

Skriptovací jazyk pro prostředí PP2CAN

Novinky a opravy

Překladač a interpret verze 3.00

CANLAB s.r.o. · www.canlab.cz

Přehled

Dokument shrnuje novinky skriptovacího jazyka PP2CAN ve verzi překladače a interpretu 3.00 a opravy chyb předchozích verzí. Úplný popis jazyka, všech funkcí a objektů je v příručce v dokumentu PP2CAN_skripty_prirucka_CZ_3_00.pdf.

Součástí instalace programu PP2CAN je i nová sada 50 ukázkových a převážně samo-testovacích skriptů.

Novinky v jazyce

Přetypování

Zápis (typ)výraz převede hodnotu na libovolný základní typ, i na menší. Dříve bylo nutné používat převodní funkce (i2c, u2c, truncd ...) a některé převody nešly vůbec.

```
c = (char)i;           // nejnizsi bajt: (char)300 = 44
i = (int)d;            // useknuti desetinne casti: (int)-3.9 = -3
l = (int64)u * 4;      // vypocet v 64 bitech bez pretečení
d = (double)a / b;     // deleni v double misto celociselného
b = (bool)x;           // true, je-li x ruzne od nuly
```

- Desetinné číslo převedené na celé se usekne; hodnota mimo rozsah cílového typu se omezí na jeho minimum nebo maximum, NaN dá 0.
- Přetypování má vyšší prioritu než aritmetika: (int)d * 2 znamená ((int)d) * 2.
- Na string lze přetypovat znak: (string)c.

Bitové operátory

Nové operátory & (AND), | (OR), ^ (XOR), ~ (negace), << a >> (posuny) pro všechny celočíselné typy. Mají prioritu jako v jazyce C a nahrazují funkce andi, ori, shl, shr, které zůstávají zachovány.

```
flags |= 0x04;          // nastavení bitu 2
flags &= ~0x01;         // nuloání bitu 0
if( (flags & 0x80) != 0 ) { ... }
value = (msb << 8) | lsb; // slození 16bitové hodnoty z bajtu
id29 = (prio << 26) | (pgn << 8) | source;
```

- Operand typu char se u ~, << a >> rozšíří na int, takže c << 8 neztratí horní bity.
- Posun vpravo u typů se znaménkem (int, int64) zachovává znaménko.

Složené přiřazení, ++ a --

```
counter++;              // take counter--, ++counter, --counter
sum += value;           // take -= *= /= %=
mask <<= 1;              // take &= |= ^= >>=
msg.size += 2;          // i pro proměnné objektu
```

```
text += "abc";           // i pro řetězce (text += c; přidava znak)
for( i = 0; i < 8; i++ ) { ... }
```

Výsledek složeného přiřazení se převede zpět na typ proměnné, proto funguje i `char c; c += 10;`

Konstanty a 64bitová čísla

- Hexadecimální konstanty `0x...` libovolné délky (dříve jen jeden bajt): `0x18FEF100`, `0xFFFFFFFFFFFFFFFF`.
- Alternativní zápisy `hex(ABC)`, binární `0b1010` a `bin(11110000)`.
- Typ hexadecimální a binární konstanty se určí podle velikosti hodnoty: do `0xFF` `char`, do `0xFFFFFFFF` `unsigned`, jinak `unsigned64`.
- Dekadická čísla nad 32 bitů jsou typu `unsigned64` (dříve se ořízla).
- Záporná čísla se zapisují přímo: `i = -1`; `d = -2.5`; (dříve `0-1` nebo `minusi/minusd`).
- Celočíselnou konstantu 0 až 255 lze přiřadit do `char` nebo předat jako parametr `char` bez přetypování: `c = 0`; `msg.fill(0);`.
- Typy `int64` a `unsigned64` mají správné automatické převody: `int` s `unsigned64` dává `int64`, `float` s 64bitovým číslem dává `double`.
- Nové vestavěné konstanty pro LIN, UDS a J1939 (`LIN_...`, `ISOTP_...`, `TP_OK`, `TP_ERR_...`).

Uživatelské funkce

Vedle sekcí `func_0` až `func_9` lze definovat plnohodnotné funkce s parametry, návratovou hodnotou a lokálními proměnnými. Funkce může být kdekoli v souboru a lze ji volat i před její definicí.

```
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;           // hodnota zustava mezi volanimi
    counter++;
    return counter;
end

body:
    d = average(1, 2) * 10;       // volani ve vyrazu
    divide(17, 5, q, r);          // q = 3, r = 2
end
```

- Parametry se předávají hodnotou; `ref` označuje výstupní parametr, jehož hodnota se po návratu zkopíruje zpět do proměnné volajícího.
- Lokální proměnné mají při každém volání hodnotu 0; `static` si hodnotu ponechá mezi voláními. Lokální jméno zakrývá globální proměnnou stejného jména.
- Funkce bez typu nebo s typem `void` nevrací hodnotu a volá se jako příkaz; `return;` ji ukončí.
- Funkce mohou volat další funkce až do hloubky 10. Rekursi interpret odhalí a skript zastaví s hlášením.
- Nová klíčová slova: `function`, `return`, `ref`, `static`, `void`.

Další změny jazyka

- Proměnné jsou při spuštění skriptu nulové, řetězce prázdné a logické proměnné `false` (dříve náhodný obsah).

- switch funguje s čísly, znaky i řetězci; za case lze uvést více hodnot oddělených čárkou.
- Celočíselné dělení nulou nevede k pádu PP2CAN: vypíše se hlášení Division by zero a výsledek je 0.
- Řetězce se při spojování bezpečně zkrátí na 1023 znaků, buffer funkcí log na 511 znaků.
- Přiřazení objektu do sebe sama nic nemění; vektor s více než 8 bajty přiřazený do CAN zprávy ji označí jako CAN FD.

LIN

Skripty mohou s převodníkem USB2LIN nastavovat LIN porty, zapisovat položky LIN tabulky a zpracovávat přijaté LIN zprávy.

- Nové typy skriptů LIN_FUNCTION a LIN_FUNCTION_AND_TIMER: sekce body se volá pro každou přijatou LIN zprávu (u AND_TIMER i periodicky).
- Funkce lin_config(port, režim, rychlost, verze) nastaví port 1 nebo 2 jako master, slave nebo scan, rychlost v Bd a verzi LIN 1.3 nebo 2.0.
- Funkce is_lin_function_call() rozliší volání přijatou LIN zprávou a časovačem.
- Objekt obj_lin popisuje LIN zprávu i položku tabulky: index, id, size, port, period, timeout, rx a data; metoda set() zapíše položku do převodníku, load() načte přijatou zprávu, rx_timeout() ohlásí vypršení odpovědi.

```
script:
  type = LIN_FUNCTION_AND_TIMER;
end
variables:
  bool ok;
end
objects:
  obj_lin item, rx;
end
init:
  ok = lin_config(1, LIN_MASTER, 19200, LIN_VER_20);
  item.port = 1;
  item.index = 0;
  item.id = 0x10;
  item.size = 8;
  item.period = 100;           // perioda vysilani hlavicky
  item.rx = true;              // odpoved posila slave
  item.set();
end
body:
  if( is_lin_function_call() ) {
    rx.load();
    logc_hex(rx.data(0));
    log_endl();
  }
end
```

Přenos dlouhých dat: UDS a J1939

Nové objekty dělí data delší než jedna CAN zpráva na rámce a z přijatých rámců data skládají. Časování (STmin, prodlevy BAM) a vyhodnocení řízení toku od protistrany ponechávají na skriptu, takže lze simulovat i nestandardní chování.

Objekt	Protokol
obj_uds	ISO-TP (ISO 15765-2) pro diagnostiku UDS: Single Frame, First Frame, Consecutive Frame, Flow Control; klasický CAN i CAN FD (rámce až 64 bajtů, data až 16 MB); normální, rozšířené a smíšené adresování; nastavitelná výplň.

Objekt	Protokol
obj_j1939	SAE J1939: samostatná zpráva, J1939-21 TP (BAM a RTS/CTS do 1785 bajtů), ETP (nad 1785 bajtů) a J1939-22 pro CAN FD (Multi-PG, FD.TP). Protokol se volí automaticky podle délky dat a cíle.

- Data se naplní řetězcem (set_string), po bajtech (set_data) nebo přiřazením z obj_vector_int.
- start() připraví rámce; next_frame(msg) nebo přiřazení msg = uds; vydá další rámec k odeslání.
- Po přijetí Flow Control nebo CTS skript povolí další rámce metodou flow(n), u J1939 i select(číslo paketu).
- Přijaté zprávy se předávají metodou receive(msg) nebo přiřazením uds = msg;. Na First Frame nebo RTS objekt sám připraví odpověď (Flow Control, CTS, EOMA) k odeslání.
- complete() hlásí přijetí všech dat, error() chyby přenosu (ztracený rámec, abort, příliš dlouhá data).

```
// tester: požadavek ReadDataByIdentifier VIN (0x22 F1 90)
uds.tx_id = 0x7E0;
uds.rx_id = 0x7E8;
uds.set_data(0, 0x22);
uds.set_data(1, 0xF1);
uds.set_data(2, 0x90);
uds.start();
while( uds.next_frame(msg) ) { msg.send(); }

// příjem odpovědi (CAN_FUNCTION)
msg.load();
if( uds.receive(msg) ) {
    while( uds.next_frame(msg) ) { msg.send(); } // Flow Control
    if( uds.complete() ) { vec = uds; vec.print_hex(); }
}
```

Nové a rozšířené funkce a objekty

- Objekty obj_uds, obj_j1939 a obj_lin (viz výše).
- Metody objektů mohou přebírat objekt jako parametr (např. uds.next_frame(msg)).
- Nové funkce lin_config, is_lin_function_call, get_bfi, mini64, minu64, maxi64, maxu64.
- obj_vector_int i obj_deque_int mají metody to_string a tostring (sjednocená jména).
- data_from_hex a přiřazení vektoru s více než 8 bajty do obj_can_msg nastaví fd = true.
- obj_cfgfile přijímá jméno souboru i bez cesty (vztahuje se k pracovnímu adresáři PP2CAN).
- Okna obj_datagraph a obj_datagauge přebírají nastavení ze skriptu bezpečně ve vlastním vlákně okna.

Překladač a dokumentace

- Chybová hlášení v češtině nebo angličtině: compiler infile -t|-b outfile [-cz|-en]. Hlášení uvádějí jména metod a čísla parametrů bez skrytých parametrů.
- Skript s novými vlastnostmi se ukládá v novém binárním formátu; starší verze PP2CAN jej odmítne načíst. Skripty bez nových vlastností zůstávají kompatibilní se staršími verzemi.
- Nová nápověda (HTML) a uživatelská příručka (docx) v češtině a angličtině.

Okno skriptu a editor

Editor

- Zvýraznění nových klíčových slov (function, return, ref, static, void, hex, bin, type); typy objektů obj_... a typy skriptů (CAN_FUNCTION, TIMER_SCRIPT, LIN_FUNCTION ...) mají vlastní barvu.

- Párování závorek: u kurzoru se zvýrazní odpovídající závorka (), { } nebo []; závorka bez protějšku se zobrazí červeně.
- Automatické odsazení: nový řádek převezme odsazení předchozího, za { , za záhlavím sekce (init:, body: ...) a za řádkem function se odsadí o úroveň; } a end se zarovnají automaticky.
- Zvýraznění všech výskytů slova pod kurzorem nebo vybraného slova, např. všech použití proměnné.
- Nabídka doplnění od tří napsaných znaků: klíčová slova, typy, všechny vestavěné funkce a konstanty a proměnné, objekty a funkce deklarované ve skriptu.
- Metody po tečce: po napsání jména objektu a tečky (msg.) nabídne metody a proměnné objektu podle jeho typu z deklarace v sekci objects.
- Nápopěda k parametrům: po napsání (za funkcí, metodou nebo uživatelskou funkcí se zobrazí její deklarace a zvýrazní se právě psaný parametr.

Překlad a spouštění

- Hlášení překladače se v okně OUTPUT vypisují po celých řádcích (dříve se mohla slít do jednoho řádku nebo roztrhnout). Script ready se zobrazí zeleně, Total errors s nenulovým počtem chyb červeně.
- Během běhu skriptu nelze skript nahrát (Load) ani přeložit (Compile); během překladu nelze skript ani spustit (Run). Běžící skript lze zastavit vždy.
- Při neúspěšném překladu se nespustí předchozí přeložená verze skriptu; Run ohlásí, že skript není přeložen.
- Překlad v jednom okně skriptu již neukončuje překlad probíhající v jiném okně.

Nápopěda a hledání

- Okno nápopědy se pro dané okno skriptu otevírá jen jednou; další stisk Help je obnoví a přenesení do popředí. Chování vůči oknu skriptu odpovídá nastavení z-order.
- Tlačítka Help a Find po otevření a zavření svého okna nezůstávají stisknutá.
- Zvýraznění výskytů slova neovlivňuje hledání v okně Find.

Změny ovlivňující existující skripty

Změna	Co upravit
obj_memory.set nemá návratovou hodnotu	Zápis <code>x = mem.set(i, v)</code> ; změnit na <code>mem.set(i, v)</code> ;
Opravený operátor <code>!=</code> pro char a bool	Skripty, které nevědomky spoléhaly na chybné chování, se nyní chovají správně.
Konstanty <code>0x...</code> a 64bitové konstanty	Konstanty delší než 1 bajt a nad 32 bitů mají nyní správnou hodnotu.
Operátory <code>&&</code> a <code> </code> mají stejnou prioritu	Beze změny proti předchozím verzím; při jejich kombinaci používejte závorky.

Opravy chyb

- Překladač: chybný překlad `!=` u char a bool, unárního minus, implicitních převodů na 64bitové typy, konstant `0x...` a velkých desetinných konstant; chybné deklarace několika funkcí a metod; zaseknutí na konci souboru bez nového řádku.
- Interpret: chybné 64bitové konstanty a násobení, chybné výsledky několika vestavěných funkcí (např. `shru64`, `b2str`, `str2d`, `roundd`, `maxc`), pády při dělení nulou a přepisy paměti u řetězců, bufferu log a mezních indexů.

- Objekty: pády a zaseknutí u kontejnerů (prázdný vektor, iterátory), obj_lin, obj_memory, obj_csvfile, obj_dataset, datových oken a socketového serveru; chybné hodnoty u obj_rs232 (CTS, DSR) a socketů (UDP).
- PP2CAN: zaseknutí nebo pád při zastavení běžícího skriptu a chyba při kopírování dat CAN FD.