadaný sériový port zadanou rychlostí. Vrací true při úspěchu.

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

Jako open, navíc počet bitů (4 až 8), parita (0 až 4 = none, odd, even, mark, space) a stop bity (0, 1, 2 = 1, 1.5, 2).

void close()

Uzavře port.

bool is_ok()

Vrací true, byl-li port úspěšně otevřen.

bool data_received()

Vrací true, jsou-li k dispozici přijatá data.

char get()

Vrací přijatý bajt.

void send(char data)

Odešle bajt.

void send_string(string str)

Odešle řetězec.

void send_log()

Odešle obsah textového bufferu log (obsah bufferu zůstane).

void set_rts(bool state)

Nastaví signál RTS.

void set_dtr(bool state)

Nastaví signál DTR.

bool get_cts()

Vrací stav signálu CTS.

bool get_dsr()

Vrací stav signálu DSR.

obj_vector_int

Pole celých čísel (zjednodušený kontejner vector). Kontejner udržuje vnitřní iterátor (aktuální prvek), se kterým pracují metody first, next, get, insert, erase, find a další.

Metody

int size()

Vrací počet prvků.

void clear()

Vymaže všechny prvky.

void push_back(int value)

Přidá prvek na konec.

int pop_back()

Vrací poslední prvek a odstraní jej.

int at(int index)

Vrací prvek na pozici index.

void modify(int index, int value)

Změní prvek na pozici index.

int first()

Nastaví iterátor na první prvek a vrací jeho hodnotu.

bool next_exist()

Vrací true, existuje-li za aktuálním prvkem další.

int next()

Posune iterátor na další prvek a vrací jeho hodnotu.

int get()

Vrací aktuální prvek (iterátoru).

void iterator(int index)

Nastaví iterátor na prvek index.

void insert(int value)

Vloží hodnotu před aktuální prvek.

void erase()

Odstraní aktuální prvek.

void remove(int first, int last)

Odstraní prvky v rozsahu first až last-1.

void sort()

Seřadí prvky vzestupně.

void reverse()

Obrátí pořadí prvků.

void fill(int value)

Nastaví všechny prvky na zadanou hodnotu.

void replace_value(int old_value, int new_value)

Nahradí všechny výskyty hodnoty.

void add(int first, int last, int value)

Přičte hodnotu k prvkům first až last-1. Obdobně sub, mul, div.

void sub(int first, int last, int value)

Viz add (odečtení).

void mul(int first, int last, int value)

Viz add (násobení).

void div(int first, int last, int value)

Viz add (dělení).

int sum()

Vrací součet prvků.

double average()

Vrací průměr prvků.

int median()

Vrací medián prvků.

void min()

Nastaví iterátor na nejmenší prvek.

void max()

Nastaví iterátor na největší prvek.

int min_val()

Vrací nejmenší hodnotu.

int max_val()

Vrací největší hodnotu.

int count(int value)

Vrací počet prvků se zadanou hodnotou.

int find(int value)

Hledá hodnotu od aktuálního prvku; vrací její index (iterátor se posune za ni) nebo -1. Opakovaným voláním lze najít všechny výskyty.

int find_string(string str)

Hledá posloupnost znaků řetězce od aktuálního prvku; vrací index nebo -1.

int find_first_lower(int value)

Vrací index prvního prvku menšího než value, nebo -1.

int find_first_bigger(int value)

Vrací index prvního prvku většího než value, nebo -1.

int adjacent_find()

Hledá od začátku dva sousední stejné prvky; vrací index prvního z nich, nebo -1.

int adjacent_next()

Jako adjacent_find, hledá od aktuálního prvku (další výskyt).

int xor()

Vrací XOR všech prvků (např. kontrolní součet).

unsigned64 crc(char bits, unsigned64 poly, unsigned64 init, bool ref_in, bool ref_out, unsigned64 xor_out)

Vypočte CRC z prvků (bajtů) s libovolnými parametry, např. CRC-16/CCITT: crc(16, 0x1021, 0xFFFF, false, false, 0).

unsigned fletcher16_mod255()

Kontrolní součet Fletcher-16 (modulo 255) z prvků (bajtů).

unsigned fletcher16_mod256()

Kontrolní součet Fletcher-16 (modulo 256).

void to_string(string str)

Převéde prvky (dolní bajty) na znaky řetězce.

void tostring(string str)

Stejně jako to_string (jméno používané staršími skripty).

void from_string(string str)

Naplní vektor kódy znaků řetězce (původní obsah se smaže).

void from_string_append(string str)

Jako from_string, znaky se přidají na konec.

void from_hex_string(string hex)

Naplní vektor bajty z hexadecimálního řetězce, např. "30313233".

void encode_base64(string str)

Zakóduje prvky (bajty) do řetězce v kódování Base64.

void decode_base64(string str)

Dekóduje řetězec Base64 do vektoru.

void print_hex()

Vypíše prvky hexadecimálně do okna OUTPUT (8 na řádek).

Ukázky: C01_vector_int_basics.prg, C02_vector_int_algorithms.prg, C03_vector_text_checksums.prg, C08_containers_more.prg

obj_vector_double

Pole desetinných čísel. Metody mají stejný význam jako u obj_vector_int, hodnoty jsou typu double.

Metody**int size()**

Počet prvků.

void clear()

Vymaže prvky.

void push_back(double value)

Přidá prvek na konec.

double pop_back()

Vrací a odstraní poslední prvek.

double at(int index)

Prvek na pozici index.

void modify(int index, double value)

Změní prvek na pozici index.

double first()

Iterátor na první prvek.

bool next_exist()

Existuje další prvek.

double next()

Další prvek.

double get()

Aktuální prvek.

void iterator(int index)

Nastaví iterátor.

void insert(double value)

Vloží hodnotu před aktuální prvek.

void erase()

Odstraní aktuální prvek.

void remove(int first, int last)

Odstraní prvky first až last-1.

void sort()

Seřadí vzestupně.

void reverse()

Obrátí pořadí.

void fill(double value)

Nastaví všechny prvky.

void replace_value(double old_value, double new_value)

Nahradí hodnotu.

void add(int first, int last, double value)

Přičte hodnotu k prvkům first až last-1.

void sub(int first, int last, double value)

Odečte hodnotu.

void mul(int first, int last, double value)

Vynásobí hodnotou.

void div(int first, int last, double value)

Vydělí hodnotou.

double sum()

Součet prvků.

double average()

Průměr prvků.

double median()

Medián prvků.

void min()

Iterátor na nejmenší prvek.

void max()

Iterátor na největší prvek.

double min_val()

Nejmenší hodnota.

double max_val()

Největší hodnota.

int find(double value)

Hledá hodnotu od aktuálního prvku, vrátí index nebo -1.

int count(double min, double max)

Počet prvků v rozsahu min až max (včetně).

int find_first_lower(double value)

Index prvního prvku menšího než value, nebo -1.

int find_first_bigger(double value)

Index prvního prvku většího než value, nebo -1.

int adjacent_find()

Index dvou sousedních stejných prvků, nebo -1.

int adjacent_next()

Další výskyt sousedních stejných prvků.

*Ukázky: C04_vector_double.prg, C08_containers_more.prg***obj_deque_int**

Obousměrná fronta celých čísel (deque). Kontejner udržuje vnitřní iterátor (aktuální prvek), se kterým pracují metody first, next, get, insert, erase, find a další.

Metody**int size()**

Vrací počet prvků.

void clear()

Vymaže všechny prvky.

void push_back(int value)

Přidá prvek na konec.

void push_front(int value)

Přidá prvek na začátek.

int pop_back()

Vrací a odstraní poslední prvek.

int pop_front()

Vrací a odstraní první prvek.

int at(int index)

Vrací prvek na pozici index.

int first()

Nastaví iterátor na první prvek a vrací jej.

bool next_exist()

Vrací true, existuje-li další prvek.

int next()

Posune iterátor a vrací další prvek.

int get()

Vrací aktuální prvek.

void insert(int value)

Vloží hodnotu před aktuální prvek.

void erase()

Odstraní aktuální prvek.

void tostring(string str)

Převede prvky (dolní bajty) na znaky řetězce.

void to_string(string str)

Stejně jako tostring.

Ukázky: C05_deque_queue_stack.prg, C08_containers_more.prg

obj_deque_double

Obousměrná fronta desetinných čísel. Metody mají stejný význam jako u obj_deque_int.

Metody**int size()**

Počet prvků.

void clear()

Vymaže prvky.

void push_back(double value)

Přidá na konec.

void push_front(double value)

Přidá na začátek.

double pop_back()

Vrací a odstraní poslední prvek.

double pop_front()

Vrací a odstraní první prvek.

double at(int index)

Prvek na pozici index.

double first()

První prvek (iterátor).

bool next_exist()

Existuje další prvek.

double next()

Další prvek.

double get()

Aktuální prvek.

void insert(double value)

Vloží před aktuální prvek.

void erase()

Odstraní aktuální prvek.

obj_timepoint

Měření časových intervalů. Objekt si pamatuje okamžik svého vytvoření (start skriptu) a dva časové body nastavené metodami `set_begin` a `set_end`.

Metody**void set_begin()**

Uloží aktuální čas jako počáteční bod.

void set_end()

Uloží aktuální čas jako koncový bod.

unsigned64 duration_s()Interval mezi `set_begin` a `set_end` v sekundách. Varianty `_ms` a `_us` v milisekundách a mikrosekundách.**unsigned64 duration_ms()**Viz `duration_s`.**unsigned64 duration_us()**Viz `duration_s`.**unsigned64 duration_s_swap()**Jako `duration_s` a navíc koncový bod se stane novým počátečním (měření po sobě jdoucích intervalů).**unsigned64 duration_ms_swap()**Viz `duration_s_swap`.**unsigned64 duration_us_swap()**Viz `duration_s_swap`.**unsigned64 duration_from_start_s()**Čas od vytvoření objektu v sekundách. Varianty `_ms` a `_us`.**unsigned64 duration_from_start_ms()**Viz `duration_from_start_s`.**unsigned64 duration_from_start_us()**Viz `duration_from_start_s`.

Ukázka: T16_time_timepoint.prg

obj_csvfile

Práce se soubory CSV (tabulková data oddělená čárkou), které lze importovat do Excelu a exportovat z něj. Hodnoty se čtou a zapisují jako řetězce; pro převod na čísla použijte konverzní funkce.

Metody**void read(string file)**

Načte soubor.

void write(string file)

Zapíše data do souboru.

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

Zapíše hodnotu do zadaného řádku a sloupce.

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

Přečte hodnotu ze zadaného řádku a sloupce do value.

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

Vytvoří nebo zvětší tabulku na zadaný rozměr, nové buňky vyplní hodnotou value.

```
unsigned size_r()
```

Vrací počet řádků.

```
unsigned size_c()
```

Vrací počet sloupců.

Ukázka: 003_files_cfg_csv_log.prg

obj_cfgfile

Práce s konfiguračními soubory cfg/ini se sekce[m]i [sekc]e a položkami jmeno=hodnota. Položky mohou být typu int, unsigned, float, bool (1, 0, y, n, yes, no) a string. Výchozí hodnota se zadává jako text.

Metody

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

Nastaví soubor, ze kterého se čte a do kterého se zapisuje.

```
int read_int(string section, string item, string default_value)
```

Přečte položku jako int; není-li nalezena, vrací default_value.

```
unsigned read_unsigned(string section, string item, string default_value)
```

Přečte položku jako unsigned.

```
float read_float(string section, string item, string default_value)
```

Přečte položku jako float.

```
bool read_bool(string section, string item, string default_value)
```

Přečte položku jako bool.

```
void read_string(string section, string item, string value, string default_value)
```

Přečte položku jako řetězec do value.

```
void write_int(string section, string item, int value)
```

Zapíše položku.

```
void write_unsigned(string section, string item, unsigned value)
```

Zapíše položku.

```
void write_float(string section, string item, float value)
```

Zapíše položku.

```
void write_bool(string section, string item, bool value)
```

Zapíše položku.

```
void write_string(string section, string item, string value)
```

Zapíše položku.

Jméno souboru bez cesty se vztahuje k pracovnímu adresáři PP2CAN.

Ukázka: 003_files_cfg_csv_log.prg

obj_logfile

Zápis do textového souboru, například záznamu událostí.

Metody

void set_name(string file)

Nastaví (otevře) soubor pro zápis.

void write_log()

Zapíše obsah textového bufferu log a konec řádku.

void write_string(string str)

Zapíše řetězec.

void write_string_endl(string str)

Zapíše řetězec a konec řádku.

void close()

Uloží a zavře soubor.

obj_datagrid

Okno pro výpis až 10 veličin (index 0 až 9) s názvem a hodnotou. Obdobné funkcím ui_..., lze však použít více oken.

Metody

void open(string name)

Otevře okno se zadaným názvem.

void set_name(int index, string name)

Nastaví název veličiny.

void set_value_string(int index, string value)

Nastaví hodnotu veličiny jako text.

void set_value_int(int index, int value)

Nastaví hodnotu typu int.

void set_value_unsigned(int index, unsigned value)

Nastaví hodnotu typu unsigned.

void set_value_double(int index, double value)

Nastaví hodnotu typu double.

void update()

Zobrazí nastavené názvy a hodnoty.

void save_pos()

Uloží polohu okna (podle jeho názvu) do nastavení aplikace.

void load_pos()

Načte polohu okna uloženou metodou save_pos.

Zobrazení dat	
Scrollbar	0
Radio	2
BO	Off
INT	-547
UNSIGNED	453
DOUBLE	-54.699999
	11111111111111

Obr. 4: Okno objektu `obj_datagrid`

obj_dataset

Okno pro zadávání až 10 hodnot uživatelem (index 0 až 9). Uživatel zapíše hodnotu a potvrdí ji tlačítkem; skript ji zjistí metodou `updated` a přečte metodami `get_value_...`. Celá čísla lze zadat i hexadecimálně (0x...).

Metody

`void open(string name)`

Otevře okno se zadaným názvem.

`void set_name(int index, string name)`

Nastaví název položky.

`bool updated()`

Vrací `true`, pokud uživatel od posledního volání potvrdil novou hodnotu.

`void get_value_string(int index, string value)`

Přečte hodnotu položky jako text.

`int get_value_int(int index)`

Přečte hodnotu jako `int`.

`unsigned get_value_unsigned(int index)`

Přečte hodnotu jako `unsigned`.

`double get_value_double(int index)`

Přečte hodnotu jako `double`.

`void set_value_string(int index, string value)`

Přednastaví hodnotu položky (jako by ji zadal uživatel).

`void set_value_int(int index, int value)`

Přednastaví hodnotu typu `int`.

`void set_value_unsigned(int index, unsigned value)`

Přednastaví hodnotu typu `unsigned`.

`void set_value_double(int index, double value)`

Přednastaví hodnotu typu `double`.

`void save_pos()`

Uloží polohu okna (podle jeho názvu) do nastavení aplikace.

void load_pos()

Načte polohu okna uloženou metodou save_pos.

obj_datagraph

Jednoduchý průběžný graf až 4 veličin (index 0 až 3) v rozsahu 0 až 60000. Okno otevíráte metodou open v sekci init. Nastavení ze skriptu se do okna promítne v jeho časovači (do 50 ms).

Metody**void open(string name)**

Otevře okno se zadaným názvem.

void set_value(int index, int value)

Přidá hodnotu veličiny.

void set_min(int value)

Nastaví minimum grafu (0 až 59000).

void set_max(int value)

Nastaví maximum grafu (1 až 60000).

void show_as_bar(int index, bool bar)

Zobrazí veličinu jako sloupcový graf.

void update()

Překreslí graf.

void save_pos()

Uloží polohu okna (podle jeho názvu) do nastavení aplikace.

void load_pos()

Načte polohu okna uloženou metodou save_pos.



Obr. 5: Okno objektu obj_datagraph

Ukázky: D02_display_windows.prg, D04_can_receiver_monitor.prg

obj_datagauge

Okno s ručkovým ukazatelem (budíkem) se třemi barevnými pásmy stupnice. Okno otevíráte metodou open v sekci init. Nastavení ze skriptu se do okna promítne v jeho časovači (do 50 ms).

Metody**void open(string name)**

Otevře okno se zadaným názvem.

void set_value(double value)

Nastaví zobrazenou hodnotu.

void set_range(double min, double max)

Nastaví rozsah stupnice.

void set_range1(double value)

Nastaví konec prvního (zeleného) pásma.

void set_range2(double value)

Nastaví konec druhého (žlutého) pásma; zbytek je červený.

void set_angle_range(double start, double end)

Nastaví úhlový rozsah stupnice.

void set_steps(int major, int minor)

Nastaví počet hlavních a vedlejších dílků stupnice.

void set_background(int r1, int g1, int b1, int r2, int g2, int b2)

Nastaví barvy pozadí (přechod ze dvou barev RGB).

void set_needle_color(int r, int g, int b)

Nastaví barvu ručky.

void set_digit_pos(int x, int y)

Nastaví polohu číselného údaje.

void save_pos()

Uloží polohu okna (podle jeho názvu) do nastavení aplikace.

void load_pos()

Načte polohu okna uloženou metodou save_pos.

obj_socket_server

Server pro komunikaci protokolem TCP nebo UDP (vždy jen jeden z nich a jeden port). K serveru se může připojit více klientů.

Metody

void OpenTCP(unsigned port)

Otevře TCP port.

void OpenUDP(unsigned port)

Otevře UDP port.

void close()

Uzavře port.

bool is_open()

Vrací true, běží-li server.

bool data_received()

Vrací true, jsou-li přijata data.

char get()

Vrací přijatý bajt.

void send(char data)

Odešle bajt všem klientům (TCP).

void send_string(string str)

Odešle řetězec všem klientům (TCP).

void send_udp(char data, char ip1, char ip2, char ip3, char ip4)

Odešle bajt (UDP) na zadanou IP adresu.

```
void send_string_udp(string str, char ip1, char ip2, char ip3, char ip4)
```

Odešle řetězec (UDP) na zadanou IP adresu.

```
void buffer_push(char data)
```

Přidá bajt do vysílacího bufferu (max. 1024 bajtů). Paket je vhodné odeslat najednou.

```
void buffer_send()
```

Odešle obsah bufferu (TCP) a buffer vyprázdní.

Ukázka: P03a_tcp_echo_server.prg

obj_socket_client

Klient pro komunikaci protokolem TCP nebo UDP.

Metody

```
void OpenTCP(unsigned ip1, unsigned ip2, unsigned ip3, unsigned ip4, unsigned port)
```

Připojí se k serveru na zadané IP adrese a TCP portu.

```
void OpenUDP(unsigned ip1, unsigned ip2, unsigned ip3, unsigned ip4, unsigned port)
```

Připojí se k serveru na zadané IP adrese a UDP portu.

```
void close()
```

Uzavře spojení.

```
bool is_open()
```

Vrací true, je-li spojení otevřeno.

```
bool data_received()
```

Vrací true, jsou-li přijata data.

```
char get()
```

Vrací přijatý bajt.

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

Odešle bajt.

```
void send_string(string str)
```

Odešle řetězec.

```
void buffer_push(char data)
```

Přidá bajt do vysílacího bufferu (max. 1024 bajtů).

```
void buffer_send()
```

Odešle obsah bufferu (TCP) a buffer vyprázdní.

Ukázka: P03b_tcp_client.prg

obj_pid

PID regulátor. Výchozí nastavení: dt 1, max 255, min 0, Kp 0.1, Kd 0.01, Ki 0.5.

Metody

```
void set_dt(double dt)
```

Perioda výpočtu.

```
void set_max(double max)
```

Maximum výstupu.

```
void set_min(double min)
```

Minimum výstupu.

void set_kp(double kp)

Proporcionální zesílení.

void set_ki(double ki)

Integrační zesílení.

void set_kd(double kd)

Derivační zesílení.

double calculate(double setpoint, double process_value)

Vypočte akční veličinu pro žádanou a skutečnou hodnotu (volat s periodou dt).

Ukázka: 004_pid_simulation.prg

obj_memory

Blok paměti, např. pro předání dat funkci v DLL (obj_dllfnc). Metody get_/set_ mají stejný význam jako u obj_can_msg a kontrolují rozsah indexu.

Metody

void alloc(unsigned size)

Alokuje blok zadané velikosti v bajtech (předchozí obsah uvolní).

void free()

Uvolní blok.

unsigned size()

Vrací velikost bloku.

unsigned pointer()

Vrací adresu bloku (pro obj_dllfnc.set_param_memory).

void set(unsigned address, char value)

Zapíše bajt na zadanou pozici; mimo velikost bloku se nic nezapíše.

char get(unsigned address)

Čte bajt ze zadané pozice.

void fill(char value)

Vyplní blok hodnotou.

void print()

Vypíše obsah bloku do okna OUTPUT.

void set_int16(int i, int value)

Viz obj_can_msg.

int get_int16(int i)

Viz obj_can_msg.

void set_int16_be(int i, int value)

Viz obj_can_msg.

int get_int16_be(int i)

Viz obj_can_msg.

void set_uint16(int i, unsigned value)

Viz obj_can_msg.

unsigned get_uint16(int i)

Viz obj_can_msg.

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

Viz obj_can_msg.

```
unsigned get_uint16_be(int i)
```

Viz obj_can_msg.

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

Viz obj_can_msg.

```
int get_int32(int i)
```

Viz obj_can_msg.

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

Viz obj_can_msg.

```
int get_int32_be(int i)
```

Viz obj_can_msg.

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

Viz obj_can_msg.

```
unsigned get_uint32(int i)
```

Viz obj_can_msg.

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

Viz obj_can_msg.

```
unsigned get_uint32_be(int i)
```

Viz obj_can_msg.

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

Viz obj_can_msg.

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

Viz obj_can_msg.

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

Viz obj_can_msg.

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

Viz obj_can_msg.

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

Viz obj_can_msg.

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

Viz obj_can_msg.

Ukázka: 002_memory_object.prg

obj_dllfnc

Volání funkce z externí knihovny DLL. Postup: load_dll, set_abi, set_param_num, typy parametrů (set_param_type) a návratové hodnoty (set_return_type), hodnoty parametrů (set_param...), call a přečtení výsledku (get_return...). Typy se zadávají konstantami PARAM_..., volací konvence konstantami ABI_... (podporována je ABI_STDCALL). Nejvýše 32 parametrů. Objekt je dostupný jen v 32bitové verzi PP2CAN.

Metody

```
bool load_dll(string dll_name, string function_name)
```

Načte knihovnu a vyhledá funkci.

bool set_abi(int abi)

Nastaví volací konvenci (ABI_STDCALL).

bool set_param_num(unsigned count)

Nastaví počet parametrů.

bool set_param_type(unsigned index, unsigned type)

Nastaví typ parametru (PARAM_UINT8 ... PARAM_POINTER).

bool set_return_type(unsigned type)

Nastaví typ návratové hodnoty (PARAM_VOID ... PARAM_POINTER).

bool set_param_sint8(unsigned index, int value)

Nastaví hodnotu parametru. Obdobně pro další typy.

bool set_param_uint8(unsigned index, unsigned value)

Viz set_param_sint8.

bool set_param_sint16(unsigned index, int value)

Viz set_param_sint8.

bool set_param_uint16(unsigned index, unsigned value)

Viz set_param_sint8.

bool set_param_sint32(unsigned index, int value)

Viz set_param_sint8.

bool set_param_uint32(unsigned index, unsigned value)

Viz set_param_sint8.

bool set_param_sint64(unsigned index, int64 value)

Viz set_param_sint8.

bool set_param_uint64(unsigned index, unsigned64 value)

Viz set_param_sint8.

bool set_param_float(unsigned index, float value)

Viz set_param_sint8.

bool set_param_double(unsigned index, double value)

Viz set_param_sint8.

bool set_param_string(unsigned index, string value)

Předá ukazatel na řetězec (PARAM_POINTER).

bool set_param_memory(unsigned index, unsigned pointer)

Předá ukazatel na blok obj_memory (hodnota z obj_memory.pointer()).

bool call()

Zavolá funkci. Při chybě (nenačtená funkce, nenastavené typy nebo hodnoty) vypíše hlášení a vrací false.

int get_return_sint8()

Vrací návratovou hodnotu. Obdobně pro další typy.

unsigned get_return_uint8()

Viz get_return_sint8.

int get_return_sint16()

Viz get_return_sint8.

unsigned get_return_uint16()

Viz get_return_sint8.

int get_return_sint32()

Viz get_return_sint8.

unsigned get_return_uint32()

Viz get_return_sint8.

int64 get_return_sint64()

Viz get_return_sint8.

unsigned64 get_return_uint64()

Viz get_return_sint8.

float get_return_float()

Viz get_return_sint8.

double get_return_double()

Viz get_return_sint8.

obj_uds

Přenos dat delších než jedna CAN zpráva podle ISO-TP (ISO 15765-2), např. pro diagnostiku UDS. Klasický CAN i CAN FD, normální, rozšířené a smíšené adresování, data do 16 MB. Viz kapitola Přenos dlouhých dat.

Datové složky

unsigned tx_id

ID odesílaných rámců (výchozí 0x7E0).

unsigned rx_id

ID přijímaných rámců (výchozí 0x7E8).

bool ext_id

29bitová ID.

int port

Port (CAN kanál) vydávaných rámců (výchozí 1).

bool fd

CAN FD.

bool brs

CAN FD: přepnutí přenosové rychlosti.

int frame_size

Délka rámce u CAN FD (8 až 64, výchozí 8).

int addressing

Adresování: ISOTP_NORMAL (výchozí), ISOTP_EXTENDED, ISOTP_MIXED. U rozšířeného a smíšeného je první bajt rámce adresa.

unsigned tx_address

Adresní bajt odesílaných rámců.

unsigned rx_address

Adresní bajt přijímaných rámců.

int padding

Výplň nevyužitých bajtů (výchozí 0xCC); -1 = bez výplně (zkrácené DLC).

int rx_block_size

Velikost bloku (BS) ve Flow Control, který objekt nachystá při příjmu (0 = bez omezení).

int rx_stmin

STmin ve Flow Control, který objekt nachystá při příjmu.

Metody**void clear()**

Vymaže data a ukončí probíhající přenos v obou směrech.

void set_string(string str)

Nastaví data k odeslání podle řetězce.

void set_data(int i, char byte)

Nastaví i-tý bajt dat (data se podle potřeby prodlouží).

char data(int i)

Vrací i-tý bajt dat (odeslaných nebo přijatých).

int size()

Vrací počet bajtů dat.

void get_string(string str)

Uloží data do řetězce (max. 1023 znaků).

void start()

Rozdělí data na rámce (Single Frame, nebo First Frame a Consecutive Frames) a zahájí odesílání. První rámec je k dispozici hned. Při neplatném nastavení nebo bez dat nastaví error() na TP_ERR_CONFIG.

bool has_frame()

Vrací true, má-li objekt rámec k odeslání (datový rámec povolený řízením toku nebo připravenou odpověď).

bool next_frame(obj_can_msg msg)

Vydá další rámec do msg a vrací true; není-li co odeslat, vrací false a msg nemění. Stejně funguje přiřazení msg = objekt;.

void flow(int n)

Povolí odeslání dalších n datových rámců (0 = všech zbývajících). Volá skript po přijetí Flow Control nebo CTS od protistrany.

void select(int index)

Nastaví index dalšího odesílaného rámce (0 = první rámec). U J1939 odpovídá index číslu paketu z CTS.

void abort()

Ukončí přenos. U J1939 se spojením (RTS/CTS, ETP) nachystá zprávu Abort k odeslání.

int frame_count()

Vrací počet rámců odesílaného přenosu.

bool tx_done()

Vrací true, byly-li vydány všechny rámce přenosu.

bool receive(obj_can_msg msg)

Předá přijatý rámec. Vrací true, pokud rámec patří k nastavenému spojení a objekt ho zpracoval; rámce řízení toku pro odesílání (Flow Control, CTS, EOMA) objekt nezpracovává a vrací false. Stejně funguje přiřazení objekt = msg;.

bool complete()

Vrací true, jsou-li přijata všechna data. Data jsou pak dostupná metodami data, size, get_string nebo přiřazením vec = objekt;.

int error()

Vrací kód chyby: TP_OK, TP_ERR_SEQUENCE (špatné pořadí rámců), TP_ERR_UNEXPECTED (rámec mimo přenos), TP_ERR_SIZE (příliš dlouhá data), TP_ERR_ABORT (přerušeno), TP_ERR_CONFIG (neplatné nastavení), TP_ERR_FRAME (neplatný rámec).

Ukázky: 005_uds_loopback.prg, P02a_uds_ecu_simulator.prg, P02b_uds_tester.prg

obj_j1939

Přenos dat podle SAE J1939: samostatná zpráva, J1939-21 TP (BAM, RTS/CTS) a ETP, J1939-22 pro CAN FD (Multi-PG, FD.TP). Protokol se volí podle délky dat a cíle. Viz kapitola Přenos dlouhých dat.

Datové složky**unsigned pgn**

PGN přenášené skupiny parametrů.

int priority

Priorita samostatné zprávy a Multi-PG (výchozí 6). Rámce transportního protokolu mají prioritu 7.

unsigned source

Vlastní adresa (výchozí 0xF9).

unsigned dest

Adresa protistrany, 0xFF = všem (výchozí). Při příjmu se s jinou hodnotou než 0xFF přijímají zprávy jen od této adresy.

int port

Port (CAN kanál) vydávaných rámců (výchozí 1).

bool fd

J1939-22 (CAN FD).

bool brs

J1939-22: přepnutí přenosové rychlosti (výchozí true).

int rx_packets_per_cts

Počet paketů v CTS, které objekt nachystá při příjmu (0 = maximum 255).

Metody**void clear()**

Vymaže data a ukončí probíhající přenos v obou směrech.

void set_string(string str)

Nastaví data k odeslání podle řetězce.

void set_data(int i, char byte)

Nastaví i-tý bajt dat (data se podle potřeby prodlouží).

char data(int i)

Vrací i-tý bajt dat (odeslaných nebo přijatých).

int size()

Vrací počet bajtů dat.

void get_string(string str)

Uloží data do řetězce (max. 1023 znaků).

void start()

Rozdělí data na rámce (samostatná zpráva, BAM, RTS/CTS, ETP nebo J1939-22) a zahájí odesílání. První rámec je k dispozici hned. Při neplatném nastavení nebo bez dat nastaví error() na TP_ERR_CONFIG.

bool has_frame()

Vrací true, má-li objekt rámec k odeslání (datový rámec povolený řízením toku nebo připravenou odpověď).

bool next_frame(obj_can_msg msg)

Vydá další rámec do msg a vrací true; není-li co odeslat, vrací false a msg nemění. Stejně funguje přiřazení msg = objekt;.

void flow(int n)

Povolí odeslání dalších n datových rámců (0 = všech zbývajících). Volá skript po přijetí Flow Control nebo CTS od protistrany.

void select(int index)

Nastaví index dalšího odesílaného rámce (0 = první rámec). U J1939 odpovídá index číslu paketu z CTS.

void abort()

Ukončí přenos. U J1939 se spojením (RTS/CTS, ETP) nachystá zprávu Abort k odeslání.

int frame_count()

Vrací počet rámců odesílaného přenosu.

bool tx_done()

Vrací true, byly-li vydány všechny rámce přenosu.

bool receive(obj_can_msg msg)

Předá přijatý rámec. Vrací true, pokud rámec patří k nastavenému spojení a objekt ho zpracoval; rámce řízení toku pro odesílání (Flow Control, CTS, EOMA) objekt nezpracovává a vrací false. Stejně funguje přiřazení objekt = msg;.

bool complete()

Vrací true, jsou-li přijata všechna data. Data jsou pak dostupná metodami data, size, get_string nebo přiřazením vec = objekt;.

int error()

Vrací kód chyby: TP_OK, TP_ERR_SEQUENCE (špatné pořadí rámců), TP_ERR_UNEXPECTED (rámec mimo přenos), TP_ERR_SIZE (příliš dlouhá data), TP_ERR_ABORT (přerušeno), TP_ERR_CONFIG (neplatné nastavení), TP_ERR_FRAME (neplatný rámec).

Ukázka: 006_j1939_loopback.prg

Přenos dlouhých dat (UDS, J1939)

Objekty `obj_uds` (ISO-TP) a `obj_j1939` (SAE J1939) dělí data delší než jedna CAN zpráva na rámce a z přijatých rámců data skládají. Časování (STmin, prodlevy mezi rámci BAM) a vyhodnocení řízení toku od protistrany řídí skript: objekt vydá další datové rámce teprve poté, co je skript povolí metodou `flow(n)`. Do té doby `has_frame()` vrací `false`. Jeden objekt odpovídá jednomu spojení a umí oba směry; pro souběžné odesílání i příjem použijte dva objekty.

Odesílání

- Data se naplní metodami `set_string`, `set_data` nebo přiřazením `objekt = vec`; z `obj_vector_int`.
- `start()` připraví rámce. První rámec (Single Frame, First Frame, BAM, RTS) je k dispozici hned.
- Rámce se vydávají metodou `next_frame(msg)` nebo přiřazením `msg = objekt`; a odesílají `msg.send()`.
- Po přijetí Flow Control (UDS) nebo CTS (J1939) od protistrany skript zavolá `flow(n)`, u J1939 předtím `select`(číslo paketu z CTS). Flow Control se stavem overflow nebo zpráva Abort se řeší metodou `abort()`.
- U J1939 CTS s počtem paketů 0 znamená pozdržení přenosu: `flow` v tom případě nevolejte (`flow(0)` povoluje všechny rámce).

Příjem

- Přijaté zprávy se předávají metodou `receive(msg)` nebo přiřazením `objekt = msg`. Objekt přijme jen rámce svého spojení (ID a adresa u UDS, PGN a adresy u J1939).
- Na First Frame (UDS) nebo RTS (J1939) objekt nachystá odpověď (Flow Control, CTS; na konci J1939 EOMA). Skript ji vydá jako ostatní rámce (`next_frame`) a odešle.
- Po přijetí všech dat vrací `complete()` `true`; chyby (např. vynechaný rámec) hlásí `error()`.
- Přijetí nového přenosu přepíše data objektu.

Protokoly obj_j1939

Data	dest = 0xFF (vše)	dest = adresa
do 8 B	samostatná zpráva	samostatná zpráva
9 až 1785 B	TP BAM	TP RTS/CTS
1786 B až 16 MB	nelze	ETP (DPO se vkládá automaticky)
<code>fd = true</code> , do 60 B	Multi-PG	Multi-PG
<code>fd = true</code> , nad 60 B	FD.TP BAM (do 15300 B)	FD.TP RTS/CTS

PGN s formátem PDU2 (PF 240 a více) se vždy vysílá všem. Abort má důvod 3 (timeout). Zprávy CTS a EOMA pro vlastní odesílání vyhodnocuje skript: TP.CM (PGN 0xEC00) s řídicím bajtem 0x11 (CTS: bajt 1 počet paketů, bajt 2 číslo dalšího paketu) a 0x13 (EOMA), ETP.CM (0xC800) s bajtem 0x15 (CTS: číslo paketu v bajtech 2 až 4) a 0x17, J1939-22 FD.TP.CM (0x4D00) s řídicím kódem v dolních 4 bitech bajtu 0 (1 = CTS: číslo segmentu v bajtech 4 až 6, počet v bajtu 7; 3 = EOMA).

Příklad: UDS čtení DID (0x22)

```
script:
    type = CAN_FUNCTION_AND_TIMER;
end
variables:
    bool sent;
end
objects:
```

```

    obj_uds uds;
    obj_can_msg msg;
    obj_vector_int vec;
end
init:
    uds.tx_id = 0x7E0;
    uds.rx_id = 0x7E8;
    uds.padding = 0x55;
    sent = false;
    ui_set_timer_script_period(10);
end
body:
    if( is_timer_call() ) {
        if( !sent ) {
            uds.clear();
            uds.set_data(0, 0x22);
            uds.set_data(1, 0xF1);
            uds.set_data(2, 0x90);
            uds.start();
            sent = true;
        }
        // požadavek a Flow Control k odpovědi
        while( uds.next_frame(msg) ) { msg.send(); }
    }
    else {
        msg.load();
        if( uds.receive(msg) ) {
            if( uds.complete() ) {
                vec = uds;
                vec.print_hex();
            }
        }
        else if( msg.id1 == 0x7E8 && (msg.data(0) & 0xF0) == 0x30 ) {
            // Flow Control k našemu dlouhému požadavku
            if( (msg.data(0) & 0x0F) == 0 ) { uds.flow(msg.data(1)); }
            if( (msg.data(0) & 0x0F) == 2 ) { uds.abort(); }
        }
    }
}
end

```

Příklad: J1939 odeslání přes RTS/CTS a BAM

```

script:
    type = CAN_FUNCTION_AND_TIMER;
end
variables:
    unsigned id;
end
objects:
    obj_j1939 j;
    obj_can_msg msg;
end
init:
    j.pgn = 0xEF00;
    j.source = 0x10;
    j.dest = 0x30;           // 0xFF = BAM
    j.set_string("Data delsi nez 8 bajtu");
    j.start();
    ui_set_timer_script_period(50);
end
body:
    if( is_can_function_call() ) {
        msg.load();
        if( !j.receive(msg) && msg.stext ) {
            id = msg.get_ID29();
            // TP.CM_CTS pro nas: select a flow
            if( ((id >> 16) & 0xFF) == 0xEC && (id & 0xFF) == 0x30 && msg.data(0) == 0x11 ) {
                if( msg.data(1) > 0 ) {

```

```
                j.select((int)msg.data(2));
                j.flow((int)msg.data(1));
            }
        }
    }
else {
    // jeden ramec za periodu casovace (u BAM 50 az 200 ms)
    if( j.next_frame(msg) ) { msg.send(); }
}
end
```

Metody `next_frame` a `receive` přebírají objekt `obj_can_msg` jako parametr.

LIN

S převodníkem USB2LIN může skript nastavit LIN port (rychlost, režim master/slave/scan, verzi LIN), zapisovat položky LIN tabulky a zpracovávat přijaté LIN zprávy. Pro příjem slouží typy skriptu LIN_FUNCTION a LIN_FUNCTION_AND_TIMER; PP2CAN je volá pro každou LIN zprávu přijatou při otevřeném dialogu LIN tabulky. Kdo chce LIN zprávy zpracovávat ve skriptu CAN_FUNCTION, nastaví v PP2CAN režim, ve kterém se LIN zpráva předává jako CAN zpráva.

- Režim master: převodník periodicky vysílá hlavičky položek tabulky s nenulovou periodou (LIN_ONE_SHOT = jednu). Je-li rx = false, doplní i data, při rx = true čeká na odpověď slave.
- Režim slave: převodník odpovídá daty na hlavičku se stejným ID, pokud rx = false; při rx = true data jen přijímá.
- Režim scan: převodník jen poslouchá provoz na sběrnici.

```
script:
    type = LIN_FUNCTION_AND_TIMER;
end
variables:
    bool ok;
end
objects:
    obj_lin tab;
    obj_lin rx;
end
init:
    ok = lin_config(1, LIN_MASTER, 19200, LIN_VER_20);
    tab.port = 1;
    tab.index = 0;
    tab.id = hex(10);
    tab.size = 8;
    tab.period = 100;
    tab.timeout = 50;
    tab.rx = true;
    tab.set();
end
body:
    if( is_lin_function_call() ) {
        rx.load();
        if( rx.rx_timeout() ) { prints("timeout"); }
        else { logc_hex(rx.data(0)); log_endl(); }
    }
end
```

Prostředí interpretu

- Současně mohou běžet 4 skripty. Každý má vlastní proměnné; data mezi skripty se předávají globálními proměnnými (global_..._get/set).
- Sekce init se provede při spuštění, sekce body při každém volání podle typu skriptu, sekce shutdown při zastavení. Proměnné si mezi voláními zachovávají hodnotu.
- Jedno provedení sekce body je omezeno na 5 s; delší běh (např. nekonečný cyklus) PP2CAN ukončí. Funkci wait používejte jen výjimečně.
- Okna objektů (obj_datagrid, obj_dataset, obj_datagraph, obj_datagauge) otevírejte v sekci init. Nastavení ze skriptu se do okna promítne jeho časovačem (do 50 ms).
- Řetězec má nejvýše 1023 znaků, buffer funkcí log nejvýše 511 znaků; delší text se zkrátí.
- Celočíslné dělení nulou vypíše hlášení Division by zero a výsledek je 0.
- Vnoření volání funkcí (uživatelských i func_N) je omezeno na 10 úrovní, rekurze není dovolena.
- V režimu V2CAN nebo při propojených portech se odeslaná zpráva může přijmout i vícekrát (ozvěna, druhý port). Skripty, které přijímají vlastní nebo opakované zprávy, by měly duplicity rozpoznat (např. podle čítače, viz D04 a P04b).

Překladač

Tlačítko Compile v okně skriptu spustí překladač compiler.exe. Ten zkontroluje syntaxi a sémantiku (deklarace, typy, počty parametrů) a přeloží skript do binárního tvaru (.scr), který vykonává interpret. Chyby se vypisují ve tvaru <file: jméno> <line: číslo> error: text; po úspěšném překladu se vypíše Script ready.

Překladač lze spustit i z příkazové řádky:

```
compiler infile -t|-b outfile [-cz|-en]
```

Parametr	Význam
-t	textový výstup (výpis instrukcí pro kontrolu)
-b	binární výstup .scr pro PP2CAN
-cz	chybová hlášení česky (výchozí)
-en	chybová hlášení anglicky

Ukázkové a testovací skripty

K příručce je přiložena sada skriptů ve dvou variantách: složka `cz` obsahuje skripty s českými komentáři (bez diakritiky), složka `en` s anglickými. Názvy proměnných a objektů jsou anglicky. Komentáře „Napoveda:“ / „Help:“ u prvního použití každé funkce, metody, klíčového slova a operátoru vysvětlují jejich význam, takže skripty slouží i jako výuková pomůcka.

Samokontrolní testy

Skripty `T`, `C` a `O` jsou samokontrolní: každý výsledek výpočtu nebo převodu porovnají s očekávanou hodnotou, vypíší `PASS` nebo `FAIL` a na konci souhrn, např. `==== T01: 29 PASS, 0 FAIL =====`. Testy běží v sekci `init`, sekce body skript ukončí. Ve výstupu by neměl být žádný `FAIL`. Porovnání provádějí uživatelské funkce `check_int`, `check_double`, `check_str` a další, které jsou na začátku každého testu.

Ukázky a dvojice skriptů

- Skripty `D` jsou interaktivní ukázky (okno skriptu, datová okna, CAN vysílač a přijímač, LIN).
- Dvojice (`D03 + D04`, `P01a + P01b`, `P02a + P02b`, `P03a + P03b`, `P04a + P04b`) se spouštějí současně ve dvou oknech CAN skriptu. Pro dvojice komunikující po CAN použijte režim `V2CAN`, dva propojené porty nebo dva převodníky.
- `D04` a `P04b` rozpoznají duplicitní zprávy (stejný čítač) a nezapočítávají je jako ztráty; `P02a` a `P02b` ignorují bezprostředně opakovaný rámec.
- `O03` si soubory `script_test.cfg`, `script_test.csv` a `script_test.log` vytvoří sám v pracovním adresáři PP2CAN; žádné předem připravené soubory nejsou potřeba.
- `P03a` a `P03b` komunikují přes TCP na adrese `127.0.0.1`, port `5000`; nejdřív spusťte server `P03a`.
- `D05` vyžaduje převodník `USB2LIN` a otevřený dialog `LIN` tabulky.

Seznam skriptů

Skript	Obsah
<code>C01_vector_int_basics.prg</code>	<code>obj_vector_int</code> - základy
<code>C02_vector_int_algorithms.prg</code>	<code>obj_vector_int</code> - algoritmy
<code>C03_vector_text_checksums.prg</code>	<code>obj_vector_int</code> - texty, hexadecimální data a kontrolní součty
<code>C04_vector_double.prg</code>	<code>obj_vector_double</code> - pole desetinných čísel
<code>C05_deque_queue_stack.prg</code>	<code>obj_deque_int</code> a <code>obj_deque_double</code> - fronta a zásobník
<code>C06_object_assignment.prg</code>	Přiřazení objektů
<code>C07_object_assignment_matrix.prg</code>	Přiřazení objektů - všechny podporované kombinace
<code>C08_containers_more.prg</code>	Další metody kontejnerů
<code>D01_ui_controls.prg</code>	Ovládací prvky okna skriptu (interaktivní ukázka)
<code>D02_display_windows.prg</code>	Okna pro zadávání a zobrazení hodnot
<code>D03_can_sender.prg</code>	Periodický vysílač CAN zpráv (dvojice s <code>D04</code>)
<code>D04_can_receiver_monitor.prg</code>	Přijímač a monitor CAN zpráv (dvojice s <code>D03</code>)
<code>D05_lin_master.prg</code>	LIN master s převodníkem <code>USB2LIN</code>
<code>O01_can_message_build.prg</code>	<code>obj_can_msg</code> - sestavení a čtení CAN zprávy (bez odeslání)
<code>O02_memory_object.prg</code>	<code>obj_memory</code> - blok paměti
<code>O03_files_cfg_csv_log.prg</code>	Soubory: konfigurace (<code>cfg/ini</code>), tabulka CSV a záznam (<code>log</code>)
<code>O04_pid_simulation.prg</code>	<code>obj_pid</code> - simulace regulace
<code>O05_uds_loopback.prg</code>	<code>obj_uds</code> (ISO-TP) - tester a řídicí jednotka v jednom skriptu

Skript	Obsah
006_j1939_loopback.prg	obj_j1939 - vysílač a přijímač v jednom skriptu
007_can_fd_messages.prg	CAN FD zprávy (obj_can_msg)
008_can_memory_timepoint_more.prg	Další metody obj_can_msg, obj_memory a obj_timepoint
P01a_global_producer.prg	Výrobce dat přes globální proměnné (spusťte současně s P01b)
P01b_global_consumer.prg	Konzument dat přes globální proměnné (spusťte současně s P01a)
P02a_uds_ecu_simulator.prg	Simulátor řídicí jednotky UDS (spusťte s P02b nebo s reálným testerem)
P02b_uds_tester.prg	UDS tester (spusťte s P02a nebo proti reálné řídicí jednotce)
P03a_tcp_echo_server.prg	TCP server s ozvěnou (spusťte s P03b)
P03b_tcp_client.prg	TCP klient (spusťte po P03a)
P04a_can_fd_sender.prg	Vysílač CAN FD zpráv (spusťte s P04b)
P04b_can_fd_receiver.prg	Přijímač CAN FD zpráv (spusťte s P04a)
T01_data_types_constants.prg	Datové typy a zápis konstant
T02_arithmetic.prg	Aritmetické operátory
T03_implicit_conversions.prg	Implicitní (automatické) konverze typů
T04_explicit_casts.prg	Explicitní přetypování (typ)výraz
T05_compound_assignment.prg	Složené přiřazení a ++ --
T06_bitwise_operators.prg	Bitové operátory & ^ ~ << >>
T07_comparison_logic.prg	Porovnání a logické operátory
T08_control_flow.prg	Řídicí struktury: if, else if, while, for, switch
T09_user_functions.prg	Uživatelské funkce
T10_func_sections.prg	Sekce func_0 až func_9 a příkaz func(N)
T11_strings.prg	Práce s řetězci
T12_string_conversions.prg	Převody mezi čísly a řetězci
T13_math_functions.prg	Matematické funkce
T14_bit_functions_bcd.prg	Funkce pro bity, bajty a kód BCD
T15_global_variables.prg	Globální proměnné (sdílené mezi až 4 skripty PP2CAN)
T16_time_timepoint.prg	Datum, čas a měření intervalů (obj_timepoint)
T17_switch_details.prg	Podrobnosti příkazu switch
T18_more_functions.prg	Další vestavěné funkce
T19_edge_cases.prg	Okrajové případy a bezpečnost jazyka
T20_expressions_priority.prg	Výrazy a priorita operátorů
T21_body_calls.prg	Opakované volání sekce body