Not eXactly C (NXC)
Príručka programátora

Manuál k verzii 1.0.1 b33

10. októbra, 2007

John Hansen

preklad: 3. augusta 2009

Pavel Petrovič
robotika.sk

Obsah

Zoznam tabuliek

1 Úvod

NXC znamená Not eXactly C, čiže „Nie celkom C“ a je to jednoduchý jazyk na programovanie stavebníc LEGO MINDSTORMS NXT. Programovateľná kocka NXT obsahuje tzv. bytecode interpreter (čiže interpréter bajtkódov) od firmy LEGO. Jeho úlohou je vykonávanie programov (čiže interpretácia bajtkódov, do ktorých sú programy preložené). Inými slovami: NXT rozumie programom vo formáte bajtkódov a nie vo formáte zdrojového textového súboru alebo naopak binárneho súboru so strojovým kódom. Prekladač (kompilátor) NXC prekladá zdrojový program do NXT bajtkódov, ktoré je možné následne spustiť na kocke NXT. Hoci syntax jazyka a jeho riadiace štruktúry sa veľmi podobajú na programovací jazyk C, NXT nie je univerzálny jazyk, obsahuje mnohé obmedzenia, vyplývajúce z obmedzení zabudovaného interprétera bajtkódov.

NXC logicky pozostáva z dvoch nezávislých celkov: 1) syntax jazyka NXC, ktorú treba dodržiavať pri písaní programov a 2) NXC API (Application Programming Interface), čiže systémové funkcie, konštanty, makrá, ktoré môžeme použiť v programoch. Celé API je definované v hlavičkovom súbore (header file), ktorý je automaticky zavedený pri kompilácii programu.

Tento dokument popisuje oba celky, čiže všetky informácie, ktoré potrebujeme na písanie programov NXC. Keďže programy v jazyku NXC možno vytvárať v rôznych vývojových prostrediach (napr. BricX Command Center, alebo kompilátor v príkazovom riadku), tento dokument sa nezaoberá žiadnym konkrétnym vývojovým prostredím. Tieto informácie nájdete v príslušnej dokumentácii (napr. *NXC User Manual*).

Aktualizovaná dokumentácia NXC je vždy k dispozícii na http://bricxcc.sourceforge.net/nxc/.

2 Jazyk NXC

V tejto kapitole opíšeme samotný jazyk NXC, čiže lexikálne pravidlá kompilátora, štruktúru programov, príkazy, výrazy a činnosť preprocesora (preprocesor je prvý priebeh kompilátora, v ktorom sa spracovávajú riadiace príkazy preprocesora, napr. #define).

Jazyk NXC je tzv. „case-sensitive“, čiže podobne ako C alebo C++ rozlišuje medzi malými a veľkými písmenami v názvoch premenných a funkcií. To znamená, že premenná *X* je rôzna od premennej *x*! Podobne, podmienku smieme zapísať pomocou „if“, ale nie pomocou „IF“ alebo „If“ - to už sú povolené identifikátory, napr. mená premenných.

2.1 Lexikálne pravidlá

Lexikálne pravidlá popisujú spôsob spracovania zdrojového súboru, ktorý sa rozkladá na jednotlivé tokeny (kľučové slová, identifikátory, operátory a pod.). Patrí sem spôsob písania poznámok/komentárov v zdrojových súboroch, spracovanie medzier, prípustné znaky a identifikátory.

2.1.1 Komentáre – poznámky v zdrojových súboroch

NXC podporuje dva druhy komentárov. Prvá forma (klasické C-čkové komentáre) začínajú `/*` a končia `*/`. Môžu obsiahnuť niekoľko riadkov, ale nemôžu byť vnorené:

```
/* malá poznámka */

/* dvojriadková poznánka
   pokračuje na ďalšom riadku */

/* tu je príklad vnoreného komentára...
           /* pokus o vnorený komentár...
              ukončenie vnoreného komentára...*/
  avšak tento text už nepatrí ani do vonkajšieho komentára! */
```

Druhá forma komentárov začína `//` a končí novým riadkom (klasické komentáre C++).

```
// jednoriadková poznámka
```

Kompilátor si poznámky nevšíma. Ich jediný zmysel je umožniť programátorovi popísať zdrojový súbor.

2.1.2 Medzery a „biele znaky“ (whitespace)

Biele znaky (medzery, tabulátory, nové riadky) sa používajú na formátovanie programu, najmä za účelom jeho lepšej čitateľnosti. NXC dovoľuje pridávať biele znaky v neobmedzenej miere – samozrejme, ak to nenaruší celistvosť jednotlivých tokenov (kľúčových slov, identifikátorov, operátorov). Biele znaky nemajú vplyv na činnosť programu. Príklad: nasledujúce dva riadky majú rovnaký význam:

```
x=2;
x = 2 ;
```

Niektoré operátory C++ pozostávajú s viacerých znakov, biele miesto nemožno vkladať medzi ne.

Príklad: prvý riadok je korektný výpočet (operátor bitového posunu vpravo), druhý riadok nemá zmysel:

```
x = 100 >> 4; // do x ulož hodnotu 100 vydelenú 16
x = 100 > > 4; // nezmysel
```

2.1.3 Numerické konštanty

Numerické konštanty môžu byť vyjadrené v desiatkovej alebo v šestnástkovej sústave. Desiatkové čísla pozostávajú z jednej alebo viacerých desiatkových číslic, hexadecimálne čísla začínajú znakmi `0x` alebo `0X` nasledovanými jednou alebo viacerými hexadecimálnymi číslicami.

```
x = 10; // nastav hodnotu x na 10
x = 0x10; // nastav x na 16 (10 hexa)
```

2.1.4 Identifikátory a kľúčové slová

Identifikátory sa používajú ako mená premenných, úloh (task-ov), funkcií alebo podprogramov (subroutine). Prvý znak identifikátora musí byť písmeno (malé alebo veľké) alebo podčiarkovník ('_'). Ostatné znaky môžu byť písmená, číslice, alebo podčiarkovníky.

Mnohé identifikátory sú v jazyku NXC vyhradené (rezervované kľúčové slová) a nesmú byť použité ako identifikátory. Úplný zoznam klúčových slov nasleduje:

__RETURN__	char	long	sub
__RETVAL__	const	mutex	switch
__STRRETVAL__	continue	priority	task
__TMPBYTE__	default	repeat	true
__TMPWORD__	do	return	typedef
__TMPLONG__	else	safecall	unsigned
abs	false	short	until
asm	for	sign	void
bool	goto	start	while
break	if	stop	
byte	inline	string	
case	int	struct	

Tabuľka 1. Kľúčové slová NXC

2.2 Štruktúra programov

Program v jazyku NXC pozostáva z blokov kódu a premenných. Existujú dva druhy blokov kódu: úlohy (task-y) a funkcie. Oba typy majú svoje špecifické použitie, ale zdieľajú rovnakú štruktúru.

2.2.1 Úlohy (task-y)

NXT podporuje paralelný výpočet viacerých vlákien (multi-threading) a jedna úloha v NXC priamo zodpvoedá jednému výpočtovému vláknu NXT. Úloha sa zadefinujne pomocou kľúčového slova `task` s nasledujúcou syntaxou:

```
task name()
{
  // telo úlohy: príkazy pre túto úlohu
}
```

Ako meno úlohy môže byť použitý ľubovoľný prípustný identifikátor. Každý program musí obsahovať aspoň jednu úlohu, ktorá je pomenovaná „main“ a ktorá je spustená pri štarte programu. Program môže obsahovať najviac 256 úloh.

Telo úlohy pozostáva zo zoznamu príkazov. Jednotlivé úlohy je možné plánovať pomocou API funkcie `Precedes` alebo `Follows`. Primárny mechanizmus na štartovanie paralelných úloh je API funkcia `Follows`. Úlohy môžu byť odštartované a zastavené pomocou príkazov `start` a `stop`. Existuje aj príkaz `StopAllTasks`, ktorý zastaví všetky paralelne bežiace úlohy, ale všetky úlohy sa dajú zastaviť aj pomocou funkcie `Stop`. Úloha môže ukončiť svoju činnosť pomocou funkcie `ExitTo` alebo jednoducho tak, že tok výpočtu dosiahne koniec tela úlohy.

2.2.2 Funkcie

Pri programovaní často potrebujeme spojiť viacero príkazov do jednej funkcie, ktorú môžeme použiť na viacerých miestach, všade kde potrebujeme. NXC podporuje funkcie s argumentami a návratovými hodnotami. Funkcie sú definované v nasledujúcej syntaxi:

```
[safecall] [inline] typ_návratovej_hodnoty meno(zoznam_argumentov)
{
 // telo funkcie
}
```

Funkcia môže vracať hodnoty iba takého typu ako vyjadruje hlavička funkcie (typ_návratovej_hodnoty). Ak funkcia nevracia žiadnu hodnotu, ako návratový typ zapíšeme `void`.

Zoznam argumentov môže byť prázdny, alebo definuje jeden alebo viacero argumentov. Každý argument je určený *typom* ktorý je nasledovaný jeho *menom*. Jednotlivé argumenty sú oddelené čiarkami. Dovolené typy sú: *bool*, *char*, *byte*, *int*, *short*, *long*, *unsigned int*, *unsigned long*, *string*, štruktúrované typy (*struct*), alebo polia ľubovoľného typu. NXC podporuje odovzdávanie argumentov hodnotou, konštantnou hodnotou, odkazom alebo odkazom na konštantu.

Pri odovzdávaní argumentov hodnotou alokuje NXC novú dočasnú premennú do ktorej sa hodnota skopíruje. Typ premennej nie je nijak obmedzený. Pochopiteľne, všetky zmeny hodnoty tejto premennej vnútri funkcie nebudú mať žiadny vplyv na premennú mimo funkcie, ktorá odovzdala hodnotu cez argument funkcie. Príklad: funkcia `foo` zmení hodnotu argumentu na 2. To je samozrejme dovolené, ale nemá to žiadny vplyv na premennú `y` v hlavnej úlohe.

```
void foo(int x)
{
  x = 2;
}

task main()
{
  int y = 1; // y is now equal to 1
  foo(y); // y is still equal to 1!
}
```

Druhý typ argumentu, `const` *`arg_type`*, je tiež odovzdávaný hodnotou, ale obmedzuje hodnoty odovzdané cez argument len na konštantné hodnoty (t.j. čísla). Takýchto funkcii je v NXT viacero, preto je dôležité uvedomiť si, že dovolené sú iba číselné konštanty.

```
void foo(const int x)
{
  PlaySound(x); // ok
  x = 1; // chyba – nemožno zmeniť argument
}

task main()
{
  foo(2); // ok
  foo(4*5); // ok – výsledkom výrazu je konštanta
  foo(x); // chyba - x nie je konštanta
}
```

Tretí druh argumentu, *typ_argumentu* `&`, odovzdáva argumenty odkazom a nie hodnotou. Vďaka tomu môže volaná funkcia zmeniť hodnoty premennej vo funkcii alebo úlohe, ktorá ju zavolala. Na druhej strane, volanie odkazom vyžaduje, aby volajúca funkcia alebo úloha použila v argumente premennú, konštanty nie sú dovolené:

```
void foo(int &x)
{
  x = 2;
}

task main()
{
  int y = 1; // priraď 1 do y
  foo(y); // y bude po návrate rovné 2
  foo(2); // chyba – argument volaný odkazom povoľuje iba premenné
}
```

Štvrtý druh argumentu, `const typ_argumentu &`, je neobvyklý. Jedná sa tiež o volanie odkazom, avšak volaná funkcia nesmie zmeniť hodnotu argumentu. Vďaka tomuto obmedzeniu dovoľuje kompilátor odovzdať v argumente aj konštantné hodnoty, nielen premenné. Vo všeobecnosti je toto najefektívnejší spôsob odovzdávania argumentov v NXC.

Funkcie musia byť volané so správnym počtom a typmi argumentov. Nasledujúci príklad ukazuje niekoľko správnych a chybných volaní funkcie `foo`:

```
void foo(int bar, const int baz)
{
  // telo funkcie...
}

task main()
{
  int x; // declarácia premennej x
  foo(1, 2); // ok
  foo(x, 2); // ok
  foo(2, x); // chyba – druhý argument nie je konštanta!
  foo(2); // chyba – zlý počet argumentov!
}
```

Funkcie v jazyku C++ môžu byť bežné alebo tzv. inline. Inline funkcie sa prekladajú ako makrá, čo znamená, že na mieste, odkiaľ sa funkcia volá, sa vloží kópia celého kódu funkcie, namiesto jednej

inštrukcie volania funkcie, ako je to u bežných funkcií. Ak je funkcia krátka, môžeme si dovoliť označiť ju ako inline a kód sa bude vykonáva rýchlejšie, lebo ušetríme celú agendu okolo volania a návratu z funkcie. Ak začneme označovať ako inline aj dlhšie funkcie, náš program môže veľmi ľahko priveľmi narásť.

Podobne je to aj v jazyku NXC: existujú bežné funkcie (ktoré sa prekladajú do podprogramov NXT, tzv. NXT subroutines) a inline funkcie. Počet bežných funkcií je obmedzený – môže ich byť najviac 256.

Jazyk NXC podporuje paralelizmus (súčasný beh viacerých výpočtových vlákien). Niekedy potrebujeme, aby príslušnú funkciu mohlo naraz vykonávať iba jedno vlákno. Ak už niektoré vlákno funkciu zavolalo a stále ju vykonáva, druhé vlákno, ktoré tiež funkciu zavolalo, musí počkať, kým prvé vlákno funkciu neopustí, až potom môže druhé vlákno začať vykonávať príslušnú funkciu. To samozrejme neplatí u bežných funkcií, ale ak funkciu označíme kľúčovým slovom safecall, tak bude prístup viacerých vlákien takto synchronizovaný (interne zabezpečený pomocou volaní Acquire a Release).

2.2.3 Premenné

Všetký premenné v NXC majú jeden z nasledujúcich typov:

Meno typu	**Popis**
bool	8 bit unsigned
byte, unsigned char	8 bit unsigned
char	8 bit signed
unsigned int	16 bit unsigned
short, int	16 bit signed
unsigned long	32 bit unsigned
long	32 bit signed
mutex	Special type used for exclusive code access
string	Array of byte
struct	User-defined structure types
Arrays	Arrays of any type

Tabuľka 2. Typy premenných

Premenné deklarujeme takto: kľúčové slovo zodpovedajúce menu príslušného typu je nasledované zoznamom mien premenných, ktoré sú oddelené čiarkami, deklarácia je ukončená bodkočiarkou. Premenné pri deklarácii môžeme inicializovať na nejakú hodnotu pomocou operátora priradenia za menom premennej. Príklady deklarácii:

```
int x; // deklarácia premennej x
bool y,z; // deklarácia premenných y a z
long b=1,c; // deklarácia premenných b a c, inicializácia b na 1
```

Globálne premenné sú deklarované mimo blokov kódu (von z funkcie alebo úlohy). Tieto premenné môžu byť použité vo všetkých úlohách, funkciách a podprogramoch od miesta deklarácie po koniec súboru.

Lokálne premenné môžu byť deklarované vnútri úloh a funkcií. Sú prístupné len vrámci daného bloku kódu, kde sú definované. Okrem funkcií a úloh môžu byť lokálne premenné deklarované aj vnútri zloženého príkazu (skupina príkazov medzi zlozženými zátvorkami), vtedy sú viditeľné iba vrámci daného zloženého príkazu:

```
int x; // x je globálna premenná

task main()
{
  int y; // y je lokálna premenná v úlohe main
  x = y; // ok
  { // začiatok zloženého príkazu
    int z; // deklarácia lokálnej premennej z
    y = z; // ok
  }
  y = z; // chyba - z nie je viditeľná
}

task foo()
{
  x = 1; // ok
  y = 2; // chyba - y nie je globálna
}
```

2.2.4 Štruktúrovaný typ (struct)

NXC podporuje užívateľom definované zložené typy, ktoré sa budujú pomocou kľúčového slova struct. Sú podobné zloženým struct typom jazyka C.

```
struct car
{
  string car_type;
  int manu_year;
};

struct person
{
  string name;
  int age;
  car vehicle;
};

myType fred = 23;
person myPerson;
```

Po zadefinovaní štruktúrovaného typu môžeme deklarovať premennú, alebo prvok vrámci iného štruktúrovaného typu. Prvky štruktúrovaného typu sú prístupné cez bodkovú notáciu.

```
myPerson.age = 40;

anotherPerson = myPerson;

fooBar.car_type = "honda";
fooBar.manu_year = anotherPerson.age;
```

Je možné priradiť hodnotu štruktúrovanej premennej inej štruktúrovanej premennej, ak sú rovnakého typu.

2.2.5 Polia

NXC pozná polia. Polia sa deklarujú rovnakým spôsobom ako bežné premenné, pričom meno premennej je nasledované otváracou a zatváracou hranatou zátvorkou:

```
int my_array[]; // deklarácia poľa s 0 prvkami
```

Viacrozmerné polia sa deklarujú tak, že zaradíme viacero párov hranatých zátvoriek. V NXC môžeme deklarovať najviac 4-rozmerné polia.

```
bool my_array[][]; // deklarácia 2-rozmerného poľa
```

Globálne jednorozmerné polia môžu byť inicializované na riadku, kde sú deklarované takto:

```
int X[] = {1, 2, 3, 4}, Y[]={10, 10}; // 2 polia
```

Prvky polí sú prístupné pomocou ich pozície – indexu v poli. Prvý prvok má index 0, druhý má index 1 atď. Napríklad:

```
my_array[0] = 123; // nastav prvý prvok poľa na 123
my_array[1] = my_array[2]; // skopíruj tretí prvok do druhého
```

V súčasnosti existujú určité obmedzenia pri práci s poľami, ktoré budú pravdepodobne odstránené v ďalších verziách NXC. Inicializácia lokálnych polí, alebo viacrozmerných polí sa robí pomocou funkcie ArrayInit. V nasledujúcom príklade je initicializované dvojrozmerné pole. Príklad tiež uvádza použitie niektorých funkcií API na prácu s poľami:

```
task main()
{
  int myArray[][];
  int myVector[];
  byte fooArray[][][];
  ArrayInit(myVector, 0, 10); // 10 núl do poľa myVector
  ArrayInit(myArray, myVector, 10); // inic. myArray 10 vektormi
  ArrayInit(fooArray, myArray, 2); // 2 polia myArray sú fooArray
  myVector = myArray[1]; // ok od verzie NXC b25
  fooArray[1] = myArray; // ok od verzie NXC b25
  myVector[4] = 34;
  myArray[1] = myVector; // ok od verzie NXC b25
  int ax[], ay[];
  ArrayBuild(ax, 5, 6);
  ArrayBuild(ay, 2, 10, 6, 43);
  int axlen = ArrayLen(ax);
  ArraySubset(ax, ay, 1, 2); // ax = {10, 6}
  if (ax == ay) { // porovnávanie polí funguje od verzie NXC b25
  }
}
```

NXC dovoľuje stanovenie začiatočnej veľkosti pre globálne a lokálne polia. Kompilátor automaticky generuje kód, ktorý takto deklarované polia inicializuje nulovými hodnotami. Ak deklarácia globálneho poľa obsahuje súčasne určenie veľkosti aj inicializujúce hodnoty, kompilátor si určenie veľkosti nevšíma.

```
task main()
{
  int myArray[10][10];
  int myVector[10];
  // ArrayInit(myVector, 0, 10); // 10 núl do poľa myVector
  // ArrayInit(myArray, myVector, 10); //pole myArray z 10 vektorov

/*
 Volania ArrayInit už nie sú potrebné, pretože sme
 určili počiatočné veľkosti polí. Dokonca, ak pole myVector
 používame iba na inicializáciu poľa myArray, môžeme ho
 celkom vynechať.
*/
}
```

2.3 Príkazy

Telo bloku kódu (úlohy alebo funkcie) pozostáva z príkazov. Príkazy sú ukončené bodkočiarkou (';').

2.3.1 Deklarácia premenných

Deklarácia premenných, popísaná v predchádzajúcej časti, je jedným z rôznych typov príkazov. Deklaruje lokálnu premennú (a prípadne ju inicializuje), ktorá je použiteľná vrámci aktuálneho bloku kódu. Syntax deklarácie premennej:

```
int premenné;
```

pričom premenné je čiarkami oddelovaný zoznam mien prenných, ktoré prípadne môžu mať určené počiatočné hodnoty:

```
meno[=výraz]
```

Deklarácia polí premenných:

```
int pole[n][= počiatočné hodnoty globálneho jednorozmerného poľa];
```

2.3.2 Priradenie

Premenným, ktoré boli deklarované, môžeme priraďovať hodnoty výrazov:

```
premenná operátor_priradenia výraz;
```

Existuje deväť rôznych operátorov priradenia. Najjednoduchší operátor je '=', ktorý premennej jednoducho priradí hodnotu výrazu. Ostatné operátory menia hodnotu premennej rôznym spôsobom, ako ukazuje nasledujúca tabuľka.

Príklady:

```
x = 2; // nastav x na 2
y = 7; // nastav y na 7
x += y; // x bude 9, y bude stále 7
```

Operátor	Akcia
=	Nastavenie premennej na danú hodnotu
+=	Pripočítanie hodnoty k premennej
–=	Odpočítanie hodnoty od premennej
*=	Vynásobenie premennej zadanou hodnotou
/=	Vydelenie premennej hodnotou
%=	Nastavenie premennú na zvyšok po delení jej hodnoty zadanou hodnotou
&=	Bitová operácia logického súčinu (AND) medzi premennou a zadanou hodnotou
|=	Bitová operácia logického súčtu (OR)
^=	Bitová operácia nerovnoznačnosti (XOR)
||=	Nastavenie premennej na absolútnu hodnotu zadaného výrazu
+–=	Nastavenie premennej na znamienko (-1,+1,0) výrazu
>>=	Bitový posun hodnoty premennej o zadaný počet bitov vpravo
<<=	Bitový posun vľavo

Tabuľka 3. Operátory

2.3.3 Riadiace štruktúry

Najjednoduchšia riadiaca štruktúra je zložený príkaz. Tvorí ho zoznam príkazov, ktorý je uzavretý v zložených zátvorkách ('`{`' and '`}`'):

```
{
  x = 1;
  y = 2;
}
```

Hoci sa nezdá, že by tento príkaz mal nejaký význam, plní dôležitú úlohu ako súčasť zložitejších riadiacich štruktúr. Mnohé riadiace štruktúry obsahujú telo, ktoré pozostáva len z jediného príkazu. Použitím zloženého príkazu môže príslušná štruktúra riadiť naraz niekoľko po sebe idúcich príkazov.

Príkaz `if` zodpovedá podmienke. Ak je podmienka splnená, vykoná sa prvá vetva príkazu (dôsledok). Podmienka môže (ale nemusí) obsahovať aj druhú vetvu (alternatívu), ktorá sa vykoná, ak podmienka nie je splnená (jej pravdivostná hodnota je nepravda/false). Podmienka teda syntakticky v jednotlivých formách vyzerá takto:

```
if (podmienka) dôsledok

if (podmienka) dôsledok else alternatíva
```

Všimnite si, že podmienka je uzavretá v zátvorkách. Príklady sú uvedené nižšie. Všimnite si, že zložený príkaz v poslednom príklade umožňuje vykonanie dvoch príkazov v časti dôsledku podmienky.

```
if (x==1) y = 2;
if (x==1) y = 3; else y = 4;
if (x==1) { y = 1; z = 2; }
```

Príkaz cyklu `while` sa používa na zápis cyklov s podmienkou opakovania. Najskôr sa vyhodnotí podmienka. Ak je splnená, vykoná sa telo cyklu a takto sa pokračuje ďalej – znovu sa vyhodnocuje podmienka. Keď je testovaná podmienka nepravdivá, alebo sa v tele vykoná príkaz `break`, opakovanie cyklu sa ukončí. Syntax cyklu `while` má tento tvar:

```
while (podmienka) telo_cyklu
```

Telo cyklu väčšinou tvorí zložený príkaz, ako ukazuje nasledujúci príklad:

```
while(x < 10)
{
  x = x+1;
  y = y*2;
}
```

Variantom cyklu `while` je cyklus `do-while`. Jeho syntax je:

```
do telo while (podmienka)
```

Rozdiel medzi cyklami `while` a `do-while` spočíva v tom, že cyklus `do-while` vykoná svoje telo cyklu vždy aspoň jeden raz (podmienka sa testuje prvýkrát až po prvom vykonaní tela cyklu), zakiaľ čo telo cyklu `while` nemusí byť vykonané ani raz.

Iný druh cyklu je cyklus `for`:

```
for (príkaz1 ; podmienka ; príkaz2) telo_cyklu
```

Príkaz `for` najskôr jeden raz vykoná príkaz1, a potom opakovane kontroluje platnosť podmienky a pokiaľ je splnená, vykoná telo cyklu a následne príkaz2. Príkaz `for` je ekvivalentný postupnosti:

```
príkaz1;
while(podmienka)
{
  telo_cyklu
  príkaz2;
}
```

Príkaz cyklu `repeat` zopakuje vykonanie tela cyklu stanovený počet opakovaní:

```
repeat (číselný_výraz) telo_cyklu
```

Číselný výraz v zátvorkách určuje koľkokrát sa telo cyklu vykoná. Poznámka: číselný výraz sa vyhodnocuje vždy iba raz, čo však nie je pravidlo u príkazov `while` a `do-while`, ktoré svoje podmienky vyhodnocujú pri každom opakovaní cyklu.

Príkaz `switch` sa používa na vykonanie jedného z viacerých alternatívnych príkazových blokov (postupností príkazov) v závislosti od hodnoty výrazu. Každý príkazový blok začína jedným alebo viacerými návestiami `case`. Každé návestie je určené konštantou, ktorá sa vrámci celého príkazu switch neopakuje viackrát. Príkaz switch najskôr vyhodnotí výraz v zátvorkách a potom hľadá návestie označené hodnotou, ktorá sa rovná výsledku výrazu. Ak sa návestie nájde, vykoná všetky príkazy, ktoré nasledujú za týmto návestím až po koniec príkazu switch, alebo po príkaz `break`. V príkaze switch sa môže vyskytnúť jedno návestie `default`, ktoré vyhovuje akejkoľvek hodnote výrazu, ktorá sa nevyskytuje v inom návestí. Príkaz switch má takúto syntax:

```
switch (výraz) telo
```

Návestia (case a default) nie sú považované za samostatné príkazy, sú iba návestiami a sú uvedené pred príkazmi, ktoré označujú. Jeden príkaz môže byť označený aj viacerými návestiami. Návestia vnútri príkazu switch majú nasledovnú syntax:

```
case konštantný_výraz :
default :
```

V typickom prípade vyzerá príkaz switch:

```
switch(x)
{
      case 1:
            // príkazy ktoré sa vykonajú ak x je rovné 1
            break;
      case 2:
      case 3:
            // nejaké iné príkazy, vykonajú sa, ak x je 2 alebo 3
            break;
      default:
            // príkazy, ktoré sa vykonajú keď x nie je 1, 2, ani 3
            break;
}
```

NXC dovoluje vo výraze switch použiť aj výrazy, ktoré sa vyhodnotia na znakový reťazec a podobne možno použiť reťazcové konštanty v návestiach case.

Príkaz `goto` je príkazom skoku – vykonávanie programu pokračuje ďalej od zadaného miesta. Príkazy v programe môžu byť označené bežnými návestiami, ktoré pozostávajú z identifikátora nasledovaného dvojbodkou. Príkaz goto obsahuje identifikátor z návestia príkazu, kam má program odskočiť. Nasledujúci príklad je nekonečný cyklus, ktorý neustále zvyšuje hodnotu premennej x:

```
nekonecny_cyklus:
      x++;
      goto nekonecny_cyklus;
```

Príkaz `goto` treba využívať len výnimočne a veľmi opatrne. Takmer v každej situácii je možné použiť riadiace štruktúry ako napr. `if`, `while`, a `switch`, ktoré robia program oveľa čitateľnejším. Program s príkazmi goto sa veľmi zle mení a udržuje.

Jazyk NXC obsahuje aj makro `until`, ktoré je len skratkou na zápis príkazu `while` s podmienkou ukončenia. Toto makro je definované takto:

```
#define until(c) while(!(c))
```

Inými slovami, `until` opakuje telo cyklu a opakovanie ukončí, keď je podmienka splnená. Najčastejšie sa používa v kombinácii s prázdnym príkazom ako jeho telom (samotná bodkočiarka tvorí tzv. prázdny príkaz):

```
until (SENSOR_1 == 1); // čakaj na stlačenie senzora
```

2.3.4 Príkaz asm

Príkaz `asm` sa používa hlavne na definovanie volaní NXC API (čiže rôznych užitočných obslužných funkcií pre senzory, motory a pod., ktoré môžeme vo svojich programoch využiť a ktoré sú naprogramované v nízkoúrovňovom jazyku bytekódov NXT). Syntax príkazu je:

```
asm {
 jeden alebo niekoľko riadkov v jazyku bytekódov
}
```

Tento príkaz umožňuje do programu zaradiť časť programu, ktorá je napísaná v jazyku NeXT Byte Codes (NBC). Je možné ho využiť na generovanie optimálnejšieho kódu, ktorý má firmware NXT interpretovať najrýchlejšie ako sa dá. V nasledujúcom príklade použitia príkazu asm sú použité deklarácie premenných, návestia, inštrukcie NBC a komentáre:

```
asm {
//    jmp __lbl00D5
      dseg segment
        sl0000 slong
        sl0005 slong
        bGTTrue byte
      dseg ends
      mov sl0000, 0x0
      mov sl0005, sl0000
      mov sl0000, 0x1
      cmp GT, bGTTrue, sl0005, sl0000
      set bGTTrue, FALSE
      brtst EQ, __lbl00D5, bGTTrue
   __lbl00D5:
}
```

Niektoré kľúčové slová NXC majú význam iba vnútri príkazov asm. Tieto kľúčové slová umožňujú vrátiť reťazec znakov alebo číselné hodnoty a využívanie dočasných celočíselných premenných typu byte, word a long.

Kľúčové slovo ASM	Význam
__RETURN__	Využíva sa na vrátenie hodnoty inej ako __RETVAL__ alebo __STRRETVAL__
__RETVAL__	Hodnota zapísaná do tejto 4-bajtovej premennej bude vrátená volajúcemu programu
__STRRETVAL__	Hodnota zapísaná do tejto reťazcovej premennej bude vrátená volajúcemu programu
__TMPBYTE__	Dočasná premenná pre jednobajtové hodnoty
__TMPWORD__	Dočasná premenná pre dvojbajtové hodnoty
__TMPLONG__	Dočasná premenná pre štvorbajtové hodnoty

Tabuľka 4. Kľúčové slová ASM

Príkazový blok asm a tieto špeciálne kľúčové slová sú využité v celom API NXC. V hlavičkovom súbore NXCDefs.h nájdete niekoľko príkladov použitia.

Kvôli udržateľnosti prehľadnosti kódu a dosiahnutiu väčšej podobnosti s jazykom C, možno bloky asm zabaliť do makier preprocesora a zaradiť do vlastných hlavičkových súborov, ktoré sú do programu vložené pomocou príkazu #include. V nasledujúcom príklade sú použité makrá na obalenie príkazového bloku asm:

```
#define SetMotorSpeed(port, cc, thresh, fast, slow) \
  asm { \
  set theSpeed, fast \
  brcmp cc, EndIfOut__I__, SV, thresh \
  set theSpeed, slow \
EndIfOut__I__: \
  OnFwd(port, theSpeed) \
  __IncI__ \
}
```

2.3.5 Iné príkazy

Volanie funkcie je príkaz v tvare:

```
meno(argumenty);
```

Zoznam argumentov je zoznam výrazov, ktoré oddelované čiarkami. Počet a typ argumentov musí sedieť s definíciou funkcie.

Príkazy na odštartovanie alebo zastavenie úloh (tasks) sú:

```
start meno_úlohy;
stop meno_úlohy;
```

Priorita úlohy sa dá zmeniť pomocou príkazu:

```
priority meno_úlohy, nová_priorita;
```

Ak sa v tele nejakého cyklu (napríklad `while`) vyskytne príkaz `break`, vykonávanie tela cyklu a jeh ďalšie opakovanie nezávisle na podmienke sa ukončí. Ak sa v tele vyskytne príkaz `continue`, vykonávanie tela cyklu sa preruší a pokračuje sa jeho ďalším opakovaním, ak podmienka opakovania je splnená. Príkaz break slúži aj na vyskočenie z tela príkazu switch.

```
break;
continue;
```

Na ukončenie vykonávania tela funkcie slúži príkaz `return`. Pri jeho vykonaní funkcia vráti hodnotu výrazu, ak je uvedený.

```
return [výraz];
```

Výrazy vo všeobecnosti nemôžu byť použité samostatne ako príkazy. Významnou výnimkou sú výrazy s operátormi na zvýšenie o 1 (inkrementovanie, ++) a zníženie o 1 (dekrementovanie, `--`).

```
x++;
```

Prázdny príkaz (holá voľne stojaca bodkočiarka) je tiež povolený príkaz.

2.4 Výrazy

Hodnoty sú najbežnejšia forma výrazov. Zložené výrazy sú kombináciou hodnôt a operátorov. Jazyk NXC pozná len dva základné druhy hodnôt: číselné konštanty a premenné. Číselné konštanty sú v kocke NXT reprezentované ako celé čísla. Konkrétny typ závisí od hodnoty konštanty. Interne sú v programoch NXC matematické výpočty vykonávané na 32-bitových číslach so znamienkom. Číselné konštanty sa v programoch môžu zapísať ako desiatkové (napr. `123`) alebo šestnáskové (e.g. `0XABC`) hodnoty. V aktuálnej verzii sa konštanty nekontrolujú na pretečenie cez rozsah a preto výpočty, ktoré počítajú s hodnotami mimo rozsah môžu dávať neočakávané výsledky.

Dve preddefinované hodnoty majú osobitný význam: `true` a `false`. Hodnota `false` je nula (0),

a hodnota `true` je jeden (1). Tieto hodnoty sú aj výsledkom vyhodnotenia relačných operátorov (napr. <): ak porovnanie neplatí, výsledkom je 0, inak je výsledkom hodnota 1.

Niektoré operátory môžu byť kombinované iba s konštantnými hodnotami alebo výrazmi, ktoré obsahujú iba konštantné hodnoty. Nasledujúca tabuľka obsahuje zoznam operátorov v poradí ich priority (od operátorov s najväčšou prioritou, ktoré sa vyhodnocujú ako prvé po operátory s najmenšou prioritou):

Operátor	Popis	Associativita	Obmedzenie	Príklad
abs()	Absolutna hodnota	-	-	abs(x)
sign()	Znamienko operandu	-	-	sign(x)
++, --	Post inkrement, Post dekrement	zľava	iba vpravo od premmennej	x++
-	Unárne mínus	sprava	-	-x
~	Bitová negácia	sprava	-	~123
!	Logická negácia	sprava	-	!x
*, /, %	Násobenie, delenie, zvyšok	zľava	-	x * y
+, -	Sčítanie, odčítanie	zľava	-	x + y
<< , >>	Ľavý a pravý bitový posun	zľava	-	x << 4
< , > , <= , >=	relačné operátory	zľava	-	x < y
==, !=	Rovná sa, nerovná sa	zľava	-	x == 1
&	Bitový logický súčin	zľava	-	x & y
^	Bitová nerovnoznačnosť	zľava	-	x ^ y
|	Bitový logický súčet	zľava	-	x | y
&&	Logický súčin logických hodnôt	zľava	-	x && y
||	Logický súčet logických hodnôt	zľava	-	x || y
? :	podmienka ako výraz	-	-	x==1 ? y : z

Tabuľka 5. Priorita a poradie vyhodnotenia operátorov

Poradie vyhodnocovania výrazov možno v prípade potreby zmeniť pomocou zátvoriek:

```
x = 2 + 3 * 4; // nastaví x na 14
y = (2 + 3) * 4; // nastaví y na 20
```

2.4.1 Podmienky

Porovnanie dvoch hodnôt tvorí podmienku. Porovnanie je pravdivé alebo nepravdivé. V podmienke môžeme tieto logické hodnoty - `true` a `false` – použiť aj priamo. Pravdivosť podmienkového logické výrazu možno obrátiť operátorom logickej negácie a jednotlivé logické výrazy sa môžu kombinovať pomocou operátorov logického súčinu a súčtu. Ttabuľka č.6 zhŕňa rôzne druhy podmienok.

2.5 Preprocesor

Preprocesor je časť NXC, ktorá sa automaticky spustí vždy pred tým ako sa kompiluje program v NXC. Jeho účelom je umožniť rozdelenie jedného súboru na viacero častí (direktíva include) a kompilovanie toho istého programu rôznym spôsobom podľa nastavenia kompilačných premenných. Preprocesor rozpoznáva nasledujúce príkazy: `#include`, `#define`, `#ifdef`, `#ifndef`, `#endif`, `#if`, `#elif`, `#undef`, `##`, `#line`, a `#pragma`. Jeho činnosť je analogická klasickému preprocesoru jazyka C, takže väčšina vecí, ktorá funguje v C-čku by mala fungovať aj v NXC. Nasledujúce state popisujú podstatné rozdiely:

Podmienka	Význam
`true`	vždy splnená
`false`	nikdy nie je splnená
`Expr`	splnená, ak výraz nie je rovný 0
`Expr1 == expr2`	splnená, ak sa dva výrazy rovnajú
`Expr1 != expr2`	splnená, ak sa dva výrazy nerovnajú
`Expr1 < expr2`	splnená, ak výraz expr1 je menší ako expr2
`Expr1 <= expr2`	splnená, ak výraz expr1 je menší alebo rovný expr2
`Expr1 > expr2`	splnená, ak výraz expr1 je väčší ako expr2
`Expr1 >= expr2`	splnená, ak výraz expr1 je väčší alebo rovný expr2
`! condition`	splnená ak uvedená podmienka je nesplnená (negácia)
`Cond1 && cond2`	splnená práve vtedy, keď obidve podmienky sú splnené
`Cond1 || cond2`	splnená práve vtedy, keď aspoň jedna podmienka je splnená

Tabuľka 6. Podmienky

2.5.1 #include

Príkaz `#include` robí, to čo možno očakávať, s tým, že meno súboru musí byť vždy medzi úvodzovkami, neexistujú štandardné adresáre pre include ako v jazyku C a preto súbor nemôže byť uvedený v ostrých zátvorkách.

```
#include "foo.h" // ok
#include <foo.h> // chyba!
```

Programy v NXC môžu, ale nemusia začínať riadkom `#include "NXCDefs.h"`. Tento štandardný hlavičkový súbor definuje množstvo dôležitých konštánt a makier, ktoré tvoria základ NXC API. Novšie verzie NXC už nevyžadujú, aby sa tento súbor inlude-oval ručne. Kompilátor ho natiahne automaticky, ak ho zvlášť nepožiadate, aby tento štandardný hlavičkový súbor nenatiahol.

2.5.2 #define

Príkaz `#define` sa používa na jednoduché nahradzovanie v zdrojovom texte programu (makro). Makro nie je možné predefinovať. Koniec riadku makro automaticky ukončuje, ale ak riadok končí opačným lomítkom ('\'), makro pokračuje na ďalšom riadku:

```
#define foo(x) do { bar(x); \
                    baz(x); } while(false)
```

Príkaz `#undef` zruší definíciu makra.

2.5.3 ## (Zreťazenie)

Príkaz `##` funguje podobne ako v preprocesore jazyka C. Je nahradený prázdnym reťazcom, takže výrazy naľavo a napravo od tohto operátora sú zreťazené. Môže byť použitý v definíciách makier na vytvorenie identifikátorov kombináciou viacerých parametrov makra.

2.5.4 Podmienená kompilácia

Funguje podobne ako v jazyku C a využíva nasledujúce príkazy:

```
#ifdef symbol
#ifndef symbol
#else
#endif
#if condition
#elif
```

3 NXC API

NXC API definuje konštanty, procedúry, operácie a makrá na prístup k jednotlivým funkciám NXT, senzorom, výstupom, komunikácii. Volania API sú jednak procedúry (príkazy, v anglickej verzii manuálu sú označené ako „Function“), operácie (čiže vlastne funkcie, ktoré vracajú nejakú hodnotu, v anglickej verzii manuálu sú označené ako „Value“). Konštánty – čiže preddefinované hodnoty sú tiež súčasťou API. Procedúry vykonávajú nejakú činnosť, alebo nastavujú nejaký parameter. Operácie môžu byť použité vo výrazoch (zapisujú sa ako procedúry, ktoré vracajú hodnoty). Konštanty sú symbolické mená konkrétnych výrazov a sú väčšinou použité ako parametre procedúr.

3.1 Všeobecné služby

3.1.1 Volania na prácu s časom

Wait(čas) **Procedúra**

Pozastaví vykonávanie úlohy (task-u) na stanovený čas (zadaný v tisícinách sekundy). Argument môže byť výraz alebo konštanta:

```
Wait(1000); // počká 1 sekundu
Wait(Random(1000)); // počká náhodný časový interval max 1 sekundu
```

CurrentTick() **Operácia**

Vráti neznamienkové 32-bitové číslo, aktuálny systémový čas ("tick") v milisekundách.

```
x = CurrentTick();
```

FirstTick() **Operácia**

Vráti neznamienkové 32-bitové číslo, systémový čas ("tick") v milisekundách v okamihu, keď program začal bežať.

```
x = FirstTick();
```

SleepTime() **Operácia**

Vráti počet minút ktoré NXT ostáva zapnuté predtým ako sa automaticky vypína.

```
x = SleepTime();
```

SleepTimer() Operácia

Vráti počet minút, ktoré zostávajú dovtedy kým sa NXT automaticky vypne.

```
x = SleepTimer();
```

ResetSleepTimer() Procedúra

Nastaví časovač automatického vypnutia znova na hodnotu SleepTime(). Ak je táto funkcia volaná periodicky, NXT sa nikdy nevypne.

```
ResetSleepTimer();
```

SetSleepTime(minúty) Procedúra

Nastaví počet minút po ktorých uplynutí od zapnutia sa NXT automaticky vypína.

```
SetSleepTime(8);
```

SleepNow() Procedúra

Vypne NXT v tomto okamihu.

```
SleepNow();
```

SetSleepTimer(minúty) Procedúra

Nastaví počet minút po ktorých sa od tohto okamihu NXT automaticky vypne.

```
SetSleepTimer(3);
```

3.1.2 Funkcie na riadenie programu

Stop(logická hodnota) Procedúra

Zastaví vykonávanie programu, ak argument je true. Všetky príkazy, ktoré nasledujú za volaním tejto funkcie budú ignorované a nevykonajú sa.

```
Stop(x == 24); // zastaví program ak x==24
```

StopAllTasks() Procedúra

Zastaví všetky bežiace úlohy (task-y). Všetky príkazy, ktoré nasledujú za volaním tejto funkcie budú ignorované a nevykonajú sa.

```
StopAllTasks(); // zastaví program
```

StartTask(task) Procedúra

Spustí zadanú úlohu.

```
StartTask(zvuk); // spustí úlohu zvuk
```

StopTask(task) Procedúra

Zastaví zadanú úlohu. Tento príkaz vyžaduje rozšírený (enhanced) NBC/NXC firmware

```
StopTask(zvuk); // zastaví úlohu zvuk
```

Acquire(mutex) Procedúra

Zamkne zadanú mutexovú premennú. Ak tento mutex už zamkla nejaká iná úloha, tak volajúca úloha bude pozastavená dovtedy, kým ho prvá úloha neodomkne. Táto funkcia sa používa na zabezpečenie výhradného prístupu k nejakému zdieľanému zdroju, napr. displeju alebo motoru.

Potom, čo táto úloha dokončí prácu so zdieľaným zdrojom by mala zavolať funkciu Release, aby mutex odomkla pre ostatné úlohy.

```
Acquire(motorMutex); // uistíme sa, že máme výhradný prístup
// práca s motorom
Release(motorMutex);
```

Release(mutex) Procedúra

Odomkne zadanú mutexovú premennú, aby ju mohľa zamknúť nejaká iná úloha. Procedúra Release by mala byť zavolaná čo najskôr po zodpovedajúcom volaní funkcie Acquire, hneď potom, ako prestane pracovať s príslušným zdieľaným zdrojom.

```
Acquire(motorMutex); // uistíme sa, že máme výhradný prístup
// práca s motorom
Release(motorMutex); // odomkneme mutex pre ostatné úlohy
```

Precedes(task1, task2, ..., taskN) Procedúra

Spustí zadané úlohy hneď potom ako aktuálna úloha skončí. Úlohy budú bežať súčasne, ak tomu nebránia iné závislosti. Táto funkcia by mala byť volaná v danej úlohe jeden raz, najlepšie na jej začiatku.

```
Precedes(pohyb, kreslenie, melodia);
```

Follows(task1, task2, ..., taskN) Procedúra

Spustí túto úlohu po skončení ktorejkoľvek zo zadaných úloh. Táto funkcia by mala byť volaná v danej úlohe jeden raz, najlepšie na jej začiatku. V prípade, že viacero úloh deklaruje, že nasledujú tú istú úlohu, budú spustené súčasne, ak im v tom nebránia iné závislosti.

```
Follows(main);
```

ExitTo(task) Procedúra

Okamžite skončí vykonávanie tejto úlohy a začne vykonávať zadanú úlohu.

```
ExitTo(ináÚloha);
```

3.1.3 Funkcie na prácu s reťazcami

StrToNum(reťazec) Operácia

Vráti číselnú hodnotu uloženú v zadanom reťazci. Ak string neobsahuje číselnú hodnotu, funkcia vráti nulu.

```
x = StrToNum(strVal);
```

StrLen(reťazec) Operácia

Vráti dĺžku zadaného reťazca. Reťazec nezahŕňa ukončovací znak s kódom nula na konci (null terminator).

```
x = StrLen(msg); // vráti dĺžku reťazca v premennej msg
```

StrIndex(reťazec, index) Operácia

Vráti hodnotu znaku na zadanej pozícii v reťazci.

```
x = StrIndex(msg, 2); // vráti hodnotu msg[2]
```

NumToStr(hodnota) Operácia

Vráti reťazec, ktorý obsahuje zadané číslo.

```
msg = NumToStr(-2); // vráti "-2" v reťazci
```

FormatNum(formát, hodnota) Operácia

Vráti reťazec naformátovaný podľa zadaného formátu a zadanej hodnoty. Formát sa zadáva rovnakým spôsobom ako vo funkcii sprintf v jazyku C.

```
msg = FormatNum("value = %d", x);
```

StrCat(reťazec1, reťazec2, ..., reťazecN) Operácia

Vráti reťazec, ktorý vznikne zreťazením všetkých zadaných reťazcových argumentov.

```
msg = StrCat("vyskusajte", "prosim"); // vráti "vyskusajteprosim"
```

SubStr(reťazec, index, dĺžka) Operácia

Vráti podreťazec požadovanej dĺžky, ktorý začína na zadanej pozícii.

```
msg = SubStr("test", 1, 2); // vráti "es"
```

StrReplace(reťazec, index, novýReťazec) Operácia

Vráti výsledný reťazec, v ktorom je časť pôvodného reťazca, ktorá začína na zadanom indexe, nahradená novým reťazcom.

```
msg = StrReplace("testing", 3, "xx"); // vráti "tesxxng"
```

Flatten(hodnota) Operácia

Vráti reťazec, ktorý zodpovedá zadanej ascii hodnote (aj viacbajtovej).

```
msg = Flatten(48); // vráti "0" lebo 48 == ascii("0")
msg = Flatten(12337); // vráti "10" (little-endian)
```

FlattenVar(premenná) Operácia

Vráti reťazcovú reprezentáciu zadanej hodnoty premennej ľubovoľného typu.

```
stringValue = FlattenVar(mojaStruktura);
```

UnflattenVar(reťazec, premenná) Procedúra

Konvertuje reťazec, ktorý obsahuje reťazcovú reprezentáciu nejakej hodnoty naspäť na hodnotu, ktorú uloží do zadanej premennej.

```
UnflattenVar(retazcovaHodnota, mojaStruktura);
```

ByteArrayToStr(pole) Operácia

Konvertuje zadané pole na reťazec, čiže na koniec pridá nulový znak (null terminator). Pole musí byť jednorozmerné pole typu byte.

```
myStr = ByteArrayToStr(mojePole);
```

ByteArrayToStrEx(pole, reťazec) Procedúra

Konvertuje zadané pole na reťazec, čiže na koniec pridá nulový znak (null terminator). Pole musí byť jednorozmerné pole typu byte.

```
ByteArrayToStrEx(mojePole, mojRetazec);
```

3.1.4 Funkcie na prácu s poľom

StrToByteArray(reťazec, pole) **Procedúra**

Konvertuje zadaný reťazec na bajtové pole tak, že odstráni nulový znak (null terminator) na konci. Výstupná premenná pole musí byť jednorozmerné pole typu byte.

```
StrToByteArray(mojRetazec, mojePole);
```

ArrayLen(pole) **Operácia**

Vráti dĺžku zadaného poľa.

```
x = ArrayLen(mojePole);
```

ArrayInit(pole, hodnota, pocet) **Procedúra**

Inicializuje pole na zadanú dĺžku, všetky prvky budú obsahovať zadanú hodnotu. Viacrozmerné polia je možné inicializovať tak, že zadaná hodnota je N-1 rozmerné pole (pričom N je počet rozmerov inicializovaného poľa).

```
ArrayInit(mojePole, 0, 10); // 10 prkov rovných nule
```

ArraySubset(výsledné pole, zdrojové pole, index, dĺžka) **Procedúra**

Skopíruje časť zdrojového poľa zadanej dĺžky počínajúc zadaným indexom do výsledného poľa.

```
ArraySubset(mojePole, zdrojovePole, 2, 5); skopíruje 5 prvkov
```

ArrayBuild(výsledné pole, zdroj1 [, zdroj2, …, zdrojN]) **Procedúra**

Vytvorí nové pole zo zadaných zdrojov. Zdrojové premenné môžu byť ľubovoľného typu. Ak niektorý zdroj je pole, všetky jeho provky sú pridané do výstupu.

```
ArrayBuild(mojePole, zdr1, zdr2);
```

3.1.5 Funkcie na prácu s číslami

Random(N) **Operácia**

Vráti neznamienkové 16-bitové náhodné číslo od 0 po N-1 (vrátane). N môže byť konštanta alebo premenná.

```
x = Random(10); // vráti hodnotu 0..9
```

Random() **Operácia**

Vráti znamienkové 16-bitové náhodné číslo.

```
x = Random();
```

Sqrt(x) **Operácia**

Vráti druhú odmocninu zadanej hodnoty x.

```
x = Sqrt(x);
```

Sin(stupne) **Operácia**

Vráti sinus zadanej hodnoty v stupňoch. Výsledná hodnota je vynásobená 100, takže výsledok je v rozmedzí (-100..100).

```
x = Sin(theta);
```

Cos(stupne) Operácia

Vráti cosinus zadanej hodnoty v stupňoch. Výsledná hodnota je vynásobená 100, takže výsledok je v rozmedzí (-100..100).

```
x = Cos(y);
```

Asin(hodnota) Operácia

Inverzná funkcia k funkcii Sin. Vráti arcussinus zadanej hodnoty, ktorá je v rozmedzí (-100..100). Výsledok je v intervale (-90..90).

```
stupne = Asin(80);
```

Acos(hodnota) Operácia

Inverzná funkcia k funkcii Cos. Vráti arcuscosinus zadanej hodnoty, ktorá je v rozmedzí (-100..100). Výsledok je v intervale (-90..90).

```
stupne = Acos(0);
```

bcd2dec(bcdHodnota) Operácia

Konvertuje binárne kódované desiatkové číslo na desiatkové číslo.

```
dec = bcd2dec(0x32);
```

3.1.6 Systémové volania

Nízkoúrovňové volania NXC API popísané v tejto časti vyžadujú argumenty štruktúrovaných typov. Pred volaním takejto systémovej funkcie treba najskôr zadefinovať premennú štruktúrovaného typu a vyplniť jej príslušné polia. Systémové volania vracajú výsledky tiež v poliach týchto štruktúr, takže po ich volaní sa výsledok nachádza tam. Mnohé z týchto systémových funkcií sú obalené do NXC API funkcií na vyššej úrovni, popísané v iných statiach tohto dokumentu, ktoré tieto štruktúry povypĺňajú automaticky, takže sa o ne programátor nemusí starať a namiesto toho iba zavolá jednu funkciu. Použitím nízkoúrovňových funkcií sa beh programu môže mierne zrýchliť a je možné sa dostať aj k tým vlastnostiam, ktoré nie sú bežnými funkciami prístupné.

Po nainštalovaní rozšíreného (enhanced) NXT firmware, všetky systémové funkcie na kreslenie na obrazovke podporujú okrem kreslenia aj zmazávanie bodov. Zmazávanie zapneme tak, že do položky Options v príslušnej štruktúre zadáme hodnotu `DRAW_OPT_CLEAR_PIXELS` (0x0004). K tejto hodnote možno pripočítať hodnotu `DRAW_OPT_CLEAR_WHOLE_SCREEN` (0x0001) ktorá znamená, že sa má pred kreslením zmazať celá obrazovka. Aj niektoré ostatné popísané funkcie sú prístupné len v rozšírenom (enhanced) firmare. Tieto funkcie sú označené krížikom **(+)**.

Prvé dve štruktúry definujú typy, ktoré sa využívajú v ďalších štruktúrach, ktoré kreslia na obrazovku: poloha a veľkosť (šírka, výška):

```
struct LocationType {
  int X;
  int Y;
};

struct SizeType {
  int Width;
  int Height;
};
```

SysDrawText(DrawTextType & args) Procedúra

Vykreslí zadaný text na LCD. Deklarácia použitej štruktúry je nasledujúca:

```
struct DrawTextType {
  char Result;
  LocationType Location;
  string Text;
  unsigned long Options;
};
```

Definujte premennú tohto typu, vyplnte jej prvky a zavolajte funkciu s touto premennou v argumente. Konštanty LCD_LINE1, … LCD_LINE7 sú čísla, ktoré zodpovedajú Y-ovým súradniciam pre jednotlivé riadky na displeji. Text vykreslený na inej Y-ovej súradnici bude automaticky zarovnaný na jeden z týchto riadkov.

```
DrawTextType dtArgs;
dtArgs.Location.X = 0;
dtArgs.Location.Y = LCD_LINE1;
dtArgs.Text = "Please Work";
dtArgs.Options = 0x01; // zmazať obrazovku vopred (0 = nemazať)
SysDrawText(dtArgs);
```

SysDrawPoint(DrawPointType & args) Procedúra

Na LCD nakreslí jednu bodku na zadanej pozícii (Location).

```
struct DrawPointType {
  char Result;
  LocationType Location;
  unsigned long Options;
};
```

Príklad:

```
DrawPointType dpArgs;
dpArgs.Location.X = 20;
dpArgs.Location.Y = 20;
dpArgs.Options = 0x04; // túto bodku chceme zmazať (0 = nakresliť)
SysDrawPoint(dpArgs);
```

SysDrawLine(DrawLineType & args) Procedúra

Na LCD nakreslí úsečku z bodu StartLoc do bodu EndLoc.

```
struct DrawLineType {
  char Result;
  LocationType StartLoc;
  LocationType EndLoc;
  unsigned long Options;
};
```

Príklad:

```
DrawLineType dlArgs;
dlArgs.StartLoc.X = 20;
dlArgs.StartLoc.Y = 20;
dlArgs.EndLoc.X = 60;
dlArgs.EndLoc.Y = 60;
dlArgs.Options = 0x01; // zmazať obrazovku pred kreslením
SysDrawLine(dlArgs);
```

SysDrawCircle(DrawCircleType & args) Procedúra

Na LCD nakreslí kružnicu so stredom Center a polomerom Size.

```
struct DrawCircleType {
  char Result;
  LocationType Center;
  byte Size;
  unsigned long Options;
};
```

Príklad:

```
DrawCircleType dcArgs;
dcArgs.Center.X = 20;
dcArgs.Center.Y = 20;
dcArgs.Size = 10; // polomer
dcArgs.Options = 0x01; // zmazať obrazovku pred kreslením
SysDrawCircle(dcArgs);
```

SysDrawRect(DrawRectType & args) Procedúra

Na LCD nakreslí obdĺžnik na zadanú pozíciu.

```
struct DrawRectType {
  char Result;
  LocationType Location;
  SizeType Size;
  unsigned long Options;
};
```

Príklad:

```
DrawRectType drArgs;
drArgs.Location.X = 20;
drArgs.Location.Y = 20;
drArgs.Size.Width = 20;  // šírka
drArgs.Size.Height = 10; // výška
drArgs.Options = 0x00;   // obrazovku pred kreslením nezmazať
SysDrawRect(drArgs);
```

SysDrawGraphic(DrawGraphicType & args) Procedúra

Na LCD vykreslí na zadané miesto obrázok zo súboru typu RIC:

```
struct DrawGraphicType {
  char Result;
  LocationType Location;
  string Filename;
  int Variables[];
  unsigned long Options;
};
```

Príklad:

```
DrawGraphicType dgArgs;
dgArgs.Location.X = 20;
dgArgs.Location.Y = 20;
dgArgs.Filename = "image.ric";
ArrayInit(dgArgs.Variables, 0, 10); // 10 núl
dgArgs.Variables[0] = 12;
```

```
dgArgs.Variables[1] = 14; // atď...
dgArgs.Options = 0x00; // nezmazať obrazovku pred kreslením
SysDrawGraphic(dgArgs);
```

SysSetScreenMode(SetScreenModeType & args) Procedúra

Nastaví režim kreslenia na LCD. Štandardný firmware podporuje iba režim `SCREEN_MODE_RESTORE`, hodnota 0x00. Rozšírený (enhanced) firmware podporuje aj režim `SCREEN_MODE_CLEAR`, s hodnotou 0x01.

```
struct SetScreenModeType {
  char Result;
  unsigned long ScreenMode;
};
```

Príklad:

```
SetScreenModeType ssmArgs;
ssmArgs.ScreenMode = 0x00; // obnoviť pôvodný režim
SysSetScreenMode(ssmArgs);
```

SysSoundPlayFile(SoundPlayFileType & args) Procedúra

Zahrá zvuk zo súboru typu RSO (obsahujúcu PCM alebo spakovanú ADPCM zvukovú vzorku) alebo zo súboru typu RMD (obsahujúcu melódiu určenú tónami, ktoré sú zadané ich frekvenciami a dĺžkami).

```
struct SoundPlayFileType {
  char Result;
  string Filename;
  bool Loop;
  byte SoundLevel;
};
```

Príklad:

```
SoundPlayFileType spfArgs;
spfArgs.Filename = "hello.rso";
spfArgs.Loop = false;
spfArgs.SoundLevel = 3;
SysSoundPlayFile(spfArgs);
```

SysSoundPlayTone(SoundPlayToneType & args) Procedúra

Zahrá tón zadanej frekvencie, dĺžky a hlasitosti.

```
struct SoundPlayToneType {
  char Result;
  unsigned int Frequency;
  unsigned int Duration;
  bool Loop;
  byte SoundLevel;
};
```

Príklad:

```
SoundPlayToneType sptArgs;
sptArgs.Frequency = 440; // tón A
sptArgs.Duration = 1000; // 1 sekunda
sptArgs.Loop = false;    // zahrať iba raz
sptArgs.SoundLevel = 3;  // hlasitosť
SysSoundPlayTone(sptArgs);
```

SysSoundGetState(SoundGetStateType & args) Procedúra

Zistí informáciu o stave zvukového modulu. Stav popisujú konštanty: SOUND_STATE_IDLE (nehrá sa), SOUND_STATE_FILE (hrá sa zvuková vzorka), SOUND_STATE_TONE a SOUND_STATE_STOP (modul vypnutý). Príznaky popisujú konštanty: SOUND_FLAGS_IDLE, SOUND_FLAGS_UPDATE, a SOUND_FLAGS_RUNNING.

```
struct SoundGetStateType {
  byte State;
  byte Flags;
};
```

Príklad:

```
SoundGetStateType sgsArgs;
SysSoundGetState(sgsArgs);
if (sgsArgs.State == SOUND_STATE_IDLE) {/* pridaj melódiu... */}
```

SysSoundSetState(SoundSetStateType & args) Procedúra

Nastaví stav zvukového modulu. Konštanty popisujúce stav sú SOUND_STATE_IDLE, SOUND_STATE_FILE, SOUND_STATE_TONE, a SOUND_STATE_STOP. Konštanty popisujúce príznaky sú SOUND_FLAGS_IDLE, SOUND_FLAGS_UPDATE, a SOUND_FLAGS_RUNNING.

```
struct SoundSetStateType {
  byte Result;
  byte State;
  byte Flags;
};
```

Príklad:

```
SoundSetStateType sssArgs;
sssArgs.State = SOUND_STATE_STOP;
SysSoundSetState(sssArgs);
```

SysReadButton(ReadButtonType & args) Procedúra

Prečíta stav tlačidiel NXT.

```
struct ReadButtonType {
  char Result;
  byte Index;    // ktoré tlačidlo
  bool Pressed;  // stlačené?
  byte Count;    // koľkokrát bolo stlačené
  bool Reset; // zmazať počet stlačení po prečítaní hodnoty
};
```

Príklad:

```
ReadButtonType rbArgs;
rbArgs.Index = BTNRIGHT;
SysReadButton(rbArgs);
if (rbArgs.Pressed) {/* urob niečo */}
```

SysRandomNumber(RandomNumberType & args) Procedúra

Vygeneruje náhodné číslo.

```
struct RandomNumberType {
  int Result;
};
```

Príklad:

```
RandomNumberType rnArgs;
SysRandomNumber(rnArgs);
int mojeNahodneCislo = rnArgs.Result;
```

SysGetStartTick(GetStartTickType & args) Procedúra

Vráti hodnotu systémového času (tick) v okamihu, keď bol program spustený.

```
struct GetStartTickType {
  unsigned long Result;
};
```

Príklad:

```
GetStartTickType gstArgs;
SysGetStartTick(gstArgs);
unsigned long myStart = gstArgs.Result;
```

SysKeepAlive(KeepAliveType & args) Procedúra

Nastaví sleep timer na jeho východziu hodnotu.

```
struct KeepAliveType {
  unsigned long Result;
};
```

Príklad:

```
KeepAliveType kaArgs;
SysKeepAlive(kaArgs); // reset sleep timer
```

SysFileOpenWrite(FileOpenType & args) Procedúra

Vytvorí a otvorí súbor v pamäti NXT typu flash na čítanie. Hodnotu, ktorú funkcia vráti v položke FileHandle, treba použiť v nasledujúcich operáciách zápisu a zatvorenia súboru. Výsledná plánovaná dĺžka súboru v bajtoch je určená hodnotou položky Length.

```
struct FileOpenType {
  unsigned int Result;
  byte FileHandle;
  string Filename;
  unsigned long Length;
};
```

Príklad:

```
FileOpenType foArgs;
foArgs.Filename = "myfile.txt";
foArgs.Length = 256; // veľkosť súboru bude 256 bajtov
SysFileOpenWrite(foArgs); // vytvor súbor
if (foArgs.Result == NO_ERR) {
// zápis do súboru pomocou vrátenej FileHandle...
}
```

SysFileOpenAppend(FileOpenType & args) Procedúra

Otvorí existujúci súbor v NXT pamäti typu flash na pridávanie. Vrátenú hodnotu FileHandle

treba použiť v nasledujúcich operáciách zápisu a uzavretia súboru. Zostávajúci počet bajtov, ktoré možno do súboru zapísať bude vrátený v položke Length.

```
struct FileOpenType {
  unsigned int Result;
  byte FileHandle;
  string Filename;
  unsigned long Length;
};
```

Príklad:

```
FileOpenType foArgs;
foArgs.Filename = "myfile.txt";
SysFileOpenAppend(foArgs); // open the file
if (foArgs.Result == NO_ERR) {
  // zápis do súboru pomocou FileHandle
  // najviac však Length bajtov
}
```

SysFileOpenRead(FileOpenType & args) Procedúra

Otvorí existujúci súbor v NXT pamäti typu flash na čítanie. Vrátená hodnota FileHandle sa použije na vo volaniach na čítanie a zatvorenie tohto súboru. Dĺžka otvoreného súboru je vrátená v položke Length.

```
struct FileOpenType {
  unsigned int Result;
  byte FileHandle;
  string Filename;
  unsigned long Length;
};
```

Príklad:

```
FileOpenType foArgs;
foArgs.Filename = "myfile.txt";
SysFileOpenRead(foArgs); // otvorí súbor na čítanie
if (foArgs.Result == NO_ERR) {
// čítaj data zo súboru pomocou FileHandle
}
```

SysFileRead(FileReadWriteType & args) Procedúra

Prečíta niekoľko bajtov zo súboru, ktorý bol otvorený na čítanie.

```
struct FileReadWriteType {
  unsigned int Result;
  byte FileHandle;
  string Buffer;
  unsigned long Length;
};
```

Príklad:

```
FileReadWriteType frArgs;
frArgs.FileHandle = foArgs.FileHandle;
frArgs.Length = 12; // koľko bajtov prečítať
SysFileRead(frArgs);
if (frArgs.Result == NO_ERR) {
```

```
  TextOut(0, LCD_LINE1, frArgs.Buffer);
  // vypíš na obrazovku koľko bajtov sa v skutočnosti prečítalo
  NumOut(0, LCD_LINE2, frArgs.Length);
}
```

SysFileWrite(FileReadWriteType & args) Procedúra

Zapíše niekoľko bajtov do súboru.

```
struct FileReadWriteType {
  unsigned int Result;
  byte FileHandle;
  string Buffer;
  unsigned long Length;
};
```

Príklad:

```
FileReadWriteType fwArgs;
fwArgs.FileHandle = foArgs.FileHandle;
fwArgs.Buffer = "data to write";
SysFileWrite(fwArgs);
if (fwArgs.Result == NO_ERR) {
  // zobrazí počet zapísaných bajtov
  NumOut(0, LCD_LINE1, fwArgs.Length);
}
```

SysFileClose(FileCloseType & args) Procedúra

Zatvorí súbor.

```
struct FileCloseType {
  unsigned int Result;
  byte FileHandle;
};
```

Príklad:

```
FileCloseType fcArgs;
fcArgs.FileHandle = foArgs.FileHandle;
SysFileClose(fcArgs);
```

SysFileResolveHandle(FileResolveHandleType & args) Procedúra

Zistí, či súbor je otvorený na zápis alebo na čítanie

```
struct FileResolveHandleType {
  unsigned int Result;
  byte FileHandle;
  bool WriteHandle;
  string Filename;
};
```

Príklad:

```
FileResolveHandleType frhArgs;
frhArgs.Filename = "myfile.txt";
SysFileResolveHandle(frhArgs);
if (frhArgs.Result == LDR_SUCCESS) {
  // volanie bolo úspešné, použijeme FileHandle podľa jej typu
  if (frhArgs.WriteHandle) {
```

```
    // súbor je otvorený na zápis
  }
  else {
    // súbor je otvorený na čítanie
  }
}
```

SysFileRename(FileRenameType & args) Procedúra

Premenuje súbor v NXT pamäti typu flash.

```
struct FileRenameType {
  unsigned int Result;
  string OldFilename;
  string NewFilename;
};
```

Príklad:

```
FileRenameType frArgs;
frArgs.OldFilename = "myfile.txt";
frArgs.NewFilename = "myfile2.txt";
SysFileRename(frArgs);
if (frArgs.Result == LDR_SUCCESS) { /* premenovanie ok */ }
```

SysFileDelete(FileDeleteType & args) Procedúra

Vymaže súbor v NXT pamäti typu flash.

```
struct FileDeleteType {
  unsigned int Result;
  string Filename;
};
```

Príklad:

```
FileDeleteType fdArgs;
fdArgs.Filename = "myfile.txt";
SysFileDelete(fdArgs); // vymaž súbor
```

SysCommLSWrite(CommLSWriteType & args) Procedúra

Vyšle paket na I2C (Lowspeed) zbernicu na zadanom porte.

```
struct CommLSWriteType {
  char Result;
  byte Port;
  byte Buffer[];
  byte ReturnLen;
};
```

Príklad:

```
CommLSWriteType args;
args.Port = S1;
args.Buffer = myBuf;
args.ReturnLen = 8;
SysCommLSWrite(args);
// zisti, či zápis prebehol v poriadku podľa položky Result
```

SysCommLSCheckStatus(CommLSCheckStatusType & args) Procedúra

Zistí stav transakcie na I2C (Lowspeed) zbernici.

```
struct CommLSCheckStatusType {
  char Result;
  byte Port;
  byte BytesReady;
};
```

Príklad:

```
CommLSCheckStatusType args;
args.Port = S1;
SysCommLSCheckStatus(args);
// je zbernica nečinná? Stav (Result) je IDLE?
if (args.Result == LOWSPEED_IDLE) { /* pokračuj */ }
```

SysCommLSRead(CommLSReadType & args) Procedúra

Prečíta paket z I2C (Lowspeed) zbernice.

```
struct CommLSReadType {
  char Result;
  byte Port;
  byte Buffer[];
  byte BufferLen;
};
```

Príklad:

```
CommLSReadType args;
args.Port = S1;
args.Buffer = myBuf;
args.BufferLen = 8;
SysCommLSRead(args);
// skontroluj Result či nedošlo k cyhbe a použi obsah bufra
```

SysMessageWrite(MessageWriteType & args) Procedúra

Zapíše správu do príslušného priečinku (mailboxu). Správy si NXT kocka vymieňa s inými zariadeniami (napr. inými NXT kockami) cez rádiové spojenie typu BlueTooth.

```
struct MessageWriteType {
  char Result;
  byte QueueID;
  string Message;
};
```

Príklad:

```
MessageWriteType args;
args.QueueID = MAILBOX1; // 0
args.Message = "sprava";
SysMessageWrite(args);
// skontroluj Result či nedošlo k chybe
```

SysMessageRead(MessageReadType & args) Procedúra

Prečíta správu z príslušného priečinku (mailboxu). Správy si NXT kocka vymieňa s inými zariadeniami (napr. inými NXT kockami) cez rádiové spojenie typu BlueTooth.

```
struct MessageReadType {
  char Result;
  byte QueueID;
  bool Remove;
  string Message;
};
```

Príklad:

```
MessageReadType args;
args.QueueID = MAILBOX1; // 0
args.Remove = true;
SysMessageRead(args);
if (args.Result == NO_ERR) {
  TextOut(0, LCD_LINE1, args.Message);
}
```

SysCommBTWrite(CommBTWriteType & args) Procedúra

Zapíše zadaný paket na zadané spojenie BlueTooth.

```
struct CommBTWriteType {
  char Result;
  byte Connection;
  byte Buffer[];
};
```

Príklad:

```
CommBTWriteType args;
args.Connection = 1;
args.Buffer = myData;
SysCommBTWrite(args);
```

SysCommBTCheckStatus(CommBTCheckStatusType & args) Procedúra

Zistí stav zadaného spojenia BlueTooth. Pole Result môže nadobudnúť nasledujúce hodnoty: `ERR_INVALID_PORT`, `STAT_COMM_PENDING`, `ERR_COMM_CHAN_NOT_READY` a `LDR_SUCCESS` (0).

```
struct CommBTCheckStatusType {
  char Result;
  byte Connection;
  byte Buffer[];
};
```

Príklad:

```
CommBTCheckStatusType args;
args.Connection = 1;
SysCommBTCheckStatus(args);
if (args.Result == LDR_SUCCESS) { /* pokračuj */ }
```

SysIOMapRead(IOMapReadType & args) Procedúra

Prečíta údaje z modulu IOMap.

```
struct IOMapReadType {
  char Result;
  string ModuleName;
  unsigned int Offset;
```

```
  unsigned int Count;
  byte Buffer[];
};
```

Príklad:

```
IOMapReadType args;
args.ModuleName = CommandModuleName;
args.Offset = CommandOffsetTick;
args.Count = 4; // ide o štvorbajtovú hodnotu
SysIOMapRead(args);
if (args.Result == NO_ERR) { /* pokračuj */ }
```

SysIOMapWrite(IOMapWriteType & args) Procedúra

Zapíše údaje do modulu IOMap.

```
struct IOMapWriteType {
  char Result;
  string ModuleName;
  unsigned int Offset;
  byte Buffer[];
};
```

Príklad:

```
IOMapWriteType args;
args.ModuleName = SoundModuleName;
args.Offset = SoundOffsetSampleRate;
args.Buffer = theData;
SysIOMapWrite(args);
```

SysIOMapReadByID(IOMapReadByIDType & args) Procedúra (+)

Prečíta údaje z modulu IOMap. Táto funkcia sa môže vykonať aj trikrát rýchlejšie ako SysIOMapRead keďže modul nemusí vyhľadávať pomocou reťazca ModuleName.

```
struct IOMapReadByIDType {
  char Result;
  unsigned long ModuleID;
  unsigned int Offset;
  unsigned int Count;
  byte Buffer[];
};
```

Príklad:

```
IOMapReadByIDType args;
args.ModuleID = CommandModuleID;
args.Offset = CommandOffsetTick;
args.Count = 4; // táto hodnota je štvorbajtová
SysIOMapReadByID(args);
if (args.Result == NO_ERR) { /* pokračuj */ }
```

SysIOMapWriteByID(IOMapWriteByIDType & args) Procedúra (+)

Zapíše údaje do modulu IOMap. Táto funkcia sa môže vykonať aj trikrát rýchlejšie ako SysIOMapWrite keďže modul nemusí vyhľadávať pomocou reťazca ModuleName.

```
struct IOMapWriteByIDType {
  char Result;
```

```
  unsigned long ModuleID;
  unsigned int Offset;
  byte Buffer[];
};
```

Príklad:

```
IOMapWriteByIDType args;
args.ModuleID = SoundModuleID;
args.Offset = SoundOffsetSampleRate;
args.Buffer = theData;
SysIOMapWriteByID(args);
```

SysDisplayExecuteFunction(DisplayExecuteFunctionType & args) Procedúra(+)

Priamo vykoná funkciu modulu Display. Hodnoty polí štruktúry sú popísané v nasledujúcej tabuľke, tam, kde je uvedené 'x' je hodnota príslušnej položky ignorovaná.

```
struct DisplayExecuteFunctionType {
  byte Status;
  byte Cmd;
  bool On;
  byte X1;
  byte Y1;
  byte X2;
  byte Y2;
};
```

Príkaz	Význam	Argumenty
DISPLAY_ERASE_ALL	zmaže celú obrazovku	()
DISPLAY_PIXEL	nastaví bod (on/off)	(true/false,X1,Y1,x,x)
DISPLAY_HORIZONTAL_LINE	nakreslí vodorovnú úsečku	(true/false,X1,Y1,X2,x)
DISPLAY_VERTICAL_LINE	nakreslí zvislú úsečku	(true/false,X1,Y1,x,Y2)
DISPLAY_CHAR	nakreslí znak (aktuálnym fontom)	(true/false,X1,Y1,Char,x)
DISPLAY_ERASE_LINE	zmaže úsečku	(x,LINE,x,x,x)
DISPLAY_FILL_REGION	vyplní oblasť na obrazovke	(true/false,X1,Y1,X2,Y2)
DISPLAY_FILLED_FRAME	nakreslí obdĺžnik (on / off)	(true/false,X1,Y1,X2,Y2)

Príklad:

```
DisplayExecuteFunctionType args;
args.Cmd = DISPLAY_ERASE_ALL;
SysDisplayExecuteFunction(args);
```

SysCommExecuteFunction(CommExecuteFunctionType & args) Procedúra (+)

Priamo vykoná funkciu modulu Comm. Hodnoty polí štruktúry sú popísané v nasledujúcej tabuľke, tam, kde je uvedené 'x' je hodnota príslušnej položky ignorovaná.

```
struct CommExecuteFunctionType {
  unsigned int Result;
  byte Cmd;
```

```
  byte Param1;
  byte Param2;
  byte Param3;
  string Name;
  unsigned int RetVal;
};
```

Príkaz	Význam	(Param1,Param2,Param3,Meno)
INTF_SENDFILE	Vyšle súbor cez spojenie BlueTooth	(Connection,x,x,Filename)
INTF_SEARCH	Search for Bluetooth devices	(x,x,x,x)
INTF_STOPSEARCH	Stop searching for Bluetooth devices	(x,x,x,x)
INTF_CONNECT	Connect to a Bluetooth device	(DeviceIndex,Connection,x,x)
INTF_DISCONNECT	Disconnect a Bluetooth device	(Connection,x,x,x)
INTF_DISCONNECTALL	Disconnect all Bluetooth devices	(x,x,x,x)
INTF_REMOVEDEVICE	Remove device from My Contacts	(DeviceIndex,x,x,x)
INTF_VISIBILITY	Set Bluetooth visibility	(true/false,x,x,x)
INTF_SETCMDMODE	Set command mode	(x,x,x,x)
INTF_OPENSTREAM	Open a stream	(x,Connection,x,x)
INTF_SENDDATA	Send data	(Length, Connection, WaitForIt, Buffer)
INTF_FACTORYRESET	Bluetooth factory reset	(x,x,x,x)
INTF_BTON	Turn Bluetooth on	(x,x,x,x)
INTF_BTOFF	Turn Bluetooth off	(x,x,x,x)
INTF_SETBTNAME	Set Bluetooth name	(x,x,x,x)
INTF_EXTREAD	Handle external? read	(x,x,x,x)
INTF_PINREQ	Handle Blueooth PIN request	(x,x,x,x)
INTF_CONNECTREQ	Handle Bluetooth connect request	(x,x,x,x)

Príklad:

```
CommExecuteFunctionType args;
args.Cmd = INTF_BTOFF;
SysCommExecuteFunction(args);
```

SysLoaderExecuteFunction(LoaderExecuteFunctionType & args) Procedúra (+)

Priamo vykoná funkciu modulu Loader. Hodnoty polí štruktúry sú popísané v nasledujúcej tabuľke, tam, kde je uvedené 'x' je hodnota príslušnej položky ignorovaná.

```
struct LoaderExecuteFunctionType {
  unsigned int Result;
  byte Cmd;
  string Filename;
  byte Buffer[];
  unsigned long Length; };
```

Príkaz	Význam	Argumenty
LDR_CMD_OPENREAD	Open a file for reading	(Filename, Length)
LDR_CMD_OPENWRITE	Creat a file	(Filename, Length)
LDR_CMD_READ	Read from a file	(Filename, Buffer, Length)
LDR_CMD_WRITE	Write to a file	(Filename, Buffer, Length)
LDR_CMD_CLOSE	Close a file	(Filename)
LDR_CMD_DELETE	Delete a file	(Filename)
LDR_CMD_FINDFIRST	Start iterating files	(Filename, Buffer, Length)
LDR_CMD_FINDNEXT	Continue iterating files	(Filename, Buffer, Length)
LDR_CMD_OPENWRITELINEAR	Create a linear file	(Filename, Length)
LDR_CMD_OPENREADLINEAR	Read a linear file	(Filename, Buffer, Length)
LDR_CMD_OPENAPPENDDATA	Open a file for writing	(Filename, Length)
LDR_CMD_FINDFIRSTMODULE	Start iterating modules	(Filename, Buffer)
LDR_CMD_FINDNEXTMODULE	Continue iterating modules	(Buffer)
LDR_CMD_CLOSEMODHANDLE	Close module handle	()
LDR_CMD_IOMAPREAD	Read IOMap data	(Filename, Buffer, Length)
LDR_CMD_IOMAPWRITE	Write IOMap data	(Filename, Buffer, Length)
LDR_CMD_DELETEUSERFLASH	Delete all files	()
LDR_CMD_RENAMEFILE	Rename file	(Filename, Buffer, Length)

Príklad:

```
LoaderExecuteFunctionType args;
args.Cmd = 0xA0; // delete user flash
SysLoaderExecuteFunction(args);
```

SysCall(funcID, args) Procedúra

Toto všeobecné makro môže byť použité na volanie ľubovoľnej systémovej funkcie. Neprebieha žiadna typová kontrola a preto je nutné, aby boli funkcii odovzdané správne vyplnené štruktúry správneho typu. Toto je najrýchlejší spôsob ako volať systémové funkcie v NXC. Procedúra SysCall rozpoznáva nasledujúce konštanty: `FileOpenRead`, `FileOpenWrite`, `FileOpenAppend`, `FileRead`, `FileWrite`, `FileClose`, `FileResolveHandle`, `FileRename`, `FileDelete`, `SoundPlayFile`, `SoundPlayTone`, `SoundGetState`, `SoundSetState`, `DrawText`, `DrawPoint`, `DrawLine`, `DrawCircle`, `DrawRect`, `DrawGraphic`, `SetScreenMode`, `ReadButton`, `CommLSWrite`, `CommLSRead`, `CommLSCheckStatus`, `RandomNumber`, `GetStartTick`, `MessageWrite`, `MessageRead`, `CommBTCheckStatus`, `CommBTWrite`, `KeepAlive`, `IOMapRead`, `IOMapWrite`, `IOMapReadByID`, `IOMapWriteByID`, `DisplayExecuteFunction`, `CommExecuteFunction`, a `LoaderExecuteFunction`.

Príklad:

```
DrawTextType dtArgs;
dtArgs.Location.X = 0;
dtArgs.Location.Y = LCD_LINE1;
dtArgs.Text = "Please Work";
SysCall(DrawText, dtArgs);
```

3.2 Modul Input

Modul Input zabezpečuje všetky vstupy okrem digitálnej zbernice I2C (LowSpeed).

Konštanta	Hodnota
InputModuleName	"Input.mod"
InputModuleID	0x00030001

Tabuľka 7. Konštanty modulu Input

Štyri senzory NXT sú interne číslované 0, 1, 2, a 3. Toto môže byť zdrojom problémov, takže boli zavedené názvy portov `S1`, `S2`, `S3`, a `S4`.

Tieto mená môžu byť použité v ľubovoľnej funkcii, ktorá očakáva číslo senzorového portu v argumente. Je tiež možné použiť konštanty NBC `IN_1`, `IN_2`, `IN_3`, a `IN_4`. Okrem toho sú zadefinované mená `SENSOR_1`, `SENSOR_2`, `SENSOR_3`, a `SENSOR_4`, ktoré môžu byť použité na čítanie aktuálnej hodnoty senzora:

```
x = SENSOR_1; // prečíta senzor a aktuálnu hodnotu uloží do x
```

3.2.1 Typy a Módy

Senzorové porty NXT môžu komunikovať s množstvom rôznych senzorov. Program musí kocke NXT sám oznámiť, aké senzory sú pripojené na jednotlivých portoch. Volanie SetSensorType nastaví typ senzora. Existuje 12 typov senzorov, každý zodpovedá jednému zo senzorov LEGO RCX alebo NXT. Posledný typ (SENSOR_TYPE_NONE) znamená, že nie je pripojený žiadny senzor. Ak program nenastaví typ senzora správne, NXT z neho pravdepodobne nebude vedieť prečítať správnu hodnotu. Je možné použiť bud konštantu popisujúcu typ senzora alebo konštantu popisujúcu NBC typ senzora.

NXT dovoľuje nakonfigurovať jednotlivé senzory v rozličných módoch. Mód senzora určuje, ako sa surový údaj, ktorý sa z neho prečíta predspracuje. Niektoré módy majú zmysel len pre niektoré typy senzorov. Napríklad mód `SENSOR_MODE_ROTATION` je použiteľný iba s otáčkovým senzorom. Volanie SetSensorMode nastavuje mód senzora. Jednotlivé módy sú zobrazené v nasledujúcej tabuľke. Možno použiť mód senzora alebo NBC mód senzora.

Logická hodnota senzorov je k dispozícii pre všetky senzory, nielen dotykové senzory. Táto konverzia je založená na predprogramovanom prahu. „Nízka“ hodnota (menšia ako 460) je považovaná za logickú jednotku a vysoká hodnota (vyššia ako 562) je považovaná za logickú nulu. Táto konverzia môže byť upravená pomocou hodnoty *slope value* ktorá je z intervalu 0 až 31 a je možné ju pridať k módu senzora pri volaní `SetSensorMode`. Ak sa hodnota senzora zmení viac ako slope value za krátky čas (3ms), tak sa logický stav senzora zmení. Takto je možné, aby prudká zmena hodnoty senzora vyvolala zmenu logického stavu. Prudký nárast vedie k logickej nule a prudký pokles k logickej jednotke. Táto konverzia sa bude vykonávať aj vtedy, keď je senzor

konfigurovaný v nejakom inom móde (napr. `SENSOR_MODE_PERCENT`). Každý senzor popisuje v danom okamihu šesť rôznych hodnôt. Tieto polia sú vysvetlené v tabuľke č. 11.

Typ senzora	NBC typ senzora	Význam
SENSOR_TYPE_NONE	IN_TYPE_NO_SENSOR	žiaden senzor nie je pripojený
SENSOR_TYPE_TOUCH	IN_TYPE_SWITCH	NXT alebo RCX dotykový senzor
SENSOR_TYPE_TEMPERATURE	IN_TYPE_TEMPERATURE	tepelný senzor RCX
SENSOR_TYPE_LIGHT	IN_TYPE_REFLECTION	sveteľný senzor RCX
SENSOR_TYPE_ROTATION	IN_TYPE_ANGLE	otáčkový senzor RCX
SENSOR_TYPE_LIGHT_ACTIVE	IN_TYPE_LIGHT_ACTIVE	sveteľný senzor NXT so svetlom
SENSOR_TYPE_LIGHT_INACTIVE	IN_TYPE_LIGHT_INACTIVE	sveteľný senzor NXT bez svetla
SENSOR_TYPE_SOUND_DB	IN_TYPE_SOUND_DB	zvukový senzor NXT v dB škále
SENSOR_TYPE_SOUND_DBA	IN_TYPE_SOUND_DBA	zvukový senzor NXT v škále dBA
SENSOR_TYPE_CUSTOM	IN_TYPE_CUSTOM	iný senzor (CUSTOM) (nepoužité)
SENSOR_TYPE_LOWSPEED	IN_TYPE_LOWSPEED	I2C digitálny senzor
SENSOR_TYPE_LOWSPEED_9V	IN_TYPE_LOWSPEED_9V	I2C digitálny sensor (s 9V napájaním)
SENSOR_TYPE_HIGHSPEED	IN_TYPE_HISPEED	Vysokorýchlostný digitálny senzor (nepoužité)

Tabuľka 8. Konštanty typov senzorov

Mód senzora	NBC mód senzora	Význam
SENSOR_MODE_RAW	IN_MODE_RAW	surová nespracovaná hodnota od 0 do 1023
SENSOR_MODE_BOOL	IN_MODE_BOOLEAN	logická hodnota (0 alebo 1)
SENSOR_MODE_EDGE	IN_MODE_TRANSITIONCNT	počíta logické prechody medzi 0 a 1 a naopak
SENSOR_MODE_PULSE	IN_MODE_PERIODCOUNTER	počíta periódy (úplný prechod 0-1-0)
SENSOR_MODE_PERCENT	IN_MODE_PCTFULLSCALE	hodnota od 0 do 100
SENSOR_MODE_FAHRENHEIT	IN_MODE_FAHRENHEIT	stupne F
SENSOR_MODE_CELSIUS	IN_MODE_CELSIUS	stupne C
SENSOR_MODE_ROTATION	IN_MODE_ANGLESTEP	otáčky (16 hodnôt na 1 otáčku)

Tabuľka 9. Konštanty senzorových módov

Typ a mód senzora možno nastaviť naraz pomocou funkcie SetSensor.

Konfigurácia senzora	Typ	Mód
SENSOR_TOUCH	SENSOR_TYPE_TOUCH	SENSOR_MODE_BOOL
SENSOR_LIGHT	SENSOR_TYPE_LIGHT	SENSOR_MODE_PERCENT
SENSOR_ROTATION	SENSOR_TYPE_ROTATION	SENSOR_MODE_ROTATION
SENSOR_CELSIUS	SENSOR_TYPE_TEMPERATURE	SENSOR_MODE_CELSIUS
SENSOR_FAHRENHEIT	SENSOR_TYPE_TEMPERATURE	SENSOR_MODE_FAHRENHEIT
SENSOR_PULSE	SENSOR_TYPE_TOUCH	SENSOR_MODE_PULSE
SENSOR_EDGE	SENSOR_TYPE_TOUCH	SENSOR_MODE_EDGE

Tabuľka 10. Konštanty konfigurácie senzorov

Položka	Význam
Type	Typ senzora (pozri Tabuľku 8).
InputMode	Mód senzora (pozri Tabuľku 9).
RawValue	Surová nespracovaná hodnota senzora
NormalizedValue	Normalizovaná hodnota senzora
ScaledValue	Škálované hodnota senzora
InvalidData	Ak je nastavená, aktuálna hodnota senzora je neplatná

Tabuľka 11. Položky popisu senzora

SetSensor(port, konfigurácia) Procedúra

Nastaví typ a mód senzora na zadanú konfiguráciu (konštanta). Port môže byť zadaný pomocou konštanty (napr. `S1`, `S2`, `S3`, alebo `S4`) alebo premennej.

```
SetSensor(S1, SENSOR_TOUCH);
```

SetSensorType(port, typ) Procedúra

Nastaví typ senzora, ktorý musí byť jedna z preddefinovaných konštánt. Port môže byť zadaný pomocou konštanty (napr. `S1`, `S2`, `S3`, alebo `S4`) alebo premennej.

```
SetSensorType(S1, SENSOR_TYPE_TOUCH);
```

SetSensorMode(port, mód) Procedúra

Nastaví mód senzora, ktorý musí byť jedna z preddefinovaných konštánt. „Slope“ môže byť pridaný do argumentu mód. Port môže byť zadaný pomocou konštanty (napr. `S1`, `S2`, `S3`, alebo `S4`) alebo premennej.

```
SetSensorMode(S1, SENSOR_MODE_RAW); // surový, nespracovaný mód
SetSensorMode(S1, SENSOR_MODE_RAW + 10); // slope 10
```

SetSensorLight(port) Procedúra

Nastaví senzor na zadanom porte ako svetelný aktívny senzor. Configure the sensor on the specified port as a light sensor (active). Port môže byť zadaný pomocou konštanty (napr. `S1`, `S2`, `S3`, alebo `S4`) alebo premennej.

```
SetSensorLight(S1);
```

SetSensorSound(port) Procedúra

Nastaví senzor na zadanom porte ako zvukový senzor s dB škálou. Port môže byť zadaný pomocou konštanty (napr. `S1`, `S2`, `S3`, alebo `S4`) alebo premennej.

```
SetSensorSound(S1);
```

SetSensorTouch(port) Procedúra

Nastaví senzor na zadanom porte ako dotykový senzor Port môže byť zadaný pomocou konštanty (napr. `S1`, `S2`, `S3`, alebo `S4`) alebo premennej.

```
SetSensorTouch(S1);
```

SetSensorLowspeed(port) Procedúra

Nastaví senzor na zadanom porte ako I2C digitálny senzor (s 9V napájaním). Port môže byť zadaný pomocou konštanty (napr. `S1`, `S2`, `S3`, alebo `S4`) alebo premennej.

```
SetSensorLowspeed(S1);
```

SetInput(port, položka, hodnota) Procedúra

Nastaví zadanú položku popisu senzora na zadanom porte na príslušnú hodnotu. Port môže byť zadaný pomocou konštanty (napr. `S1`, `S2`, `S3`, alebo `S4`) alebo premennej. Položka musí byť jedna z položiek uvedených v tabuľke 11. Hodnota môže byť ľubovoľný prijateľný výraz.

```
SetInput(S1, Type, IN_TYPE_SOUND_DB);
```

ClearSensor(const port) Procedúra

Vynuluje hodnotu senzora – ovplyvňuje iba tie senzory, ktoré sú nastavené na meranie akumulovanej veličiny, napr. počet otáčok alebo pulzov. Port môže byť zadaný pomocou konštanty (napr. `S1`, `S2`, `S3`, alebo `S4`) alebo premennej.

```
ClearSensor(S1);
```

ResetSensor(port) Procedúra

Inicializuje senzor. Po zmenení typu a módu senzora ho treba inicializovať, aby boli hodnoty, ktoré sú zo senzora čítané platné. Port môže byť zadaný pomocou konštanty (napr. `S1`, `S2`, `S3`, alebo `S4`) alebo premennej.

```
ResetSensor(x); // x = S1
```

SetCustomSensorZeroOffset(const p, hodnota) Procedúra

Nastavuje posun (offset) pre nulovú hodnotu pre iný senzor (CUSTOM). Port môže byť zadaný pomocou konštanty (napr. `S1`, `S2`, `S3`, alebo `S4`).

```
SetCustomSensorZeroOffset(S1, 12);
```

SetCustomSensorPercentFullScale(const p, hodnota) Procedúra

Nastavuje hodnotu maxima škály iného senzora (CUSTOM). Port môže byť zadaný pomocou konštanty (napr. `S1`, `S2`, `S3`, alebo `S4`).

```
SetCustomSensorPercentFullScale(S1, 100);
```

SetCustomSensorActiveStatus(const p, hodnota) Procedúra

Nastaví hodnotu aktívneho stavu pre iný senzor (CUSTOM). Port môže byť zadaný pomocou konštanty (napr. `S1`, `S2`, `S3`, alebo `S4`).

```
SetCustomSensorActiveStatus(S1, true);
```

SetSensorDigiPinsDirection(const p, hodnota) Procedúra

Nastaví smer digitálnych pinov senzora. Port musí byť konštanta (napr. `S1`, `S2`, `S3`, alebo `S4`). Hodnota 1 nastaví smer pinov na výstupný, hodnota 0 nastaví smer na vstupný.

```
SetSensorDigiPinsDirection(S1, 1);
```

SetSensorDigiPinsStatus(const p, hodnota) Procedúra

Nastaví stav digitálnych pinov senzora. Port musí byť konštanta (napr. `S1`, `S2`, `S3`, alebo `S4`).

```
SetSensorDigiPinsStatus(S1, false);
```

SetSensorDigiPinsOutputLevel(const p, hodnota) Procedúra

Nastaví výstupnú úroveň digitálnych pinov senzora. Port musí byť konštanta (napr. `S1`, `S2`, `S3`, alebo `S4`).

```
SetSensorDigiPinsOutputLevel(S1, 100);
```

3.2.2 Informácie o senzoroch

O každom senzore je k dispozícii množstvo informácií. Vo všetkých prípadoch musí byť port senzora zadaný ako konštanta (napr. `S1`, `S2`, `S3`, alebo `S4`), ak nie je uvedené inak.

Sensor(n) Operácia

Vráti spracovanú hodnotu senzora na porte n, kde n je 0, 1, 2, alebo 3 (alebo konštanta určujúca meno portu senzora). Toto je rovnaká hodnota ako vracajú mená senzorov (napr. `SENSOR_1`). V tomto prípade možno použiť ako číslo portu aj premennú.

```
x = Sensor(S1); // čítaj senzor na porte 1
```

SensorUS(n) Operácia

Vráti nameranú hodnotu z ultrazvukového senzora na porte n, kde n je 0, 1, 2, alebo 3 (alebo konštanta určujúca meno portu senzora). Keďže ultrazvukový senzor je I2C digitálny senzor, jeho hodnota sa nedá čítať pomocou štandardného volania Sensor(n). V tomto prípade možno použiť ako číslo portu aj premennú.

```
x = SensorUS(S4); // čítaj ultrazvukový senzor na porte 4
```

SensorType(n) Operácia

Vráti nastavený typ senzora na porte n, kde n je 0, 1, 2, alebo 3 (alebo konštanta určujúca meno portu senzora). V tomto prípade možno použiť ako číslo portu aj premennú.

```
x = SensorType(S1);
```

SensorMode(n) Operácia

Vráti nastavený mód senzora na porte n, kde n je 0, 1, 2, alebo 3 (alebo konštanta určujúca meno portu senzora). V tomto prípade možno použiť ako číslo portu aj premennú.

```
x = SensorMode(S1);
```

SensorRaw(n) Operácia

Vráti nespracovanú hodnotu senzora na porte n, kde n je 0, 1, 2, alebo 3 (alebo konštanta určujúca meno portu senzora). V tomto prípade možno použiť ako číslo portu aj premennú.

```
x = SensorRaw(S1);
```

SensorNormalized(n) Operácia

Vráti normalizovanú hodnotu senzora na porte n, kde n je 0, 1, 2, alebo 3 (alebo konštanta určujúca meno portu senzora). V tomto prípade možno použiť ako číslo portu aj premennú.

```
x = SensorNormalized(S1);
```

SensorScaled(n) Operácia

Vráti nastavený typ senzora na porte n, kde n je 0, 1, 2, alebo 3 (alebo konštanta určujúca meno portu senzora). V tomto prípade možno použiť ako číslo portu aj premennú. Toto volanie je identické so štandardným volaním Sensor(n).

```
x = SensorScaled(S1);
```

SensorInvalid(n) Operácia

Vráti neplatnosť hodnoty senzora (flag InvalidData) na porte n, kde n je 0, 1, 2, alebo 3 (alebo konštanta určujúca meno portu senzora). V tomto prípade možno použiť ako číslo portu aj premennú.

```
x = SensorInvalid(S1);
```

SensorBoolean(const n) Operácia

Vráti logickú hodnotu senzora na porte n, kde n je 0, 1, 2, alebo 3 (alebo konštanta určujúca meno portu senzora). Logická konverzia sa vykonáva buď na základe predprogramovaného prahu alebo na základe parametra slope zadaného pri volaní `SetSensorMode`.

```
x = SensorBoolean(S1);
```

GetInput(n, const field) Operácia

Vráti požadovanú hodnotu položky popisu senzora na porte n, kde n je 0, 1, 2, alebo 3 (alebo konštanta určujúca meno portu senzora). V tomto prípade možno použiť ako číslo portu aj premennú. Možné položky popisu senzora sú v tabuľke 11.

```
x = GetInput(S1, Type);
```

CustomSensorZeroOffset(const p) Operácia

Nastaví hodnotu nulového posunu (offset) pre iný senzor (CUSTOM) na porte p, ktorý musí byť 0, 1, 2, alebo 3 (alebo meno senzorovej konštanty).

```
x = CustomSensorZeroOffset(S1);
```

CustomSensorPercentFullScale(const p) Operácia

Vráti škálovanú hodnotu iného senzora (CUSTOM) na porte n, kde n je 0, 1, 2, alebo 3 (alebo konštanta určujúca meno portu senzora).

```
x = CustomSensorPercentFullScale(S1);
```

CustomSensorActiveStatus(const p) Operácia

Vráti stav aktivity iného senzora (CUSTOM) na porte n, kde n je 0, 1, 2, alebo 3 (alebo konštanta určujúca meno portu senzora).

```
x = CustomSensorActiveStatus(S1);
```

SensorDigiPinsDirection(const p) Operácia

Vráti nastavený smer digitálnych pinov senzora na porte p, ktorý musí byť 0, 1, 2, alebo 3 (alebo konštanta určujúca meno portu senzora).

```
x = SensorDigiPinsDirection(S1);
```

SensorDigiPinsStatus(const p) Operácia

Vráti stav digitálnych pinov senzora na porte p, ktorý musí byť 0, 1, 2, alebo 3 (alebo konštanta určujúca meno portu senzora).

```
x = SensorDigiPinsStatus(S1);
```

SensorDigiPinsOutputLevel(const p) Operácia

Vráti výstupnú úroveň digitálnych pinov senzora na porte p, ktorý musí byť 0, 1, 2, alebo 3 (alebo konštanta určujúca meno portu senzora).

```
x = SensorDigiPinsOutputLevel(S1);
```

3.2.3 Adresy položiek štruktúry IOMap (IOMap Offsets)

Offset modulu Input	Hodnota	Veľkosť
InputOffsetCustomZeroOffset(p)	(((p)*20)+0)	2
InputOffsetADRaw(p)	(((p)*20)+2)	2
InputOffsetSensorRaw(p)	(((p)*20)+4)	2
InputOffsetSensorValue(p)	(((p)*20)+6)	2
InputOffsetSensorType(p)	(((p)*20)+8)	1
InputOffsetSensorMode(p)	(((p)*20)+9)	1
InputOffsetSensorBoolean(p)	(((p)*20)+10)	1
InputOffsetDigiPinsDir(p)	(((p)*20)+11)	1
InputOffsetDigiPinsIn(p)	(((p)*20)+12)	1
InputOffsetDigiPinsOut(p)	(((p)*20)+13)	1
InputOffsetCustomPctFullScale(p)	(((p)*20)+14)	1
InputOffsetCustomActiveStatus(p)	(((p)*20)+15)	1
InputOffsetInvalidData(p)	(((p)*20)+16)	1

Tabuľka 12. Položky IOMap pre modul Input

3.3 Modul Output

Modul Output sa stará o riadenie motorov.

Konštanta	Hodnota
OutputModuleName	"Output.mod"
OutputModuleID	0x00020001

Tabuľka 13. Konštanty modulu Output

Takmer všetky funkcie NXC API, ktoré pracujú s výstupmi majú prvý argument, ktorý určuje výstup alebo skupinu výstupov, ktorých sa operácia týka. V závislosti od toho, ktorej funkcie sa to týka možno v tomto argumente použiť konštantu alebo aj premennú. Konštanty `OUT_A`, `OUT_B`, a `OUT_C` určujú jednotlivé výstupné porty. Ak chceme určiť, že operácia sa týka dvoch alebo troch výstupných portov, tieto konštanty (narozdiel od NQC) nemožno sčítavať. Namiesto toho NXC definuje preddefinované konštanty, ktoré popisujú jednotlivé kombinácie výstupov: `OUT_AB`,

OUT_AC, OUT_BC, a OUT_ABC. V skutočnosti tieto konštanty obsahujú polia a preto ručne ich môžeme vyrobiť tak, že pridáme dve alebo tri jednotlivé výstupné konštanty do poľa.

Napäťová úroveň výstupu sa pohybuje v rozmedzí od 0 (najnižšie) po 100 (najvyššie). Negatívne hodnoty je možné použiť na zmenu smeru otáčania motorov (čiže dopredu rýchlosťou -100 znamená cúvať rýchlosťou 100).

Každý z výstupov je popísaný niekoľkými položkami, ktoré popisujú jeho aktuálny stav, sú uvedené v nasledujúcej tabuľke:

Položka	Typ	Prístup	Rozsah	Význam
UpdateFlags	ubyte	Čitanie/ Zápis	0,26	Toto pole môže obsahovať ľubovoľnú kombináciu príznakov popísaných v tabuľke 15. Použitím príznakov UF_UPDATE_MODE, UF_UPDATE_SPEED, UF_UPDATE_TACHO_LIMIT, a UF_UPDATE_PID_VALUES spolu s ostatnými poliami uloží zmeny do výstupného modulu. Po nastavení jednotlivých výstupných polí je možné nastaviť prislušný príznak, aby zmena bola potvrdená a účinná.
OutputMode	ubyte	Čitanie/ Zápis	0,26	Toto bitové pole môže obsahovať ľubovoľné hodnoty popísané v tabuľke 16. Bit OUT_MODE_MOTORON musí byť nastavený, aby bolo na motor privedené napájanie. Pridaním OUT_MODE_BRAKE sa zapne elektronická brzda, ktorá motory aktívne zabrzdí. Takto je možné dosiahnuť väčšiu presnosť, ale zvyšuje sa tým spotreba prúdu. Regulácia motora sa zapína príznakom OUT_MODE_REGULATED. Zmeny v tomto poli je potrebné potvrdiť zapísaním príznaku UF_UPDATE_MODE do poľa UpdateFlags.
Power	sbyte	Čitanie/ Zápis	-100,100	Určuje napätie na výstupe. Absolútna hodnota tohto poľa určuje percentuálnu časť maximálneho možného napätia na motore. Znamienko tohto poľa určuje smer otáčania motora. Pri kladných hodnotách sa motor otáča vpred, pri záporných sa otáča vzad. Zmeny v tomto poli je potrebné potvrdiť zapísaním príznaku UF_UPDATE_POWER do poľa UpdateFlags.
ActualSpeed	sbyte	Čítanie	-100,100	Obsahuje percentuálnu časť maximálneho napätia, ktoré riadiaci program NXT posiela na výstup. Táto hodnota sa môže líšiť od hodnoty Power, keď riadiaci program reaguje na záťaž na výstupe.
TachoCount	slong	Čítanie	úplný rozsah signed long	Vráti pozíciu interného počítadla pre zadaný výstup. Interné počítadlo sa automaticky nuluje potom, ako je nastavená nová cieľová hodnota v poli TachoLimit a je potvrdená príznakom UF_UPDATE_TACHO_LIMIT. Túto hodnotu je možné vynulovať zápisom príznaku UF_UPDATE_RESET_COUNT do poľa UpdateFlags, čím sa zároveň zruší TachoLimit. Znamienko poľa TachoCount určuje smer otáčania motora.
TachoLimit	ulong	Čitanie/ Zápis	úplný rozsah unsigned long	Určuje počet stupňov o ktoré sa motor má otočiť. Zmeny v tomto poli treba potvrdiť zápisom UF_UPDATE_TACHO_LIMIT do UpdateFlags. Hodnota tohto poľa určuje relatívnu vzdialenosť od aktuálnej pozície motora v momente, keď sa potvrdí zápisom príznaku UF_UPDATE_TACHO_LIMIT.
RunState	ubyte	Čitanie/ Zápis	0..255	Určuje stav behu motora. Nastavením poľa na hodnotu OUT_RUNSTATE_RUNNING sa motor zapne. Nastavením hodnoty OUT_RUNSTATE_RAMPUP sa zapne automatické postupné zrýchlovanie (ramping up) na novú úroveň, ktorá bude vyššia ako súčasná úroveň. Nastavením OUT_RUNSTATE_RAMPDOWN sa zapne automatické

Položka	Typ	Prístup	Rozsah	Význam
				spomaľovanie na novú úroveň, ktorá bude nižšia ako aktuálna. Oba príznaky musia byť použité v súhre s poliami TachoLimit a Power. Postupné zrýchlovanie/spomaľovanie bude trvať toľko stupňov, ako je určené poľom TachoLimit.
TurnRatio	sbyte	Čitanie/ Zápis	-100,100	Určuje proporcionálny pomer otáčania. Toto pole sa používa v kombinácii s inými hodnotami: OutputMode musí obsahovať OUT_MODE_MOTORON a OUT_MODE_REGULATED, RegMode, musí byť nastavená na OUT_REGMODE_SYNC, RunState nesmie byť OUT_RUNSTATE_IDLE a Speed musí byť nenulová. Sú len tri platné kombinácie výstupov s ktorými je možné použiť TurnRatio: OUT_AB, OUT_BC, a OUT_AC. Vo všetkých troch je za ľavý motor považovaný prvý z nich. Záporné hodnoty TurnRatio presúvajú silu na ľavý motor a kladné presúvajú silu na pravý motor. Absolútna hodnota 50 väčšinou spôsobí, že jeden z motorov sa zastaví. Absolútna hodnota 100 väčšinou znamená, že motory sa točia opačným smerom a rovnakou rýchlosťou.
RegMode	ubyte	Čitanie/ Zápis	0..255	Pole určuje spôsob regulácie na špecifikovanom porte alebo portoch. Ak príznak OUT_MODE_REGULATED v poli OutputMode nie je nastavený, hodnota tohto poľa je ignorovaná. Na rozdieľ od poľa OutputMode, pole RegMode nie je bitovým poľom, čiže iba jedna z prípustných hodnôt môže byť použitá naraz (nie ich súčtové kombinácie). Prípustné hodnoty pre toto pole sú zobrazené v tabuľke 18. Regulácia rýchlosti (Speed regulation) znamená, že riadiaci program sa pokúša udržovať rýchlosť zadanú v poli Power: v prípade väčšej záťaže upravuje napätie na výstupe a toto upravené napätie je možné prečítať v poli ActualSpeed. V prípade regulácie rýchlosti nemá zmysel nastaviť Power na maximum, pretože riadiaci program potom nemá ako zabezpečiť reguláciu rýchlosti. V prípade synchronizácie sa riadiaci program usiluje udržovať otáčky oboch motorov v súlade, nezávisle od fyzickej záťaže. Tento režim je vhodný napríklad na docielenie pohybu robota v priamom smere. Synchronizačný režim je možné kombinovať s hodnotou v poli TurnRatio na zabezpečenie stanoveného polomeru otáčania. Na aktivovanie synchronizácie treba zapísať hodnotu OUT_REGMODE_SYNC aspoň na dvoch portoch. V prípade, že je OUT_REGMODE_SYNC nastavená na všetkých troch, synchronizovať sa budú prvé dva (OUT_A a OUT_B).
Overload	ubyte	Čítanie	0..1	Toto pole je nastavené na 1 (true) ak riadiaci program nemôže prekonať fyzickú záťaž motora. Inými slovami, motor sa otáča pomalšie ako by sa mal. Ak je požadovaná rýchlosť motora docielená napriek zvýšenej záťaži, tak hodnota tohto poľa je nula (false). Toto pole je platné, len ak RunState nie je IDLE, OutputMode je OUT_MODE_MOTORON alebo OUT_MODE_REGULATED, a RegMode je nastavený na OUT_REGMODE_SPEED.
RegPValue	ubyte	Čitanie/ Zápis	0..255	Toto pole určuje proporcionálnu zložku v internom PID riadiacom algoritme (proportional-integral-derivative). Nastavenie tejto hodnoty (a súčasne hodnôt RegIValue a RegDValue) treba potvrdiť zápisom príznaku UF_UPDATE_PID_VALUES do UpdateFlags.
RegIValue	ubyte	Čitanie/ Zápis	0..255	Toto pole určuje integrálnu zložku v internom PID riadiacom algoritme (proportional-integral-derivative). Nastavenie tejto

Položka	Typ	Prístup	Rozsah	Význam
				hodnoty (a súčasne hodnôt RegPValue a RegDValue) treba potvrdiť zápisom príznaku UF_UPDATE_PID_VALUES do UpdateFlags.
RegDValue	ubyte	Čitanie/ Zápis	0..255	Toto pole určuje derivačnú zložku v internom PID riadiacom algoritme (proportional-integral-derivative). Nastavenie tejto hodnoty (a súčasne hodnôt RegPValue a RegIValue) treba potvrdiť zápisom príznaku UF_UPDATE_PID_VALUES do UpdateFlags.
BlockTachoCount	slong	Čítanie	úplný rozsah signed long	Vráti hodnotu počítadla blokovo-relatívnej pozície na príslušnom porte. Pozri popis UpdateFlags. Nastavením príznaku UF_UPDATE_RESET_BLOCK_COUNT v UpdateFlags riadiaci program vynuluje pole BlockTachoCount. Znamienko poľa BlockTachoCount určuje smer otáčania. Kladné hodnoty určujú doprednú rotáciu a záporné hodnoty určujú spätnú rotáciu (v závislosti na natočení motora).
RotationCount	slong	Čítanie	Úplný rozsah signed long	Vráti hodnotu počítadla programovo-relatívnej pozície na príslušnom porte. Pozri popis UpdateFlags. Nastavením príznaku UF_UPDATE_RESET_ROTATION_COUNT v UpdateFlags riadiaci program vynuluje RotationCount. Znamienko poľa RotationCount určuje smer otáčania. Kladné hodnoty určujú doprednú rotáciu a záporné hodnoty určujú spätnú rotáciu (v závislosti na natočení motora).

Tabuľka 14. Položky popisu výstupných portov

Prípustné hodnoty pre pole UpdateFlags sú zobrazené v nasledujúcej tabuľke.

Konštanta poľa UpdateFlags	Význam
UF_UPDATE_MODE	Potvrdí zmeny v poli OutputMode
UF_UPDATE_SPEED	Potvrdí zmeny v poli Power
UF_UPDATE_TACHO_LIMIT	Potvrdí zmeny v poli TachoLimit
UF_UPDATE_RESET_COUNT	Vynuluje všetky počítadlá otáčok, zruší aktuálny cieľ a inicializuje systém na opravu chyby otáčok
UF_UPDATE_PID_VALUES	Potvrdí zmeny parametrov PID regulátora motora
UF_UPDATE_RESET_BLOCK_COUNT	Vynuluje blokovo-relatívne počítadlo otáčok
UF_UPDATE_RESET_ROTATION_COUNT	Vynuluje programovo-relatívne počítadlo otáčok

Tabuľka 15. Konštanty poľa UpdateFlag

Prípustné hodnoty poľa OutputMode sú popísané v nasledujúcej tabuľke.

Konštanta poľa OutputMode	Hodnota	Význam
OUT_MODE_COAST	0x00	Žiadne napätie ani brzdenie, motory sa dajú ručne voľne otáčať
OUT_MODE_MOTORON	0x01	Zapne motor na silu podľa hodnoty v poli Power
OUT_MODE_BRAKE	0x02	Elektronická brzda
OUT_MODE_REGULATED	0x04	Zapne reguláciu motora v závislosti na nastavení poľa RegMode
OUT_MODE_REGMETHOD	0xf0	

Tabuľka 16. Konštanty poľa OutputMode

Prípustné hodnoty poľa RunState sú popísané v nasledujúcej tabuľke.

Konštanta poľa RunState	Hodnota	Význam
OUT_RUNSTATE_IDLE	0x00	Vypne napájanie motora
OUT_RUNSTATE_RAMPUP	0x10	Zapne postupné zrýchlovanie na novú úroveň Power počas doby stanovenej poľom TachoLimit
OUT_RUNSTATE_RUNNING	0x20	Zapne napájanie motora na úrovni stanovenej poľom Power
OUT_RUNSTATE_RAMPDOWN	0x40	Zapne postupné spomaľovanie na novú úroveň Power počas doby stanovenej poľom TachoLimit

Tabuľka 17. Konštanty poľa RunState

Prípustné hodnoty poľa RegMode sú popísané v nasledujúcej tabuľke.

Konštanta poľa RegMode	Hodnota	Význam
OUT_REGMODE_IDLE	0x00	Bez regulácie
OUT_REGMODE_SPEED	0x01	Regulácia rýchlosti (Power)
OUT_REGMODE_SYNC	0x02	Synchronizácia otáčok dvoch motorov

Tabuľka 18. KonŠtanty poľa RegMode

3.3.1 Pomocné volania

Keďže riadenie výstupov je jedna z najbežnejších operácií v programoch, NXT API poskytuje viacero zjednodušujúcich pomocných volaní. Žiadne z nich neposkytuje žiadnu funkciu nad rámec nízkoúrovňových funkcií popísaných v nasledujúcej stati. Sú zavedené len na zvýšenie prehľadnosti programov a jednoduchšie použitie.

Verzie funkcií s príponou Ex používajú špeciálne resetovacie konštanty, ktoré sú uvedené v nasledujúcej tabuľke. Funkcie označené Var vyžadujú, že argument Výstupy je premenná, zakiaľ čo ostatné funkcie vyžadujú, aby tento argument bola konštanta.

Off(výstupy) **Procedúra**

Vypne zadané výstupy a zapne elektronickú brzdu. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami.

```
Off(OUT_A); // zastav motor na porte A
```

OffEx(výstupy, const reset) **Procedúra**

Vypne zadané výstupy a zapne elektronickú brzdu. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami. Argument reset určuje, či sa niektoré z troch otáčkových počítadiel má vynulovať a musí byť zadaný ako konštanta. Prípustné hodnoty sú v tabuľke 19.

```
OffEx(OUT_A, RESET_NONE); // zastav motor na porte A
```

Coast(výstupy) **Procedúra**

Vypne zadané výstupy a nechá ich voľne sa otáčať, kým sa zastavia. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami.

```
Coast(OUT_A); // vypni motor na porte A
```

Restovacia konštanta	Hodnota
RESET_NONE	0x00
RESET_COUNT	0x08
RESET_BLOCK_COUNT	0x20
RESET_ROTATION_COUNT	0x40
RESET_BLOCKANDTACHO	0x28
RESET_ALL	0x68

Tabuľka 19. Resetovacie konštanty

Konštanta pre výstupný port	Hodnota
OUT_A	0x00
OUT_B	0x01
OUT_C	0x02
OUT_AB	0x03
OUT_AC	0x04
OUT_BC	0x05
OUT_ABC	0x06

Tabuľka 20. Konštanty pre argument určujúci výstupný port

CoastEx(výstupy, const reset) Procedúra

Vypne zadané výstupy a nechá ich voľne sa otáčať, kým sa zastavia. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami. Argument reset určuje, či sa niektoré z troch otáčkových počítadiel má vynulovať a musí byť zadaný ako konštanta. Prípustné hodnoty sú v tabuľke 19.

```
CoastEx(OUT_A, RESET_NONE); // vypni motor na porte A
```

Float(výstupy) Procedúra

Vypne zadané výstupy a nechá ich voľne sa otáčať, kým sa zastavia. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami. Float je druhé meno pre Coast.

```
Float(OUT_A); // float output A
```

OnFwd(výstupy, sila) Procedúra

Nastaví výstupy smerom vpred, na zadanú silu a zapne ich. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami.

```
OnFwd(OUT_A, 75);
```

OnFwdEx(výstupy, sila, const reset) Procedúra

Nastaví výstupy smerom vpred, na zadanú silu a zapne ich. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami. Argument reset určuje, či sa niektoré z troch otáčkových

počítadiel má vynulovať a musí byť zadaný ako konštanta. Prípustné hodnoty sú v tabuľke 19.

```
OnFwdEx(OUT_A, 75, RESET_NONE);
```

OnRev(výstupy, sila) Procedúra

Výstupy nastaví na spätný chod, zadanú silu a zapne ich. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami.

```
OnRev(OUT_A, 75);
```

OnRevEx(výstupy, sila, const reset) Procedúra

Výstupy nastaví na spätný chod, zadanú silu a zapne ich. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami. Argument reset určuje, či sa niektoré z troch otáčkových počítadiel má vynulovať a musí byť zadaný ako konštanta. Prípustné hodnoty sú v tabuľke 19.

```
OnRevEx(OUT_A, 75, RESET_NONE);
```

OnFwdReg(výstupy, sila, regmod) Procedúra

Nastaví výstupy smerom vpred, na zadanú silu a zapne ich, pričom aktivuje požadovaný regulačný režim. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami. Prípustné regulačné režimy sú v tabuľke 18.

```
OnFwdReg(OUT_A, 75, OUT_REGMODE_SPEED); // regulácia rýchlosti
```

OnFwdRegEx(výstupy, sila, regmod, const reset) Procedúra

Nastaví výstupy smerom vpred, na zadanú silu a zapne ich, pričom aktivuje požadovaný regulačný režim. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami. Prípustné regulačné režimy sú v tabuľke 18. Argument reset určuje, či sa niektoré z troch otáčkových počítadiel má vynulovať a musí byť zadaný ako konštanta. Prípustné hodnoty sú v tabuľke 19.

```
OnFwdRegEx(OUT_A, 75, OUT_REGMODE_SPEED, RESET_NONE);
```

OnRevReg(výstupy, sila, regmod) Procedúra

Nastaví výstupy na spätný chod, na zadanú silu a zapne ich, pričom aktivuje požadovaný regulačný režim. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami. Prípustné regulačné režimy sú v tabuľke 18.

```
OnRevReg(OUT_A, 75, OUT_REGMODE_SPEED); // regulácia rýchlosti
```

OnRevRegEx(výstupy, sila, regmod, const reset) Procedúra

Nastaví výstupy na spätný chod, na zadanú silu a zapne ich, pričom aktivuje požadovaný regulačný režim. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami. Prípustné regulačné režimy sú v tabuľke 18. Argument reset určuje, či sa niektoré z troch otáčkových počítadiel má vynulovať a musí byť zadaný ako konštanta. Prípustné hodnoty sú v tabuľke 19.

```
OnRevRegEx(OUT_A, 75, OUT_REGMODE_SPEED, RESET_NONE);
```

OnFwdSync(výstupy, sila, smerPohybu) Procedúra

Nastaví výstupy smerom vpred, na zadaný smer pohybu a zapne ich, pričom aktivuje synchronizačný režim. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami.

```
OnFwdSync(OUT_AB, 75, -100); // otáčaj sa na mieste vpravo
```

OnFwdSyncEx(výstupy, sila, smerPohybu, const reset) Procedúra

Nastaví výstupy smerom vpred, na zadaný smer pohybu a zapne ich, pričom aktivuje synchronizačný režim. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami. Argument reset určuje, či sa niektoré z troch otáčkových počítadiel má vynulovať a musí byť zadaný ako konštanta. Prípustné hodnoty sú v tabuľke 19.

```
OnFwdSyncEx(OUT_AB, 75, 0, RESET_NONE);
```

OnRevSync(výstupy, sila, smerPohybu) Procedúra

Nastaví výstupy smerom vzad, na zadaný smer pohybu a zapne ich, pričom aktivuje synchronizačný režim. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami.

```
OnRevSync(OUT_AB, 75, -100); // otáčaj sa na mieste vľavo
```

OnRevSyncEx(výstupy, sila, smerPohybu, const reset) Procedúra

Nastaví výstupy smerom vzad, na zadaný smer pohybu a zapne ich, pričom aktivuje synchronizačný režim. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami. Argument reset určuje, či sa niektoré z troch otáčkových počítadiel má vynulovať a musí byť zadaný ako konštanta. Prípustné hodnoty sú v tabuľke 19.

```
OnRevSyncEx(OUT_AB, 75, -100, RESET_NONE); // točsa na mieste vľavo
```

RotateMotor(výstupy, sila, uhol) Procedúra

Zapne zadané výstupy smerom vpred na zadanú silu a nechá ich bežať zadaný počet otáčkových stupňov. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami.

```
RotateMotor(OUT_A, 75, 45); // dopredu 45 stupňov
RotateMotor(OUT_A, -75, 45); // vzad 45 stupňov
```

RotateMotorPID(výstupy, sila, uhol, p, i, d) Procedúra

Zapne zadané výstupy smerom vpred na zadanú silu a nechá ich bežať zadaný počet otáčkových stupňov. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami. Táto funkcia tiež nastaví parametre pre riadiaci program, ktorý používa algoritmus PID na riadenie motorov.

```
RotateMotorPID(OUT_A, 75, 45, 20, 40, 100);
```

RotateMotorEx(výstupy, sila, uhol, smerPohybu, sync, stop) Procedúra

Zapne zadané výstupy smerom vpred na zadanú silu a nechá ich bežať zadaný počet otáčkových stupňov. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami. Ak je smerPohybu nenulový, sync musí

byť nastavený na true, inak otáčanie nenastane. Posledný argument určuje, či na konci majú byť motory zastavené elektronickou brzdou.

```
RotateMotorEx(OUT_AB, 75, 360, 50, true, true);
```

RotateMotorExPID(výstupy, sila, uhol, smerPohyb, sync, stop, p, i, d) Procedúra

Zapne zadané výstupy smerom vpred na zadanú silu a nechá ich bežať zadaný počet otáčkových stupňov. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami. Ak je smerPohybu nenulový, sync musí byť nastavený na true, inak otáčanie nenastane. Posledný argument určuje, či na konci majú byť motory zastavené elektronickou brzdou. Táto funkcia tiež nastaví parametre pre riadiaci program, ktorý používa algoritmus PID na riadenie motorov.

```
RotateMotorExPID(OUT_AB, 75, 360, 50, true, true, 30, 50, 90);
```

ResetTachoCount(výstupy) Procedúra

Vynuluje počítadlo otáčok TachoCount a cieľ tachometra TachoLimit pre všetky zadané výstupy. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami.

```
ResetTachoCount(OUT_AB);
```

ResetBlockTachoCount(výstupy) Procedúra

Vynuluje blokovo-relatívne počítadlo otáčok BlockTachoCount pre všetky zadané výstupy. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami.

```
ResetBlockTachoCount(OUT_AB);
```

ResetRotationCount(výstupy) Procedúra

Vynuluje programovo-relatívne počítadlo otáčok RotationCount pre všetky zadané výstupy. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami.

```
ResetRotationCount(OUT_AB);
```

ResetAllTachoCounts(výstupy) Procedúra

Vynuluje všetky tri počítadlá otáčok a cieľ tachometra TachoLimit pre všetky zadané výstupy. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami.

```
ResetAllTachoCounts(OUT_AB);
```

3.3.2 Nízkoúrovňové volania

SetOutput(výstupy, const pole1, hodn1, …, const poleN, hodnN) Procedúra

Nastaví požadované pole alebo polia na dané hodnoty. Argument výstupy môže byť konštanta alebo premenná, ktorá obsahuje požadované porty, pozri tabuľku 20 s preddefinovanými konštantami. Pole musí byť prípustná konštanta popisujúca pole výstupného portu, sú uvedené v tabuľke 14. Táto funkcia akceptuje premenlivý počet argumentov.

```
SetOutput(OUT_AB, TachoLimit, 720); // nastav tacholimit
```

GetOutput(výstup, const pole) Operácia

Vráti hodnotu požadovaného poľa na jednom z výstupných portov. Výstup môže byť OUT_A, OUT_B, OUT_C, alebo premenná, ktorá obsahuje jednu z týchto hodnôt. Pole musí byť prípustná konštanta popisujúca pole výstupného portu, sú uvedené v tabuľke 14.

```
x = GetOutput(OUT_A, TachoLimit);
```

MotorMode(výstup) Operácia

Vráti mód na zadanom výstupnom porte. Výstup môže byť OUT_A, OUT_B, OUT_C, alebo premenná, ktorá obsahuje jednu z týchto hodnôt.

```
x = MotorMode(OUT_A);
```

MotorPower(výstup) Operácia

Vráti nastavenie sily na zadanom výstupnom porte. Výstup môže byť OUT_A, OUT_B, OUT_C, alebo premenná, ktorá obsahuje jednu z týchto hodnôt.

```
x = MotorPower(OUT_A);
```

MotorActualSpeed(výstup) Operácia

Zistí skutočnú silu privedenú na zadaný výstupný port (ak je aktívny režim regulácie rýchlosti). Výstup môže byť OUT_A, OUT_B, OUT_C, alebo premenná, ktorá obsahuje jednu z týchto hodnôt.

```
x = MotorActualSpeed(OUT_A);
```

MotorTachoCount(výstup) Operácia

Zistí hodnotu počítadla TachoCount na zadanom výstupnom porte. Výstup môže byť OUT_A, OUT_B, OUT_C, alebo premenná, ktorá obsahuje jednu z týchto hodnôt.

```
x = MotorTachoCount(OUT_A);
```

MotorTachoLimit(výstup) Operácia

Zistí hodnotu TachoLimit pre zadaný výstupný port. Výstup môže byť OUT_A, OUT_B, OUT_C, alebo premenná, ktorá obsahuje jednu z týchto hodnôt.

```
x = MotorTachoLimit(OUT_A);
```

MotorRunState(výstup) Operácia

Zistí hodnotu poľa RunState pre zadaný výstupný port. Výstup môže byť OUT_A, OUT_B, OUT_C, alebo premenná, ktorá obsahuje jednu z týchto hodnôt.

```
x = MotorRunState(OUT_A);
```

MotorTurnRatio(výstup) Operácia

Zistí hodnotu poľa TurnRatio pre zadaný výstupný port. Výstup môže byť OUT_A, OUT_B, OUT_C, alebo premenná, ktorá obsahuje jednu z týchto hodnôt.

```
x = MotorTurnRatio(OUT_A);
```

MotorRegulation(výstup) Operácia

Zistí hodnotu poľa RegMode pre zadaný výstupný port. Výstup môže byť OUT_A, OUT_B, OUT_C, alebo premenná, ktorá obsahuje jednu z týchto hodnôt.

```
x = MotorRegulation(OUT_A);
```

MotorOverload(výstup) Operácia

Zistí hodnotu poľa Overload na zadanom výstupnom porte. Výstup môže byť OUT_A, OUT_B, OUT_C, alebo premenná, ktorá obsahuje jednu z týchto hodnôt.

```
x = MotorOverload(OUT_A);
```

MotorRegPValue(výstup) Operácia

Zistí proporcionálnu zložku algoritmu PID pre zadaný výstupný port. Výstup môže byť OUT_A, OUT_B, OUT_C, alebo premenná, ktorá obsahuje jednu z týchto hodnôt.

```
x = MotorRegPValue(OUT_A);
```

MotorRegIValue(výstup) Operácia

Zistí integrálnu zložku algoritmu PID pre zadaný výstupný port. Výstup môže byť OUT_A, OUT_B, OUT_C, alebo premenná, ktorá obsahuje jednu z týchto hodnôt.

```
x = MotorRegIValue(OUT_A);
```

MotorRegDValue(výstup) Operácia

Zistí derivačnú zložku algoritmu PID pre zadaný výstupný port. Výstup môže byť OUT_A, OUT_B, OUT_C, alebo premenná, ktorá obsahuje jednu z týchto hodnôt.

```
x = MotorRegDValue(OUT_A);
```

MotorBlockTachoCount(výstup) Operácia

Zistí hodnotu počítadla otáčok BlockTachoCount pre zadaný výstupný port. Výstup môže byť OUT_A, OUT_B, OUT_C, alebo premenná, ktorá obsahuje jednu z týchto hodnôt.

```
x = MotorBlockTachoCount(OUT_A);
```

MotorRotationCount(výstup) Operácia

Zistí hodnotu počítadla otáčok RotationCount pre zadaný výstupný port. Výstup môže byť OUT_A, OUT_B, OUT_C, alebo premenná, ktorá obsahuje jednu z týchto hodnôt.

```
x = MotorRotationCount(OUT_A);
```

MotorPwnFreq() Operácia

Zistí aktuálnu frekvenciu PWM (pulse width modulation), ktorá sa vysiela na výstupné porty.

```
x = MotorPwnFreq();
```

SetMotorPwnFreq(hodnota) Procedúra

Nastaví frekvenciu PWM (pulse width modulation), ktorá sa vysiela na výstupné porty.

```
SetMotorPwnFreq(x);
```

3.3.3 Adresy položiek štruktúry IOMap (IOMap Offsets)

Offset Modulu Output	Hodnota	Veľkosť
OutputOffsetTachoCount(p)	(((p)*32)+0)	4
OutputOffsetBlockTachoCount(p)	(((p)*32)+4)	4
OutputOffsetRotationCount(p)	(((p)*32)+8)	4
OutputOffsetTachoLimit(p)	(((p)*32)+12)	4
OutputOffsetMotorRPM(p)	(((p)*32)+16)	2
OutputOffsetFlags(p)	(((p)*32)+18)	1
OutputOffsetMode(p)	(((p)*32)+19)	1
OutputOffsetSpeed(p)	(((p)*32)+20)	1
OutputOffsetActualSpeed(p)	(((p)*32)+21)	1
OutputOffsetRegPParameter(p)	(((p)*32)+22)	1
OutputOffsetRegIParameter(p)	(((p)*32)+23)	1
OutputOffsetRegDParameter(p)	(((p)*32)+24)	1
OutputOffsetRunState(p)	(((p)*32)+25)	1
OutputOffsetRegMode(p)	(((p)*32)+26)	1
OutputOffsetOverloaded(p)	(((p)*32)+27)	1
OutputOffsetSyncTurnParameter(p)	(((p)*32)+28)	1
OutputOffsetPwnFreq	96	1

Tabuľka 21. Položky IOMap pre modul Output

3.4 Adresy IO Map

Hodnoty polí popísujúcich vstupné a výstupné porty je možné čítať a zapisovať pomocou špeciálnych nízko-úrovňových konštnánt, ktoré sa označujú IO Map Adresy (IOMA). Platné konštanty IOMA sú zobrazené v tabuľke 22.

IOMA(const n) — Operácia

Vráti hodnotu z IO Map na zadanej adrese. Platné adresy sú uvedené v tabuľke 22.

```
x = IOMA(InputIORawValue(S3));
```

SetIOMA(const n, hodnota) — Procedúra

Nastaví položku na zadanej adrese v IO Map na stanovenú hodnotu. Platné adresy sú uvedené v tabuľke 22. Hodnota môže byť určená konštantou, konštantným výrazom alebo premennou.

```
SetIOMA(OutputIOPower(OUT_A), x);
```

Konštanta IOMA	Parameter	Význam
InputIOType(p)	S1..S4	Typ senzora
InputIOInputMode(p)	S1..S4	Mód senzora
InputIORawValue(p)	S1..S4	Nespracovaná surová hodnota prečítaná zo senzora
InputIONormalizedValue(p)	S1..S4	Normalizovaná hodnota prečítaná zo senzora
InputIOScaledValue(p)	S1..S4	Škálovaná hodnota prečítaná zo senzora
InputIOInvalidData(p)	S1..S4	Hodnota poľa InvalidData (neplatnosť hodnoty senzora)
OutputIOUpdateFlags(p)	OUT_A..OUT_C	Hodnota poľa UpdateFlags
OutputIOOutputMode(p)	OUT_A..OUT_C	Hodnota poľa OutputMode
OutputIOPower(p)	OUT_A..OUT_C	Hodnota poľa Power
OutputIOActualSpeed(p)	OUT_A..OUT_C	Hodnota poľa ActualSpeed
OutputIOTachoCount(p)	OUT_A..OUT_C	Hodnota poľa TachoCount
OutputIOTachoLimit(p)	OUT_A..OUT_C	Hodnota poľa TachoLimit
OutputIORunState(p)	OUT_A..OUT_C	Hodnota poľa RunState
OutputIOTurnRatio(p)	OUT_A..OUT_C	Hodnota poľa TurnRatio
OutputIORegMode(p)	OUT_A..OUT_C	Hodnota poľa RegMode
OutputIOOverload(p)	OUT_A..OUT_C	Hodnota poľa Overload
OutputIORegPValue(p)	OUT_A..OUT_C	Hodnota poľa RegPValue
OutputIORegIValue(p)	OUT_A..OUT_C	Hodnota poľa RegIValue
OutputIORegDValue(p)	OUT_A..OUT_C	Hodnota poľa RegDValue
OutputIOBlockTachoCount(p)	OUT_A..OUT_C	Hodnota poľa BlockTachoCount
OutputIORotationCount(p)	OUT_A..OUT_C	Hodnota poľa RotationCount

Tabuľka 22. Konštanty adries IOMA

3.5 Modul Sound

Modul Sound má na starosti všetky zvukové výstupy NXT. Umožňuje zahrať jednoduché tóny a prehrať zvukové súbory dvoch rôznych typov.

Konštanta	Hodnota
SoundModuleName	"Sound.mod"
SoundModuleID	0x00080001

Tabuľka 23. Konštanty modulu Sound

Zvukové súbory (.rso) sú podobné ako známe súbory .wav. Obsahujú tisícky zvukových vzoriek, ktoré digitálne popisujú zvukovú vlnu. V zvukových súboroch môže byť zaznamenaná reč, hudba alebo akýkoľvek predstaviteľný zvuk.

Súbory s melódiou (.rmd) sa podobajú známym súborom MIDI. Obsahujú viacero tónov, pričom každý z nich je zadaný frekvenciou a dobou trvania. NXT ich prehráva pomocou jednoduchého generátora čistých sinusových frekvencií. Hoci melódie nie sú také skvelé ako zvukové súbory, zaberajú oveľa menej pamäte.

Ak NXT prikážeme, aby zahralo zvuk, program nečaká dovtedy, kým sa predchádzajúci zvuk dohrá. Na zahratie viacerých tónov alebo zvukových súborov za sebou program musí počkať, kým predchádzajúci zvuk skončí, napr. pomocou API funkcie `Wait` alebo pomocou kontrolovania stavu zvuku v cykle while.

NXC API pozná frekvencie a dĺžky známych tónov, takže pri volaní `PlayTone` alebo `PlayToneEx` môžeme použiť tieto konštanty. Najnižší tón je `TONE_A3` ('A' v 3. oktáve) a najvyšší tón je `TONE_B7` ('H' v 7. oktáve – v angličtine sa tón H označuje B). Dĺžky tónov sú v rozmedzí `MS_1` (1 milisekunda) po `MIN_1` (60000 milisekúnd) a niekoľko konštánt medzitým – pozri súbor NBCCommon.h kde je kompletný zoznam.

3.5.1 Vysokoúrovňové volania

PlayTone(frekvencia, trvanie) Procedúra

Začne hrať tón zadanej frekvencie (v Hz) a dĺžky (v milisekundách). Oba argumenty môžu obsahovať ľubovoľné platné výrazy.

```
PlayTone(440, 500); // Zahrá polsekundové 'A'
```

PlayToneEx(frekvencia, trvanie, hlasitosť, bDokola) Procedúra

Začne hrať tón zadanej frekvencie (v Hz), dĺžky (v milisekundách) a hlasitosti (od 0 do 4). Všetky argumenty môžu obsahovať ľubovoľné platné výrazy.

```
PlayToneEx(440, 500, 2, false);
```

PlayFile(menoSúboru) Procedúra

Začne hrať zadaný zvukový súbor (.rso) alebo melódiu (.rmd). Meno súboru môže byť ľubovoľný reťazcový výraz.

```
PlayFile("startup.rso");
```

PlayFileEx(menoSúboru, hlasitosť, bDokola) Procedúra

Začne hrať zadaný zvukový súbor (.rso) alebo melódiu (.rmd). Meno súboru môže byť ľubovoľný reťazcový výraz. Hlasitosť musí byť číslo v rozmedzí 0 (nehrá) po 4 (najhlasnejšie). BDokola je logická hodnota, ktorá určuje, či sa zvuk má prehrávať dokola.

```
PlayFileEx("startup.rso", 3, true);
```

3.5.2 Nízkoúrovňové volania

Prípustné príznaky pre zvuk zobrazuje nasledujúca tabuľka.

Konštanta zvukového príznaku	Čítanie/Zápis	Význam
SOUND_FLAGS_IDLE	Čítanie	Zvuk nehrá
SOUND_FLAGS_UPDATE	Zápis	Potvrdenie požadovaných zmien
SOUND_FLAGS_RUNNING	Čítanie	Hrá tón alebo súbor

Tabuľka 24. Zvukové príznaky

Prípustné stavy zvukového modulu sú v nasledujúcej tabuľke.

Stav	Čítanie/Zápis	Význam
SOUND_STATE_IDLE	Čítanie	Nehrá, pripravený začať hrať zvuk
SOUND_STATE_FILE	Čítanie	Prehráva súbor so zvukom alebo melódiou
SOUND_STATE_TONE	Čítanie	Prehráva tón
SOUND_STATE_STOP	Zápis	Okamžite vypni zvuk

Tabuľka 25. Konštanty pre stav zvukového modulu

Prípustné zvukové módy sú v nasledujúcej tabuľke.

Mód	Čítanie/Zápis	Význam
SOUND_MODE_ONCE	Čítanie	Zahrá súbor iba raz
SOUND_MODE_LOOP	Čítanie	Prehráva súbor až kým sa nezapíše SOUND_STATE_STOP do stavu (State).
SOUND_MODE_TONE	Čítanie	Zahrá tón so zadanou frekvenciou a dĺžkou.

Tabuľka 26. Konštanty pre mód

Ostatné zvukové konštanty sú v nasledujúcej tabuľke.

Konštanta	Hodnota	Význam
FREQUENCY_MIN	220	Minimálna frekvencia v Hz.
FREQUENCY_MAX	14080	Maximálna frekvencia v Hz.
SAMPLERATE_MIN	2000	Minimálna vzorkovacia frekvencia podporovaná NXT
SAMPLERATE_DEFAULT	8000	Štandardná vzorkovacia frekvencia
SAMPLERATE_MAX	16000	Maximálna vzorkovacia frekvencia podporovaná NXT

Tabuľka 27. Ostatné konštanty modulu Sound

SoundFlags() Operácia

Vráti aktuálne zvukové príznaky. Možné hodnoty sú uvedené v tabuľke 24.

```
x = SoundFlags();
```

SetSoundFlags(n) Procedúra

Nastaví zvukové príznaky. Prípustné hodnoty príznakov sú v tabuľke 24.

```
SetSoundFlags(SOUND_FLAGS_UPDATE);
```

SoundState() Operácia

Vráti aktuálny stav zvukového modulu. Možné stavy sú zobrazené v tabuľke 25.

```
x = SoundState();
```

SetSoundModuleState(n) Procedúra

Nastaví stav zvukového modulu. Prípustné hodnoty stavu sú v tabuľke 25.

```
SetSoundModuleState(SOUND_STATE_STOP);
```

SoundMode() — Operácia

Vráti aktuálny mód zvuku. Možné módy sú uvedené v tabuľke 26.

```
x = SoundMode();
```

SetSoundMode(n) — Procedúra

Nastaví mód zvuku. Prípustné hodnoty pre mód sú v tabuľke 26.

```
SetSoundMode(SOUND_MODE_ONCE);
```

SoundFrequency() — Operácia

Vráti aktuálnu frekvenciu zvuku.

```
x = SoundFrequency();
```

SetSoundFrequency(n) — Procedúra

Nastaví aktuálnu frekvenciu zvuku.

```
SetSoundFrequency(440);
```

SoundDuration() — Operácia

Vráti aktuálnu hodnotu trvania tónu.

```
x = SoundDuration();
```

SetSoundDuration(n) — Procedúra

Nastaví hodnotu trvania tónu.

```
SetSoundDuration(500);
```

SoundSampleRate() — Operácia

Vráti aktuálnu vzorkovaciu frekvenciu.

```
x = SoundSampleRate();
```

SetSoundSampleRate(n) — Procedúra

Nastaví vzorkovaciu frekvenciu.

```
SetSoundSampleRate(4000);
```

SoundVolume() — Operácia

Vráti aktuálnu nastavenú hlasitosť zvuku.

```
x = SoundVolume();
```

SetSoundVolume(n) — Procedúra

Nastaví hlasitosť zvuku.

```
SetSoundVolume(3);
```

StopSound() — Procedúra

Vypne prehrávanie tónu alebo zvuku.

```
StopSound();
```

3.5.3 Adresy položiek štruktúry IOMap (IOMap Offsets)

Offset modulu Sound	Hodnota	Veľkosť
SoundOffsetFreq	0	2
SoundOffsetDuration	2	2
SoundOffsetSampleRate	4	2
SoundOffsetSoundFilename	6	20
SoundOffsetFlags	26	1
SoundOffsetState	27	1
SoundOffsetMode	28	1
SoundOffsetVolume	29	1

Tabuľka 28. Offsety IOMAP pre modul Sound

3.6 Modul IOCtrl

Modul ioctrl zodpovedá za nízkoúrovňovú komunikáciu dvoch procesorov, ktoré sa nachádzajú v NXT. NXC API sprístupňuje dve funkcie z tohto modulu.

Konštanta	Hodnota
IOCtrlModuleName	"IOCtrl.mod"
IOCtrlModuleID	0x00060001

Tabuľka 29. Konštanty modulu IOCtrl

PowerDown() **Procedúra**

Okamžite vypnúť NXT.

```
PowerDown();
```

RebootInFirmwareMode() **Procedúra**

Reštartovať NXT v režime SAMBA, ktorý umožňuje download novej verzie firmware. Táto funkcia sa v normálnom programe NXC pravdepodobne nevyužije.

3.6.1 Adresy položiek štruktúry IOMap (IOMap Offsets)

Offset modulu IOCtrl	Hodnota	Veľkosť
IOCtrlOffsetPowerOn	0	2

Tabuľka 30. IOMap offsety modulu IOCtrl

3.7 Modul Display

Modul display sa stará o možnosti kreslenia na LCD. NXT dokáže na LCD vykresliť body, úsečky, obdĺžniky a kružnice. Dokáže vykresliť obrázky zo súborov, texty a čísla.

Konštanta	Hodnota
DisplayModuleName	"Display.mod"
DisplayModuleID	0x000A0001

Tabuľka 31. Konštanty modulu Display

Obrazovka LCD má počiatok (0, 0) v ľavom dolnom rohu obrazovky, kladné Y-ové súradnice rastú smerom nahor a kladné X-ové súradnice rastú smerom doprava. Pre vypisovanie textov a čísel možno použiť preddefinované konštanty pre Y-ovú súradnicu, smerom zhora: `LCD_LINE1`,

`LCD_LINE2`, `LCD_LINE3`, `LCD_LINE4`, `LCD_LINE5`, `LCD_LINE6`, `LCD_LINE7`,

`LCD_LINE8`. Možno ich použiť vo volaniach NumOut a TextOut. Iné hodnoty budú automaticky upravené tak, že text a čísla sa budú nachádzať na jednej z týchto riadkových pozíciách.

3.7.1 Vysokoúrovňové volania

NumOut(x, y, hodnota, zmazať = false) — Procedúra

Vykreslí číselnú hodnotu na obrazovku na dané súradnice. V poslednom argumente je možné vyžiadať zmazanie obrazovky pred vykreslením. Ak sa tento argument neuvedie, obrazovka sa nezmaže.

```
NumOut(0, LCD_LINE1, x);
```

TextOut(x, y, msg, clear = false) — Procedúra

Vykreslí textovú hodnotu na obrazovku na dané súradnice. V poslednom argumente je možné vyžiadať zmazanie obrazovky pred vykreslením. Ak sa tento argument neuvedie, obrazovka sa nezmaže.

```
TextOut(0, LCD_LINE3, "Hello World!");
```

GraphicOut(x, y, menoSúboru, zmazať = false) — Procedúra

Vykreslí obrázok (icon) zo zadaného súboru na obrazovku na dané súradnice. V poslednom argumente je možné vyžiadať zmazanie obrazovky pred vykreslením. Ak sa tento argument neuvedie, obrazovka sa nezmaže. Ak sa súbor nenájde, nič sa nenakreslí a program nevyhlási žiadnu chybu.

```
GraphicOut(40, 40, "image.ric");
```

GraphicOutEx(x, y, menoSúboru, vars, zmazať = false) — Procedúra

Vykreslí obrázok (icon) zo zadaného súboru na obrazovku na dané súradnice. Použitím hodnôt v poli vars sa obrázok upraví na základe príkazov v danom obrázkovom súbore. V poslednom argumente je možné vyžiadať zmazanie obrazovky pred vykreslením. Ak sa tento argument neuvedie, obrazovka sa nezmaže. Ak sa súbor nenájde, nič sa nenakreslí a program nevyhlási žiadnu chybu.

```
GraphicOutEx(40, 40, "image.ric", variables);
```

CircleOut(x, y, polomer, zmazať = false) — Procedúra

Vykreslí kružnicu na obrazovku so stredom na daných súradniciach a so zadaným polomerom. V poslednom argumente je možné vyžiadať zmazanie obrazovky pred vykreslením. Ak sa tento argument neuvedie, obrazovka sa nezmaže.

```
CircleOut(40, 40, 10);
```

LineOut(x1, y1, x2, y2, zmazať = false) — Procedúra

Vykreslí úsečku na obrazovku z bodu x1, y1 do bodu x2, y2. V poslednom argumente je možné vyžiadať zmazanie obrazovky pred vykreslením. Ak sa tento argument neuvedie, obrazovka sa

nezmaže.

```
LineOut(40, 40, 10, 10);
```

PointOut(x, y, zmazať = false) Procedúra

Vykreslí bod na obrazovku na dané súradnice. V poslednom argumente je možné vyžiadať zmazanie obrazovky pred vykreslením. Ak sa tento argument neuvedie, obrazovka sa nezmaže.

```
PointOut(40, 40);
```

RectOut(x, y, šírka, výška, zmazať = false) Procedúra

Vykreslí obdĺžnik na obrazovku na pozíciu x, y so zadanou výškou a šírkou. V poslednom argumente je možné vyžiadať zmazanie obrazovky pred vykreslením. Ak sa tento argument neuvedie, obrazovka sa nezmaže.

```
RectOut(40, 40, 30, 10);
```

ResetScreen() Procedúra

Obnoví štandardnú obrazovku NXT, ktorá ukazuje bežiaci program.

```
ResetScreen();
```

ClearScreen() Procedúra

Vyčistí obrazovku dočista dočista.

```
ClearScreen();
```

3.7.2 Nízkoúrovňové volania

Prípustné príznaky pre modul display sú v nasledujúcej tabuľke.

Príznak	Čítanie/Zápis	Význam
DISPLAY_ON	Zápis	Display je zapnutý
DISPLAY_REFRESH	Zápis	Zapnúť obnovovanie (refresh)
DISPLAY_POPUP	Zápis	Použiť „popup display memory“
DISPLAY_REFRESH_DISABLED	Čítanie	Refresh je vypnutý
DISPLAY_BUSY	Čítanie	Refresh sa vykonáva

Tabuľka 32. Príznaky modulu Display

DisplayFlags() Operácia

Vráti aktuálne príznaky. Prípustné príznaky sú v tabuľke 32.

```
x = DisplayFlags();
```

SetDisplayFlags(n) Procedúra

Nastaví príznaky. Prípustné príznaky sú v tabuľke 32.

```
SetDisplayFlags(x);
```

DisplayEraseMask() Operácia

Vráti súčasnú masku vymazávania.

```
x = DisplayEraseMask();
```

SetDisplayEraseMask(n) Procedúra

Nastaví masku vymazávania.

```
SetDisplayEraseMask(x);
```

DisplayUpdateMask() Operácia

Vráti súčasnú masku aktualizácie displeja (display update mask).

```
x = DisplayUpdateMask();
```

SetDisplayUpdateMask(n) Procedúra

Nastaví masku aktualizácie displeja.

```
SetDisplayUpdateMask(x);
```

DisplayDisplay() Operácia

Vráti aktuálnu adresu pamäte displeja (display memory address).

```
x = DisplayDisplay();
```

SetDisplayDisplay(n) Procedúra

Nastaví aktuálnu adresu pamäte displeja.

```
SetDisplayDisplay(x);
```

DisplayTextLinesCenterFlags() Operácia

Vráti aktuálne príznaky centrovania textových riadkov.

```
x = DisplayTextLinesCenterFlags();
```

SetDisplayTextLinesCenterFlags(n) Procedúra

Nastaví príznaky centrovania textových riadkov.

```
SetDisplayTextLinesCenterFlags(x);
```

GetDisplayNormal(x, riadok, počet, data) Procedúra

Prečíta zadaný "počet" bajtov z normálnej pamäte displeja do poľa data. Začína čítať na zadaných súradniciach x, riadok. Každý bajt v tejto pamäti je vertikálny pásik 8 bodov na príslušnej pozícii, každý reprezentovaný jedným bitom. Argument riadok vyžaduje hodnoty `TEXTLINE_1` až `TEXTLINE_8`.

```
GetDisplayNormal(0, TEXTLINE_1, 8, ScreenMem);
```

SetDisplayNormal(x, riadok, počet, data) Procedúra

Zapíše "počet" bajtov do normálnej pamäte displeja z poľa data. Začne zapisovať na zadaných súradniciach x, line. Každý bajt v tejto pamäti je vertikálny pásik 8 bodov na príslušnej pozícii, každý reprezentovaný jedným bitom. Argument riadok vyžaduje hodnoty `TEXTLINE_1` až `TEXTLINE_8`.

```
SetDisplayNormal(0, TEXTLINE_1, 8, ScreenMem);
```

GetDisplayPopup(x, riadok, počet, data) Procedúra

Prečíta "počet" bajtov z „popup“ pamäte displeja do poľa data. Začne čítať na súradniciach x,

riadok. Každý bajt v tejto pamäti je vertikálny pásik 8 bodov na príslušnej pozícii, každý reprezentovaný jedným bitom. Argument riadok vyžaduje hodnoty `TEXTLINE_1` až `TEXTLINE_8`.

```
GetDisplayPopup(0, TEXTLINE_1, 8, PopupMem);
```

SetDisplayPopup(x, riadok, počet, data) Procedúra

Zapíše "počet" bajtov do popup pamäte displeja z poľa data. Začne zapisovaťna súradniciach x, riadok. Každý bajt v tejto pamäti je vertikálny pásik 8 bodov na príslušnej pozícii, každý reprezentovaný jedným bitom. Argument riadok vyžaduje hodnoty `TEXTLINE_1` až `TEXTLINE_8`. `SetDisplayPopup(0, TEXTLINE_1, 8, PopupMem);`

3.7.3 Adresy položiek štruktúry IOMap (IOMap Offsets)

Offset modulu Display	Hodnota	Veľkosť
DisplayOffsetPFunc	0	4
DisplayOffsetEraseMask	4	4
DisplayOffsetUpdateMask	8	4
DisplayOffsetPFont	12	4
DisplayOffsetPTextLines(p)	(((p)*4)+16)	4*8
DisplayOffsetPStatusText	48	4
DisplayOffsetPStatusIcons	52	4
DisplayOffsetPScreens(p)	(((p)*4)+56)	4*3
DisplayOffsetPBitmaps(p)	(((p)*4)+68)	4*4
DisplayOffsetPMenuText	84	4
DisplayOffsetPMenuIcons(p)	(((p)*4)+88)	4*3
DisplayOffsetPStepIcons	100	4
DisplayOffsetDisplay	104	4
DisplayOffsetStatusIcons(p)	((p)+108)	1*4
DisplayOffsetStepIcons(p)	((p)+112)	1*5
DisplayOffsetFlags	117	1
DisplayOffsetTextLinesCenterFlags	118	1
DisplayOffsetNormal(l,w)	(((l)*100)+(w)+119)	800
DisplayOffsetPopup(l,w)	(((l)*100)+(w)+919)	800

Tabuľka 33. IOMap offsety modulu Display

3.8 Modul Loader

Modul Loader sa stará o služby súborového systému NXT. Dovoľuje vytvárať nové súbory, otvárať existujúce súbory, čítať, zapisovať, premenovávať a mazať súbory.

Mená súborov v súborovom systéme NXT musia dodržiavať konvenciu 15.3 znakov (spolu 19 znakov). Hoci naraz môže byť otvorených viacero súborov, najviac štyri súbory môžu byť otvorené na zápis súčasne. Počas prístupu k súborom môžu nastať rôzne chyby. NXC API definuje konštanty pre jednotlivé chybové situácie. Sú zobrazené v tabuľke 35.

Konštanta	Hodnota
LoaderModuleName	"Loader.mod"
LoaderModuleID	0x00090001

Tabuľka 34. Konštanty modulu Loader

Chybový kód (Result)	Hodnota
LDR_SUCCESS	0x0000
LDR_INPROGRESS	0x0001
LDR_REQPIN	0x0002
LDR_NOMOREHANDLES	0x8100
LDR_NOSPACE	0x8200
LDR_NOMOREFILES	0x8300
LDR_EOFEXPECTED	0x8400
LDR_ENDOFFILE	0x8500
LDR_NOTLINEARFILE	0x8600
LDR_FILENOTFOUND	0x8700
LDR_HANDLEALREADYCLOSED	0x8800
LDR_NOLINEARSPACE	0x8900
LDR_UNDEFINEDERROR	0x8A00
LDR_FILEISBUSY	0x8B00
LDR_NOWRITEBUFFERS	0x8C00
LDR_APPENDNOTPOSSIBLE	0x8D00
LDR_FILEISFULL	0x8E00
LDR_FILEEXISTS	0x8F00
LDR_MODULENOTFOUND	0x9000
LDR_OUTOFBOUNDARY	0x9100
LDR_ILLEGALFILENAME	0x9200
LDR_ILLEGALHANDLE	0x9300
LDR_BTBUSY	0x9400
LDR_BTCONNECTFAIL	0x9500
LDR_BTTIMEOUT	0x9600
LDR_FILETX_TIMEOUT	0x9700
LDR_FILETX_DSTEXISTS	0x9800
LDR_FILETX_SRCMISSING	0x9900
LDR_FILETX_STREAMERROR	0x9A00
LDR_FILETX_CLOSEERROR	0x9B00

Tabuľka 35. Chybové kódy (Result) pre modul Loader

FreeMemory() Operácia

Vráti počet bajtov pamäte flash, ktoré sú neobsadené.

```
x = FreeMemory();
```

CreateFile(menoSúboru, veľkosť, out handle) Operácia

Vytvorí nový súbor zadanej veľkosti so zadaným menom a otvorí ho na zápis. Posledný argument musí byť premenná, do ktorej toto volanie zapíše handle – hodnotu pomocou ktorej program bude naďalej pracovať so súborom. Volanie vráti chybový kód (Result). Meno súboru a veľkosť môžu byť konštanty, konštantné výrazy alebo premenné. Do súboru s veľkosťou nula bajtov sa nedá zapisovať, keďže funkcie NXC na zápis do súboru nedokážu zväčšiť súbor, keď je kapacita súboru (veľkosť zadaná pri jeho vytváraní) prekročená.

```
result = CreateFile("data.txt", 1024, handle);
```

OpenFileAppend(menoSúboru, out veľkosť, out handle) Operácia

Otvorí existujúci súbor so zadaným menom na zápis. Volanie vráti veľkosť súboru v druhom argumente, ktorý musí byť premenná. V poslednom argumente (tiež musí byť premenná) vráti handle – hodnotu pomocou ktorej bude program naďalej pracovať so súborom. Volanie vráti chybový kód (Result). Argument menoSúboru musí byť konštanta alebo premenná.

```
result = OpenFileAppend("data.txt", fsize, handle);
```

OpenFileRead(menoSúboru, out veľkosť, out handle) Operácia

Otvorí existujúci súbor so zadaným menom na čítanie. Volanie vráti veľkosť súboru v druhom argumente, ktorý musí byť premenná. V poslednom argumente (tiež musí byť premenná) vráti handle – hodnotu pomocou ktorej bude program naďalej pracovať so súborom. Volanie vráti chybový kód (Result). Argument menoSúboru musí byť konštanta alebo premenná.

```
result = OpenFileRead("data.txt", fsize, handle);
```

CloseFile(handle) Operácia

Zatvorí súbor, ktorý je asociovaný so zadanou handle. Volanie vráti chybový kód (Result). Argument handle musí byť konštanta alebo premenná.

```
result = CloseFile(handle);
```

ResolveHandle(menoSúboru, out handle, out bZapisovateľný) Operácia

Nájde file handle k súboru zadanému menom. Volanie vráti príslušný handle v druhom argumente, ktorý musí byť premenná. V tretom argumente (tiež musí byť premenná) vráti logickú hodnotu, ktorá udáva, či sa do tohto otvoreného súboru smie zapisovať. Volanie vráti chybový kód (Result).

```
result = ResolveHandle("data.txt", handle, bCanWrite);
```

RenameFile(staréMenoSúboru, novéMenoSúboru) Operácia

Premenuje súbor zo starého mena na nové. Volanie vráti chybový kód. Argumenty musia byť konštanty alebo premenné.

```
result = RenameFile("data.txt", "mydata.txt");
```

DeleteFile(menoSúboru) Operácia

Vymaže zadaný súbor. Argument musí byť konštanta alebo premenná. Volanie vráti chybový kód.

```
result = DeleteFile("data.txt");
```

Read(handle, out hodnota) Operácia

Prečíta číselnú hodnotu z otvoreného súboru asociovaného s príslušným handle (musí byť premenná). Volanie vráti chybový kód. Argument hodnota musí byť premenná a jeho typ určuje počet bajtov, ktoré sa prečítajú.

```
result = Read(handle, hodnota);
```

ReadLn(handle, out hodnota) Operácia

Prečíta číselnú hodnotu z otvoreného súboru asociovaného s príslušným handle (musí byť premenná). Volanie vráti chybový kód. Argument hodnota musí byť premenná a jeho typ určuje počet bajtov, ktoré sa prečítajú. Volanie na konci prečíta dva bajty na vyše, o ktorých predpokladá, že sú to znaky CR a LF.

```
result = ReadLn(handle, value);
```

ReadBytes(handle, in/out dĺžka, out buf) Operácia

Prečíta zadaný počet bajtov zo súboru asociovaného s príslušným handle (musí byť premenná). Volanie vráti chybový kód. Argument dĺžka musí byť premenná. Argument buf musí byť pole reťazcových premenných. Skutočný počet bajtov, ktoré volanie prečíta sa vráti v argumente dĺžka.

```
result = ReadBytes(handle, dlzka, buffer);
```

Write(handle, hodnota) Operácia

Zapíše číselnú hodnotu do súboru asociovaného s príslušným handle (musí byť premenná). Volanie vráti chybový kód. Argument hodnota musí byť konštanta, konštantný výraz alebo premenná, jeho typ určuje počet bajtov, ktoré sa zapíšu.

```
result = Write(handle, hodnota);
```

WriteLn(handle, hodnota) Operácia

Zapíše číselnú hodnotu do súboru asociovaného s príslušným handle (musí byť premenná). Volanie vráti chybový kód. Argument hodnota musí byť konštanta, konštantný výraz alebo premenná, jeho typ určuje počet bajtov, ktoré sa zapíšu. Volanie na koniec zapíše do súboru znaky CR a LF.

```
result = WriteLn(handle, hodnota);
```

WriteString(handle, reťazec, out počet) Operácia

Zapíše reťazec do súboru asociovaného s príslušným handle (musí byť premenná). Volanie vráti chybový kód. Argument počet musí byť premenná. Argument reťazec musí byť reťazcová premenná alebo konštanta. Skutočný počet bajtov, ktoré boli zapísané volanie vráti v argumente počet.

```
result = WriteString(handle, "testing", count);
```

WriteLnString(handle, reťazec, out počet) Operácia

Zapíše reťazec do súboru asociovaného s príslušným handle (musí byť premenná). Volanie vráti chybový kód. Argument počet musí byť premenná. Argument reťazec musí byť reťazcová premenná alebo konštanta. Skutočný počet bajtov, ktoré boli zapísané volanie vráti v argumente počet. Volanie na koniec zapíše do súboru znaky CR a LF.

```
result = WriteLnString(handle, "testing", count);
```

WriteBytes(handle, data, out počet) Operácia

Zapíše pole bajtov do súboru asociovaného s príslušným handle (musí byť premenná). Volanie vráti chybový kód. Argument počet musí byť premenná. Argument data musí byť pole. Počet bajtov, ktoré boli zapísané volanie vráti v argumente počet.

```
result = WriteBytes(handle, buffer, count);
```

WriteBytesEx(handle, in/out dĺžka, buf) Operácia

Zapíše určený počet bajtov do súboru asociovaného s príslušným handle (musí byť premenná). Volanie vráti chybový kód. Argument dĺžka musí byť premenná. Argument buf musí byť pole alebo reťazcová premenná alebo reťazcová konštanta. Počet bajtov, ktoré boli zapísané volanie vráti v argumente dĺžka.

```
result = WriteBytesEx(handle, dlzka, buffer);
```

3.8.1 Adresy položiek štruktúry IOMap (IOMap Offsets)

Offset modulu Loader	Hodnota	Veľkosť
LoaderOffsetPFunc	0	4
LoaderOffsetFreeUserFlash	4	4

Tabuľka 36. IOMap offsety modulu Loader

3.9 Modul Command

Modul command má v NXT nastarosti vykonávanie užívateľským programov pomocou „NXT virtual machine“. Implementuje protokol priamych príkazov (direct command protocol), ktorý umožňuje, aby NXT vykonávalo príkazy prijaté cez USB alebo BlueTooth od iných zariadení, napr. PC alebo druhá NXT kocka.

Konštanta	Hodnota
CommandModuleName	"Command.mod"
CommandModuleID	0x00010001

Tabuľka 37. Konštanty modulu Command

3.9.1 Adresy položiek štruktúry IOMap (IOMap Offsets)

Offset modulu Command	Hodnota	Veľkosť
CommandOffsetFormatString	0	16
CommandOffsetPRCHandler	16	4
CommandOffsetTick	20	4
CommandOffsetOffsetDS	24	2
CommandOffsetOffsetDVA	26	2
CommandOffsetProgStatus	28	1
CommandOffsetAwake	29	1
CommandOffsetActivateFlag	30	1
CommandOffsetDeactivateFlag	31	1
CommandOffsetFileName	32	20
CommandOffsetMemoryPool	52	32k

Tabuľka 38. IOMap offsety modulu Command

3.10 Modul Button

Modul button obsluhuje a umožňuje snímať stav štyroch tlačidiel na kocke NXT.

Konštanta	Hodnota
ButtonModuleName	"Button.mod"
ButtonModuleID	0x00040001

Tabuľka 39. Konštanty modulu Button

3.10.1 Vysokoúrovňové funkcie

Prípustné konštanty sú zobrazené v nasledujúcej tabuľke.

Konštanta	Hodnota
BTN1, BTNEXIT	0
BTN2, BTNRIGHT	1
BTN3, BTNLEFT	2
BTN4, BTNCENTER	3
NO_OF_BTNS	4

Tabuľka 40. Konštanty tlačidiel

ButtonCount(btn, vynulovať) **Operácia**

Vráti počet stlačení zadaného tlačidla od posledného vynulovania počítadla stlačení. Logický argument vynulovať dovoľuje súčasné vynulovanie počítadla stlačení. Prípustné hodnoty pre argument btn sú v tabuľke 40.

```
value = ButtonCount(BTN1, true);
```

ButtonPressed(btn, vynulovať) Operácia

Vráti logickú hodnotu, ktorá je true, ak je zadané tlačidlo práve stlačené. Logický argument vynulovať dovoľuje súčasné vynulovanie počítadla stlačení. Prípustné hodnoty pre argument btn sú v tabuľke 40.

```
value = ButtonPressed(BTN1, true);
```

ReadButtonEx(btn, vynulovať, out stlačené, out počet) Procedúra

Prečíta zadané tlačidlo. Nastaví argumenty stlačené a počet zodpovedajúco od aktuálneho stavu tlačidla. Logický argument vynulovať dovoľuje súčasné vynulovanie počítadla stlačení. Prípustné hodnoty pre argument btn sú v tabuľke 40.

```
ReadButtonEx(BTN1, true, stlacene, pocet);
```

3.10.2 Nízkoúrovňové funkcie

Prípustné stavy tlačidla uvádza nasledujúca tabuľka.

Stav	Hodnota
BTNSTATE_PRESSED_EV	0x01
BTNSTATE_SHORT_RELEASED_EV	0x02
BTNSTATE_LONG_PRESSED_EV	0x04
BTNSTATE_LONG_RELEASED_EV	0x08
BTNSTATE_PRESSED_STATE	0x80

Tabuľka 41. Konštanty pre stav tlačidla

ButtonPressCount(btn) Operácia

Vráti počet stlačení zadaného tlačidla. Prípustné hodnoty pre argument btn sú v tabuľke 40.

```
value = ButtonPressCount(BTN1);
```

SetButtonPressCount(btn, hodnota) Procedúra

Nastaví počet stlačení pre zadané tlačidlo. Prípustné hodnoty pre argument btn sú v tabuľke 40.

```
SetButtonPressCount(BTN1, value);
```

ButtonLongPressCount(btn) Operácia

Vráti „long“ počet stlačení zadaného tlačidla. Prípustné hodnoty pre argument btn sú v tab. 40.

```
value = ButtonLongPressCount(BTN1);
```

SetButtonLongPressCount(btn, hodnota) Procedúra

Nastaví „long“ počet stlačení pre zadané tlačidlo. Prípustné hodnoty pre argument btn sú v tabuľke 40.

```
SetButtonLongPressCount(BTN1, value);
```

ButtonShortReleaseCount(btn) Operácia

Vráti „short“ počet pustení pre zadané tlačidlo. Prípustné hodnoty pre argument btn sú v tab. 40.

```
value = ButtonShortReleaseCount(BTN1);
```

SetButtonShortReleaseCount(btn, hodnota) Procedúra

Nastaví „short“ počet pustení pre zadané tlačidlo. Prípustné hodnoty pre argument btn sú v tabuľke 40.

```
SetButtonShortReleaseCount(BTN1, hodnota);
```

ButtonLongReleaseCount(btn) Operácia

Vráti „long“ počet pustení pre zadané tlačidlo. Prípustné hodnoty pre argument btn sú v tab. 40.

```
value = ButtonLongReleaseCount(BTN1);
```

SetButtonLongReleaseCount(btn, value) Procedúra

Nastaví „long“ počet pustení pre zadané tlačidlo. Prípustné hodnoty pre argument btn sú v tabuľke 40.

```
SetButtonLongReleaseCount(BTN1, hodnota);
```

ButtonReleaseCount(btn) Operácia

Vráti počet pustení pre zadané tlačidlo. Prípustné hodnoty pre argument btn sú v tabuľke 40.

```
value = ButtonReleaseCount(BTN1);
```

SetButtonReleaseCount(btn, value) Procedúra

Nastaví počet pustení pre zadané tlačidlo. Prípustné hodnoty pre argument btn sú v tabuľke 40.

```
SetButtonReleaseCount(BTN1, value);
```

ButtonState(btn) Operácia

Vráti stav zadaného tlačidla. Prípustné hodnoty pre argument btn sú v tabuľke 40. Možné stavy tlačidla sú v tabuľke 41.

```
value = ButtonState(BTN1);
```

SetButtonState(btn, hodnota) Procedúra

Nastaví stav zadaného tlačidla. Prípustné hodnoty pre argument btn sú v tabuľke 40. Možné stavy tlačidla sú v tabuľke 41.

```
SetButtonState(BTN1, BTNSTATE_PRESSED_EV);
```

3.10.3 Adresy položiek štruktúry IOMap (IOMap Offsets)

Konštanta	Hodnota	Veľkosť
ButtonOffsetPressedCnt(b)	(((b)*8)+0)	1
ButtonOffsetLongPressCnt(b)	(((b)*8)+1)	1
ButtonOffsetShortRelCnt(b)	(((b)*8)+2)	1
ButtonOffsetLongRelCnt(b)	(((b)*8)+3)	1
ButtonOffsetRelCnt(b)	(((b)*8)+4)	1
ButtonOffsetState(b)	((b)+32)	1*4

Tabuľka 42. IOMap offsety modulu Button

3.11 Modul UI

Modul UI zabezpečuje rôzne aspekty používateľského rozhrania kocky NXT.

Konštanta	Hodnota
UIModuleName	"Ui.mod"
UIModuleID	0x000C0001

Tabuľka 43. Konštanty modulu UI

Prípustné príznaky sú v nasledujúcej tabuľke

Konštanta	Hodnota
UI_FLAGS_UPDATE	0x01
UI_FLAGS_DISABLE_LEFT_RIGHT_ENTER	0x02
UI_FLAGS_DISABLE_EXIT	0x04
UI_FLAGS_REDRAW_STATUS	0x08
UI_FLAGS_RESET_SLEEP_TIMER	0x10
UI_FLAGS_EXECUTE_LMS_FILE	0x20
UI_FLAGS_BUSY	0x40
UI_FLAGS_ENABLE_STATUS_UPDATE	0x80

Tabuľka 44. Príznaky pre príkazy modulu UI

Prípustné stavy modulu UI sú v nasledujúcej tabuľke:

Konštanta	Hodnota
UI_STATE_INIT_DISPLAY	0
UI_STATE_INIT_LOW_BATTERY	1
UI_STATE_INIT_INTRO	2
UI_STATE_INIT_WAIT	3
UI_STATE_INIT_MENU	4
UI_STATE_NEXT_MENU	5
UI_STATE_DRAW_MENU	6
UI_STATE_TEST_BUTTONS	7
UI_STATE_LEFT_PRESSED	8
UI_STATE_RIGHT_PRESSED	9
UI_STATE_ENTER_PRESSED	10
UI_STATE_EXIT_PRESSED	11
UI_STATE_CONNECT_REQUEST	12
UI_STATE_EXECUTE_FILE	13
UI_STATE_EXECUTING_FILE	14
UI_STATE_LOW_BATTERY	15
UI_STATE_BT_ERROR	16

Tabuľka 45. Konštanty stavov UI

Prípustné stavy tlačidiel sú v nasledujúcej tabuľke.

Konštanta	Hodnota
UI_BUTTON_NONE	1
UI_BUTTON_LEFT	2
UI_BUTTON_ENTER	3
UI_BUTTON_RIGHT	4
UI_BUTTON_EXIT	5

Tabuľka 46. Konštanty tlačidiel

Prípustné stavy BlueTooth modulu UI sú v nasledujúcej tabuľke.

Konštanta	Hodnota
UI_BT_STATE_VISIBLE	0x01
UI_BT_STATE_CONNECTED	0x02
UI_BT_STATE_OFF	0x04
UI_BT_ERROR_ATTENTION	0x08
UI_BT_CONNECT_REQUEST	0x40
UI_BT_PIN_REQUEST	0x80

Tabuľka 47. Konštanty pre stav BlueTooth

Volume() — Operácia

Vráti hlasitosť zvukov používateľského rozhrania, 0 až 4.

```
x = Volume();
```

SetVolume(hodnota) — Procedúra

Nastaví úroveň hlasitosti zvukov používateľského rozhrania, 0 až 4.

```
SetVolume(3);
```

BatteryLevel() — Operácia

Vráti napätie na baterke v milivoltoch.

```
x = BatteryLevel();
```

BluetoothState() — Operácia

Vráti stav BlueTooth. Prípustné stavy Bluetooth sú v tabuľke 47.

```
x = BluetoothState();
```

SetBluetoothState(hodnota) — Procedúra

Nastaví stav Bluetooth. Prípustné stavy Bluetooth sú v tabuľke 47.

```
SetBluetoothState(UI_BT_STATE_OFF);
```

CommandFlags() — Operácia

Vráti príznaky príkazov. Prípustné hodnoty sú v tabuľke 44.

```
x = CommandFlags();
```

SetCommandFlags(hodnota) Procedúra

Nastaví príznaky príkazov. Prípustné hodnoty sú v tabuľke 44.

```
SetCommandFlags(UI_FLAGS_REDRAW_STATUS);
```

UIState() Operácia

Zistí stav používateľského rozhrania. Možné stavy sú uvedené v tabuľke 45.

```
x = UIState();
```

SetUIState(hodnota) Procedúra

Nastaví stav používateľského rozhrania. Prípustné hodnoty sú v tabuľke 45.

```
SetUIState(UI_STATE_LOW_BATTERY);
```

UIButton() Operácia

Vráti informáciu o tlačidlách používateľského rozhrania. Prípustné hodnoty sú v tabuľke 46.

```
x = UIButton();
```

SetUIButton(hodnota) Procedúra

Nastaví informáciu o tlačidlách používateľského rozhrania. Prípustné hodnoty sú v tabuľke 46.

```
SetUIButton(UI_BUTTON_ENTER);
```

VMRunState() Operácia

Vráti informáciu o stave „virtual machine“.

```
x = VMRunState();
```

SetVMRunState(value) Procedúra

Nastaví informáciu o stave „virtual machine“.

```
SetVMRunState(0); // vypnutá
```

BatteryState() Operácia

Vráti stavovú informáciu o baterke (0..4).

```
x = BatteryState();
```

SetBatteryState(hodnota) Procedúra

Nastaví stavovú informáciu o batérii.

```
SetBatteryState(4);
```

RechargeableBattery() Operácia

Zistí, či je v NXT vložená nabíjateľná batéria.

```
x = RechargeableBattery();
```

ForceOff(n) Procedúra

Prinúti kocku NXT, aby sa vypla, ak zadaná hodnota je väčšia ako nula.

```
ForceOff(true);
```

UsbState() Operácia

Vráti informáciu o stave USB (0=odpojené, 1=pripojené, 2=používa sa).

```
x = UsbState();
```

SetUsbState(hodnota) Procedúra

Nastaví informáciu o stave USB (0=odpojené, 1=pripojené, 2=používa sa).

```
SetUsbState(2);
```

OnBrickProgramPointer() Operácia

Vráti aktuálny krok OBP (on-brick program);

```
x = OnBrickProgramPointer();
```

SetOnBrickProgramPointer(hodnota) Procedúra

Nastaví aktuálny krok OBP (on-brick program).

```
SetOnBrickProgramPointer(2);
```

LongAbort() Operácia (+)

Vráti hodnotu atribútu dlhého prerušenia programu rozšíreného firmware (true alebo false). Ak je nastavené na true, program má prístup k tlačidlu Escape (tmavosivé tlačidlo). Na prerušenie behu programu potom treba toto tlačidlo dlhšie podržať.

```
x = LongAbort();
```

SetLongAbort(hodnota) Procedúra (+)

Nastaví hodnotu atribútu dlhého prerušenia programu rozšíreného firmware (true alebo false). Set the enhanced NBC/NXC firmware's long abort setting (true or false). Ak je nastavené na true, program má prístup k tlačidlu Escape (tmavosivé tlačidlo). Na prerušenie behu programu potom treba toto tlačidlo dlhšie podržať.

```
SetLongAbort(true);
```

3.11.1 Adresy položiek štruktúry IOMap (IOMap Offsets)

Konštanta	Hodnota	Veľkosť
UIOffsetPMenu	0	4
UIOffsetBatteryVoltage	4	2
UIOffsetLMSfilename	6	20
UIOffsetFlags	26	1
UIOffsetState	27	1
UIOffsetButton	28	1
UIOffsetRunState	29	1
UIOffsetBatteryState	30	1
UIOffsetBluetoothState	31	1
UIOffsetUsbState	32	1
UIOffsetSleepTimeout	33	1

UIOffsetSleepTimer	34	1
UIOffsetRechargeable	35	1
UIOffsetVolume	36	1
UIOffsetError	37	1
UIOffsetOBPPointer	38	1
UIOffsetForceOff	39	1

Tabuľka 48. IOMap offsety modulu UI

3.12 Modul LowSpeed

Modul lowspeed zabezpečuje komunikáciu so senzormi po digitálnej zbernici I2C.

Konštanta	Hodnota
LowSpeedModuleName	"Low Speed.mod"
LowSpeedModuleID	0x000B0001

Tabuľka 49. Konštanty modulu LowSpeed

Zariadenia, ktoré používajú I2C protokol a pripájajú sa na jeden zo štyroch vstupných portov obsluhuje modul modul LowSpeed. Typ senzora na príslušnom porte musí byť nastavený na `SENSOR_TYPE_LOWSPEED` alebo `SENSOR_TYPE_LOWSPEED_9V`, ak zariadenie vyžaduje 9V napájanie z kocky NXT. Majte tiež na pamäti, že po nastavení nového typu je tiež potrebné nastaviť atribút daného portu `InvalidData` na true a počkať v cykle, kým firmware NXT nenastaví tento atribút naspäť na false. Tento proces slúži na to, aby mal firmware dostatok času na inicializáciu portu prípadne vrátane 9V napätia. Niektoré zariadenia môžu potrebovať ďalší čas na inicializáciu po pripojení na napájanie. API funkcia SetSensorLowspeed nastaví daný port na

`SENSOR_TYPE_LOWSPEED_9V` a potom zavolá ResetSensor, ktorý sám vykoná inicializačný cyklus cez atribút InvalidData popísaný vyššie.

Pri komunikácii so zariadeniami I2C sa používa režim master/slave, pričom NXT vždy vystupuje v roli master. To znamená, že firmware riadi všetky operácie čítania a zápisu na zbernici. NXT firmware má pre každý zo štyroch portov vyhradené vyrovnávacie pamäte na čítanie a zápis, a tri hlavné metódy modulu LowSpeed umožňujú k týmto pamätiam pristupovať.

Volanie LowspeedWrite naštartuje asynchrónny prenos medzi kockou NXT a digitálnym zariadením. Program pokračuje v behu zakiaľ čo sa v pozadí firmware stará o posielanie požadoaných bajtov z vyrovnávacej pamäte a načítanie odpovede zariadenia. Keďže NXT je v roli master, je ku každej operácii zápisu nutné zadať aj počet bajtov, ktoré bude obsahovať odpoveď zariadenia. V každom smere je takto možné počas jednej transakcie preniesť najviac 16 bajtov. Po odštartovaní transakcie pomocou LowspeedWrite môžeme použiť volanie LowspeedStatus, napríklad v cykle, aby sme zistili stav portu. Keď volanie vráti chybový (status) kód 0 a počet bajtov, ktoré čakajú prijaté vo vyrovnávacej pamäti čítania, môžeme ich prečítať pomocou volania LowspeedRead z vyrovnávacej pamäte do zadanej premennej. Ktorékoľvek z týchto volaní môže vrátiť rôzne chybové kódy, ak niečo neprebehne v poriadku. Chybový kód 0 znamená, že port je voľný a posledná transakcia (ak nejaká bola) prebehla bez chýb.

Záporné chybové kódy a kladný kód 32 znamenajú chybu. Každé volanie pripúšťa niekoľko rôznych chýb. Prípustné návratové hodnoty sú zobrazené v nasledujúcej tabuľke.

Konštanta	Hodnota	Význam
NO_ERR	0	Operácia bola úspešná.
STAT_COMM_PENDING	32	Zadaný port je obsadený a prebieha na ňom komunikácia.
ERR_INVALID_SIZE	-19	Zadaná premenná alebo počet presahuje 16-bajtový limit.
ERR_COMM_CHAN_NOT_READY	-32	Zadaný port je obsadený, alebo nesprávne nakonfigurovaný.
ERR_COMM_CHAN_INVALID	-33	Zadaný port je nesprávny, číslo musí byť v rozmedzí od 0 do 3.
ERR_COMM_BUS_ERR	-35	Posledná transakcia neprebehla bez chýb, možno kvôli chybe na strane zariadenia.

Tabuľka 50. Návratové hodnoty pri komunikácii I2C

3.12.1 Vysokoúrovňové funkcie

LowspeedWrite(port, dĺžkaOdpovede, buffer) Procedúra

Naštartuje komunikáciu s I2C zariadením na zadanom porte. Vysielať sa začnú bajty v poli buffer. Zároveň sa zariadeniu vyšle počet očakávaných bajtov, ktoré má vyslať v odpovednom pakete. Maximálny počet bajtov, ktoré môžu byť zapísané alebo prečítané je 16. Port môže byť určený konštantou (napr. `IN_1`, `IN_2`, `IN_3`, alebo `IN_4`) alebo premennou. Konštanty by mali byť použité všade kde sa má, lebo inak sa blokuje prístup k zariadeniam I2C v ostatných úlohách. Návratové hodnoty sú v tabuľke 50.

```
x = LowspeedWrite(IN_1, 1, inbuffer);
```

LowspeedStatus(port, out početPripravenýchBajtov) Procedúra

Zistí stav komunikácie s I2C zariadením na zadanom porte. Ak posledná operácia na tomto porte bol úspešný zápis pomocou volania LowspeedWrite, ktoré vyžadovalo odpoveď od zariadenia, potom argument početPripravenýchBajtov bude nastavený na počet prijatých bajtov vo vyrovnávacej pamäti na čítanie. Port môže byť zadaný pomocou konštanty (napr. `IN_1`, `IN_2`, `IN_3`, alebo `IN_4`) alebo premennej. Konštanty by mali byť použité všade, kde je to možné, lebo inak sa blokuje prístup k zariadeniam I2C v ostatných úlohách. Návratové hodnoty sú v tabuľke 50. Ak je návratová hodnota 0, tak posledná operácia prebehla bez chýb. Kým LowSpeedStatus vracia hodnotu `STAT_COMM_PENDING`, program by nemal používať volania LowspeedRead a LowspeedWrite.

```
x = LowspeedStatus(IN_1, nRead);
```

LowspeedCheckStatus(port) Procedúra

Zistí stav komunikácie I2C so zariadením na zadanom porte. Port môže byť zadaný pomocou konštanty (napr. `IN_1`, `IN_2`, `IN_3`, alebo `IN_4`) alebo premennej. Konštanty by mali byť použité všade, kde je to možné, lebo inak sa blokuje prístup k zariadeniam I2C v ostatných úlohách. Návratové hodnoty sú v tabuľke 50. Ak je návratová hodnota 0, tak posledná operácia prebehla bez chýb. Kým LowSpeedStatus vracia hodnotu `STAT_COMM_PENDING`, program by nemal používať volania LowspeedRead a LowspeedWrite.

```
x = LowspeedCheckStatus(IN_1);
```

LowspeedBytesReady(port) Procedúra

Zistí stav komunikácie I2C so zariadením na zadanom porte. Ak posledná operácia na tomto porte bol úspešný zápis pomocou volania LowspeedWrite, ktoré vyžadovalo odpoveď od

zariadenia, potom volanie vráti počet prijatých bajtov vo vyrovnávacej pamäti na čítanie. Port môže byť zadaný pomocou konštanty (napr. `IN_1`, `IN_2`, `IN_3`, alebo `IN_4`) alebo premennej. Konštanty by mali byť použité všade, kde je to možné, lebo inak sa blokuje prístup k zariadeniam I2C v ostatných úlohách.

```
x = LowspeedBytesReady(IN_1);
```

LowspeedRead(port, početBajtov, out buffer) Procedúra

Prečíta požadovaný počet bajtov zo zariadenia I2C na zadanom porte a uloží ich do poľa buffer. Maximálny počet bajtov, ktoré môžu byť zapísané alebo prečítané je 16. Port môže byť určený konštantou (napr. `IN_1`, `IN_2`, `IN_3`, alebo `IN_4`) alebo premennou. Konštanty by mali byť použité všade kde sa má, lebo inak sa blokuje prístup k zariadeniam I2C v ostatných úlohách. Návratové hodnoty sú v tabuľke 50. Ak je návratová hodnota záporná, výstupný buffer bude prázdny.

```
x = LowspeedRead(IN_1, 1, outbuffer);
```

I2CWrite(port, dĺžkaOdpovede, buffer) Procedúra

Druhé meno (alias) pre LowspeedWrite.

```
x = I2CWrite(IN_1, 1, inbuffer);
```

I2CStatus(port, out početPripravenýchBajtov) Procedúra

Druhé meno (alias) pre LowspeedStatus.

```
x = I2CStatus(IN_1, nRead);
```

I2CCheckStatus(port) Procedúra

Druhé meno (alias) pre LowspeedCheckStatus.

```
x = I2CCheckStatus(IN_1);
```

I2CBytesReady(port) Procedúra

Druhé meno (alias) pre LowspeedBytesReady.

```
x = I2CBytesReady(IN_1);
```

I2CRead(port, početBajtov, out buffer) Procedúra

Druhé meno (alias) pre LowspeedRead.

```
x = I2CRead(IN_1, 1, outbuffer);
```

I2CBytes(port, inbuf, in/out počet, out outbuf) Procedúra

Zapíše bajty z poľa inbuf na zariadenie na zadanom porte a počká, kým zariadenie neodpovie vyslaním požadovaného počtu bajtov, ktoré naplní do poľa outbuf. Port môže byť určený konštantou (napr. `IN_1`, `IN_2`, `IN_3`, alebo `IN_4`) alebo premennou. Vráti true/false poľa toho, či bol transakcia úspešná alebo nie. Toto volanie len obaluje volania troch hlavných funkcií I2C. Okrem toho udržiava obsah posledného dobre prečítaného bufra a vráti v bufri tieto hodnoty, ak operácia nie je úspešná.

```
x = I2CBytes(IN_4, writebuf, cnt, readbuf);
```

3.12.2 Nízkoúrovňové funkcie

Možné stavy modulu lowspeed su zobrazené v nasledujúcej tabuľke.

Konštanta	Hodnota
COM_CHANNEL_NONE_ACTIVE	0x00
COM_CHANNEL_ONE_ACTIVE	0x01
COM_CHANNEL_TWO_ACTIVE	0x02
COM_CHANNEL_THREE_ACTIVE	0x04
COM_CHANNEL_NONE_ACTIVE	0x08

Tabuľka 51. Stavy modulu lowspeed

Prípustné stavy kanálov lowspeed sú zobrazené v nasledujúcej tabuľke.

Konštanta	Hodnota
LOWSPEED_IDLE	0
LOWSPEED_INIT	1
LOWSPEED_LOAD_BUFFER	2
LOWSPEED_COMMUNICATING	3
LOWSPEED_ERROR	4
LOWSPEED_DONE	5

Tabuľka 52. Stavy kanálov lowspeed

Prípustné módy lowspeed sú zobrazené v nasledujúcej tabuľke.

Konštanta	Hodnota
LOWSPEED_TRANSMITTING	1
LOWSPEED_RECEIVING	2
LOWSPEED_DATA_RECEIVED	3

Tabuľka 53. Konštanty pre módy lowspeed

Prípustné chybové kódy lowspeed modulu sú zobrazené v nasledujúcej tabuľke.

Konštanta	Hodnota
LOWSPEED_NO_ERROR	0
LOWSPEED_CH_NOT_READY	1
LOWSPEED_TX_ERROR	2
LOWSPEED_RX_ERROR	3

Tabuľka 54. Konštanty pre chybové kódy lowspeed

GetLSInputBuffer(port, offset, počet, out data) Procedúra

Prečíta údaje z vyrovnávacej pamäte na čítanie na zadanom porte.

```
GetLSInputBuffer(IN_1, 0, 8, buffer);
```

SetLSInputBuffer(port, offset, počet, data) — Procedúra

Zapíše údaje do vyrovnávacej pamäte na čítanie na zadanom porte.

```
SetLSInputBuffer(IN_1, 0, 8, data);
```

GetLSOutputBuffer(port, offset, počet, out data) — Procedúra

Prečíta údaje z vyrovnávacej pamäte na zápis na zadanom porte.

```
GetLSOutputBuffer(IN_1, 0, 8, outbuffer);
```

SetLSOutputBuffer(port, offset, count, data) — Procedúra

Zapíše údaje do vyrovnávacej pamäte na zápis na zadanom porte.

```
SetLSOutputBuffer(IN_1, 0, 8, data);
```

LSInputBufferInPtr(port) — Operácia

Vráti hodnotu ukazovateľa čítania z vyrovnávacej pamäte na čítanie na zadanom porte. Port musí byť konštanta (IN_1..IN_4).

```
x = LSInputBufferInPtr(IN_1);
```

SetLSInputBufferInPtr(port, n) — Procedúra

Nastaví hodnotu ukazovateľa čítania z vyrovnávacej pamäte na čítanie na zadanom porte. Port musí byť konštanta (IN_1..IN_4).

```
SetLSInputBufferInPtr(IN_1, x);
```

LSInputBufferOutPtr(port) — Operácia

Vráti hodnotu ukazovateľa zápisu z vyrovnávacej pamäte na čítanie na zadanom porte. Port musí byť konštanta (IN_1..IN_4).

```
x = LSInputBufferOutPtr(IN_1);
```

SetLSInputBufferOutPtr(port, n) — Procedúra

Nastaví hodnotu ukazovateľa zápisu z vyrovnávacej pamäte na čítanie na zadanom porte. Port musí byť konštanta (IN_1..IN_4).

```
SetLSInputBufferOutPtr(IN_1, x);
```

LSInputBufferBytesToRx(port) — Operácia

Vráti počet bajtov, ktoré sa majú prijať zo zariadenia na zadanom porte. Port musí byť konštanta (IN_1..IN_4).

```
x = LSInputBufferBytesToRx(IN_1);
```

SetLSInputBufferBytesToRx(port, n) — Procedúra

Nastaví počet bajtov, ktoré sa majú prijať zo zariadenia na zadanom porte. Port musí byť konštanta (IN_1..IN_4).

```
SetLSInputBufferBytesToRx(IN_1, x);
```

LSOutputBufferInPtr(port) Operácia

Vráti hodnotu ukazovateľa čítania z vyrovnávacej pamäte na zápis na zadanom porte. Port musí byť konštanta (IN_1..IN_4).

```
x = LSOutputBufferInPtr(IN_1);
```

SetLSOutputBufferInPtr(port, n) Procedúra

Nastaví hodnotu ukazovateľa čítania z vyrovnávacej pamäte na zápis na zadanom porte. Port musí byť konštanta (IN_1..IN_4).

```
SetLSOutputBufferInPtr(IN_1, x);
```

LSOutputBufferOutPtr(port) Operácia

Vráti hodnotu ukazovateľa zápisu z vyrovnávacej pamäte na zápis na zadanom porte. Port musí byť konštanta (IN_1..IN_4).

```
x = LSOutputBufferOutPtr(IN_1);
```

SetLSOutputBufferOutPtr(port, n) Procedúra

Nastaví hodnotu ukazovateľa zápisu z vyrovnávacej pamäte na zápis na zadanom porte. Port musí byť konštanta (IN_1..IN_4).

```
SetLSOutputBufferOutPtr(IN_1, x);
```

LSOutputBufferBytesToRx(port) Operácia

Vráti počet bajtov vo výstupnej vyrovnávacej pamäti na zadanom porte I2C. Port musí byť konštanta (IN_1..IN_4).

```
x = LSOutputBufferBytesToRx(IN_1);
```

SetLSOutputBufferBytesToRx(port, n) Procedúra

Nastaví počet bajtov vo výstupnej vyrovnávacej pamäti na zadanom porte I2C. Port musí byť konštanta (IN_1..IN_4).

```
SetLSOutputBufferBytesToRx(IN_1, x);
```

LSMode(port) Operácia

Vráti mód I2C na príslušnom porte. Port musí byť konštanta (IN_1..IN_4).

```
x = LSMode(IN_1);
```

SetLSMode(port,mód) Procedúra

Nastaví mód I2C na príslušnom porte. Port musí byť konštanta (IN_1..IN_4).

```
SetLSMode(IN_1, LOWSPEED_TRANSMITTING);
```

LSChannelState(port) Operácia

Vráti stav kanálu I2C komunikácie na príslušnom porte. Port musí byť konštanta (IN_1..IN_4).

```
x = LSChannelState(IN_1);
```

SetLSChannelState(port, stavKanálu) — Procedúra

Nastaví stav kanálu I2C komunikácie na príslušnom porte. Port musí byť konštanta (IN_1..IN_4).

```
SetLSChannelState(IN_1, LOWSPEED_IDLE);
```

LSErrorType(port) — Operácia

Vráti chybový kód komunikácie I2C. Port musí byť konštanta (IN_1..IN_4).

```
x = LSErrorType(IN_1);
```

SetLSErrorType(port, typChyby) — Procedúra

Nastaví chybový kód komunikácie I2C. Port musí byť konštanta (IN_1..IN_4).

```
SetLSErrorType(IN_1, LOWSPEED_CH_NOT_READY);
```

LSState() — Operácia

Vráti stav modulu I2C.

```
x = LSState();
```

SetLSState(n) — Procedúra

Nastaví stav modulu I2C.

```
SetLSState(COM_CHANNEL_THREE_ACTIVE);
```

LSSpeed() — Operácia

Vráti rýchlosť komunikácie I2C.

```
x = LSSpeed();
```

SetLSSpeed(n) — Procedúra

Nastaví rýchlosť komunikácie I2C.

```
SetLSSpeed(100);
```

3.12.3 Adresy položiek štruktúry IOMap (IOMap Offsets)

Konštanta	Hodnota	Veľkosť
LowSpeedOffsetInBufBuf(p)	(((p)*19)+0)	16
LowSpeedOffsetInBufInPtr(p)	(((p)*19)+16)	1
LowSpeedOffsetInBufOutPtr(p)	(((p)*19)+17)	1
LowSpeedOffsetInBufBytesToRx(p)	(((p)*19)+18)	58
LowSpeedOffsetOutBufBuf(p)	(((p)*19)+76)	16
LowSpeedOffsetOutBufInPtr(p)	(((p)*19)+92)	1
LowSpeedOffsetOutBufOutPtr(p)	(((p)*19)+93)	1
LowSpeedOffsetOutBufBytesToRx(p)	(((p)*19)+94)	58
LowSpeedOffsetMode(p)	((p)+152)	4
LowSpeedOffsetChannelState(p)	((p)+156)	4

LowSpeedOffsetErrorType(p)	((p)+160)	4
LowSpeedOffsetState	164	1
LowSpeedOffsetSpeed	165	1

Tabuľka 55. IOMap offsety modulu LowSpeed

3.13 Modul Comm

Modul Comm zabezpečuje všetky formy komunikácie cez BlueTooth, USB a vysokorýchlostný port (HiSpeed communication port).

Konštanta	Hodnota
CommModuleName	"Comm.mod"
CommModuleID	0x00050001

Tabuľka 56. Konštanty modulu Comm

Komunikácia cez BlueTooth slúži na komunikáciu s inými zariadeniami, ktoré sú ku kocke NXT pripojené rádiovým BlueTooth spojením. NXT firmware zabezpečuje uchovávanie správ v schránkach, ku ktorým možno pristupovať pomocou týchto metód.

Komunikácia cez Bluetooth využíva režim komunikácie master/slave. Predtým, ako program, ktorý komunikuje cez BlueTooth môže začat bežať, jedno zariadenie musí byť v roli master. NXT môže byť v roli master pripojené k trom rôznym zariadeniam, ktoré su v roli slave – budú prístupné na spojeniach označených 1, 2 a 3. NXT môže byť v roli slave iba v spojení s jedným iným zariadením a toto spojenie je označené číslom 0.

Programy, ktoré bežia na NXT, ktoré je nastavené v roli master, môžu posielať dátové pakety pripojeným NXT kockám, ktoré sú nastavené v roli slave, pomocou metódy BluetoothWrite. Zariadenia v režime slave posielajú svoje odpovedné pakety do svojho výstupného frontu správ, kde tieto správy čakajú, kým si ich pripojená kocka v roli master nevyzdvihne.

Pomocou protokolu priamych príkazov (direct command protocol), môže zariadenie v roli master posielať správy zariadeniu v roli slave vo forme textových reťazcov, ktoré sú uložené do príslušnej schránky. Každá schránka na strane zariadenia slave je kruhový front, ktorý môže udržať až päť správ. Každá správa môže byť najviac 58 znakov dlhá.

Správy z kocky master do kocky slave sa dajú poslať pomocou volania BluetoothWrite vyslaním priameho príkazu MessageWrite. Táto správa sa na kocke slave prijme pomocou volania ReceiveMessage. Na kocke v roli slave vtedy musí bežať program. V opačnom prípade bude správa ignorovaná a stratená.

3.13.1 Vysokoúrovňové volania

SendRemoteBool(spojenie, front, logickáHodnota) Operácia

Pošle logickú hodnotu cez zadané spojenie do príslušného frontu (schránky) na vzdialenej kocke.

```
x = SendRemoteBool(1, queue, false);
```

SendRemoteNumber(spojenie, front, hodnota) Operácia

Pošle číselnú správu cez zadané spojenie do príslušného frontu (schránky) na vzdialenej kocke.

```
x = SendRemoteNumber(1, queue, 123);
```

SendRemoteString(spojenie, front, reťazcováHodnota) Operácia

Pošle reťazcovú hodnotu na zadané spojenie do príslušného frontu (schránky) na vzdialenej kocke.

```
x = SendRemoteString(1, queue, "hello world");
```

SendResponseBool(front, logickáHodnota) Operácia

Pošle logickú hodnotu ako odpoveď na prijatú správu. Správa sa na kocke v roli slave uloží do zadaného frontu (schránky) +10, takže ju kocka v roli master môže vyzdvihnúť pri najbližšej požiadavke.

```
x = SendResponseBool(queue, false);
```

SendResponseNumber(front, hodnota) Operácia

Pošle číselnú hodnotu ako odpoveď na prijatú správu. Správa sa na kocke v roli slave uloží do zadaného frontu (schránky) +10, takže ju kocka v roli master môže vyzdvihnúť pri najbližšej požiadavke.

```
x = SendResponseNumber(queue, 123);
```

SendResponseString(front, reťazcováHodnota) Operácia

Pošle reťazcovú hodnotu ako odpoveď na prijatú správu. Správa sa na kocke v roli slave uloží do zadaného frontu (schránky) +10, takže ju kocka v roli master môže vyzdvihnúť pri najbližšej požiadavke.

```
x = SendResponseString(queue, "hello world");
```

ReceiveRemoteBool(front, odstrániť, out logickáHodnota) Operácia

Na kocke v roli master toto volanie vyzdvihne logickú správu z kocky v roli slave z požadovaného frontu (schránky) V prípade, že argument odstrániť je true, vyzdvihnutá správa bude odstránená zo schránky.

```
x = ReceiveRemoteBool(queue, true, bvalue);
```

ReceiveRemoteNumber(front, odstrániť, out hodnota) Operácia

Na kocke v roli master toto volanie vyzdvihne číselnú správu z kocky v roli slave z požadovaného frontu (schránky) V prípade, že argument odstrániť je true, vyzdvihnutá správa bude odstránená zo schránky.

```
x = ReceiveRemoteBool(queue, true, value);
```

ReceiveRemoteString(front, odstrániť, out reťazcováHodnota) Operácia

Na kocke v roli master toto volanie vyzdvihne reťazcovú správu z kocky v roli slave z požadovaného frontu (schránky) V prípade, že argument odstrániť je true, vyzdvihnutá správa bude odstránená zo schránky.

```
x = ReceiveRemoteString(queue, true, strval);
```

ReceiveRemoteMessageEx(front, odstrániť, out reťazcováHodnota, out hodnota, out logickáHodnota) Operácia

Na kocke v roli master toto volanie vyzdvihne reťazcovú, číselnú alebo logickú správu z kocky v roli slave z požadovaného frontu (schránky) V prípade, že argument odstrániť je true,

vyzdvihnutá správa bude odstránená zo schránky.

```
x = ReceiveRemoteMessageEx(queue, true, strval, val, bval);
```

SendMessage(front, správa) Operácia

Zapíše správu do požadovaného frontu (schránky). Maximálna dĺžka správy je 58 bajtov.

```
x = SendMessage(mbox, data);
```

ReceiveMessage(front, odstrániť, out buffer) Operácia

Prečíta správu z príslušného frontu (schránky) a uloží ju do premennej buffer. V prípade, že argument odstrániť je true, bude prečítaná správa odstránená zo schránky.

```
x = RecieveMessage(mbox, true, buffer);
```

BluetoothStatus(spojenie) Operácia

Vráti stav zadaného Bluetooth spojenia. Kým toto volanie vracia hodnotu `STAT_COMM_PENDING`, program by nemal používať volania BluetoothWrite, alebo akékoľvek iné, ktoré vysielajú data cez spojenie Bluetooth.

```
x = BluetoothStatus(1);
```

BluetoothWrite(spojenie, buffer) Operácia

Vyšle dáta v bufri na zadané bluetooth spojenie. Volanie BluetoothStatus sa dá použiť na určenie kedy bude požiadavka na vyslanie dát splnená.

```
x = BluetoothWrite(1, data);
```

RemoteMessageRead(spojenie, front) Operácia

Vyšle priamy príkaz (direct command) typu MessageRead na zariadenie, ktoré je pripojené na zadanom spojení. Pomocou volania BluetoothStatus je možné zistiť kedy je požiadavka splnená.

```
x = RemoteMessageRead(1, 5);
```

RemoteMessageWrite(spojenie, front, správa) Operácia

Vyšle priamy príkaz (direct command) MessageWrite na zariadenie, ktoré je pripojené na zadanom spojení. Pomocou volania BluetoothStatus je možné zistiť kedy je požiadavka splnená.

```
x = RemoteMessageWrite(1, 5, "test");
```

RemoteStartProgram(spojenie, menoSúboru) Operácia

Vyšle priamy príkaz (direct command) StartProgram na zariadenie pripojené na zadanom spojení. Pomocou volania BluetoothStatus je možné zistiť kedy je požiadavka splnená.

```
x = RemoteStartProgram(1, "myprog.rxe");
```

RemoteStopProgram(spojenie) Operácia

Vyšle priamy príkaz (direct command) StopProgram na zariadenie pripojené na zadanom spojení. Pomocou volania BluetoothStatus je možné zistiť kedy je požiadavka splnená.

```
x = RemoteStopProgram(1);
```

RemotePlaySoundFile(spojenie, menoSúboru, bOpakovať) Operácia

Vyšle priamy príkaz (direct command) PlaySoundFile na zariadenie pripojené na zadanom

spojení. Pomocou volania BluetoothStatus je možné zistiť kedy je požiadavka splnená.

```
x = RemotePlaySoundFile(1, "click.rso", false);
```

RemotePlayTone(spojenie, frekvencia, trvanie) Operácia

Vyšle priamy príkaz (direct command) PlayTone na zariadenie pripojené na zadanom spojení. Pomocou volania BluetoothStatus je možné zistiť kedy je požiadavka splnená.

```
x = RemotePlayTone(1, 440, 1000);
```

RemoteStopSound(spojenie) Operácia

Vyšle priamy príkaz (direct command) StopSound na zariadenie pripojené na zadanom spojení. Pomocou volania BluetoothStatus je možné zistiť kedy je požiadavka splnená.

```
x = RemoteStopSound(1);
```

RemoteKeepAlive(spojenie) Operácia

Vyšle priamy príkaz (direct command) KeepAlive na zariadenie pripojené na zadanom spojení. Pomocou volania BluetoothStatus je možné zistiť kedy je požiadavka splnená.

```
x = RemoteKeepAlive(1);
```

RemoteResetScaledValue(spojenie, port) Operácia

Vyšle priamy príkaz (direct command) ResetScaledValue na zariadenie pripojené na zadanom spojení. Pomocou volania BluetoothStatus je možné zistiť kedy je požiadavka splnená.

```
x = RemoteResetScaledValue(1, S1);
```

RemoteResetMotorPosition(spojenie, port, bRelatívne) Operácia

Vyšle priamy príkaz (direct command) ResetMotorPosition na zariadenie pripojené na zadanom spojení. Pomocou volania BluetoothStatus je možné zistiť kedy je požiadavka splnená.

```
x = RemoteResetMotorPosition(1, OUT_A, true);
```

RemoteSetInputMode(spojenie, port, typ, mód) Operácia

Vyšle priamy príkaz (direct command) SetInputMode na zariadenie pripojené na zadanom spojení. Pomocou volania BluetoothStatus je možné zistiť kedy je požiadavka splnená.

```
x = RemoteSetInputMode(1, S1, IN_TYPE_LOWSPEED, IN_MODE_RAW);
```

RemoteSetOutputState(spojenie, port, rýchlosť, mód, módRegulácie, smerovanie, runstate, tacholimit) Operácia

Vyšle priamy príkaz (direct command) SetOutputState na zariadenie pripojené na zadanom spojení. Pomocou volania BluetoothStatus je možné zistiť kedy je požiadavka splnená.

```
x = RemoteSetOutputState(1, OUT_A, 75, OUT_MODE_MOTORON,
      OUT_REGMODE_IDLE, 0, OUT_RUNSTATE_RUNNING, 0);
```

3.13.2 Nízkoúrovňové volania

Prípustné konštanty sú zobrazené v nasledujúcej tabuľke.

Konštanta	Hodnota
SIZE_OF_USBBUF	64
USB_PROTOCOL_OVERHEAD	2
SIZE_OF_USBDATA	62
SIZE_OF_HSBUF	128
SIZE_OF_BTBUF	128
BT_CMD_BYTE	1
SIZE_OF_BT_DEVICE_TABLE	30
SIZE_OF_BT_CONNECT_TABLE	4
SIZE_OF_BT_NAME	16
SIZE_OF_BRICK_NAME	8
SIZE_OF_CLASS_OF_DEVICE	4
SIZE_OF_BDADDR	7
MAX_BT_MSG_SIZE	60000
BT_DEFAULT_INQUIRY_MAX	0
BT_DEFAULT_INQUIRY_TIMEOUT_LO	15
LR_SUCCESS	0x50
LR_COULD_NOT_SAVE	0x51
LR_STORE_IS_FULL	0x52
LR_ENTRY_REMOVED	0x53
LR_UNKNOWN_ADDR	0x54
USB_CMD_READY	0x01
BT_CMD_READY	0x02
HS_CMD_READY	0x04

Tabuľka 57. Konštanty pre modul Comm

Prípustné stavy BtState sú uvedené v nasledujúcej tabuľke.

Konštanta	Hodnota
BT_ARM_OFF	0
BT_ARM_CMD_MODE	1
BT_ARM_DATA_MODE	2

Tabuľka 58. Konštanty BtState modulu Comm

Prípustné hodnoty pre BtStateStatus sú v nasledujúcej tabuľke.

Konštanta	Hodnota
BT_BRICK_VISIBILITY	0x01
BT_BRICK_PORT_OPEN	0x02
BT_CONNECTION_0_ENABLE	0x10
BT_CONNECTION_1_ENABLE	0x20
BT_CONNECTION_2_ENABLE	0x40
BT_CONNECTION_3_ENABLE	0x80

Tabuľka 59. Konštatny BtStateStatus modulu Comm

Prípustné hodnoty BtHwStatus sú v nasledujúcej tabuľke.

Konštanta	Hodnota
BT_ENABLE	0x00
BT_DISABLE	0x01

Tabuľka 60. Konštanty BtHwStatus modulu Comm

Prípustné hodnoty HsFlags sú v nasledujúcej tabuľke.

Konštanta	Hodnota
HS_UPDATE	1

Tabuľka 61. Konštanty HsFlags modulu Comm

Prípustné hodnoty HsState sú v nasledujúcej tabuľke.

Konštanta	Hodnota
HS_INITIALISE	1
HS_INIT_RECEIVER	2
HS_SEND_DATA	3
HS_DISABLE	4

Tabuľka 62. Konštanty HsState modulu Comm

Prípustné hodnoty DeviceStatus sú v nasledujúcej tabuľke.

Konštanta	Hodnota
BT_DEVICE_EMPTY	0x00
BT_DEVICE_UNKNOWN	0x01
BT_DEVICE_KNOWN	0x02
BT_DEVICE_NAME	0x40
BT_DEVICE_AWAY	0x80

Tabuľka 63. Konštanty DeviceStatus modulu Comm

Prípustné Interface hodnoty modulu Comm sú zobrazené v nasledujúcej tabuľke.

Konštanta	Hodnota
INTF_SENDFILE	0
INTF_SEARCH	1
INTF_STOPSEARCH	2
INTF_CONNECT	3
INTF_DISCONNECT	4
INTF_DISCONNECTALL	5
INTF_REMOVEDEVICE	6
INTF_VISIBILITY	7
INTF_SETCMDMODE	8
INTF_OPENSTREAM	9
INTF_SENDDATA	10
INTF_FACTORYRESET	11
INTF_BTON	12
INTF_BTOFF	13
INTF_SETBTNAME	14
INTF_EXTREAD	15
INTF_PINREQ	16
INTF_CONNECTREQ	17

Tabuľka 64. Interface konštanty modulu Comm

3.13.2.1 Volania pre obsluhu USB

GetUSBInputBuffer(adresa, počet, out data) Procedúra

Prečíta zadaný počet bajtov zo vstupnej vyrovnávacej pamäte USB na zadanej adrese a zapíše ich do premennej data.

```
GetUSBInputBuffer(0, 10, buffer);
```

SetUSBInputBuffer(adresa, počet, data) Procedúra

Zapíše zadaný počet bajtov z premennej data do vstupnej vyrovnávacej pamäte USB na zadanej adrese.

```
SetUSBInputBuffer(0, 10, buffer);
```

SetUSBInputBufferInPtr(n) Procedúra

Nastaví adresu čítania zo vstupnej vyrovnávacej pamäte USB na zadanú hodnotu.

```
SetUSBInputBufferInPtr(0);
```

USBInputBufferInPtr() Operácia

Vráti adresu čítania zo vstupnej vyrovnávacej pamäte USB.

```
byte x = USBInputBufferInPtr();
```

SetUSBInputBufferOutPtr(n) Procedúra

Nastaví adresu zapisovania do vstupnej vyrovnávacej pamäte USB na zadanú hodnotu.

```
SetUSBInputBufferOutPtr(0);
```

USBInputBufferOutPtr() Operácia

Vráti adresu zapisovania do vstupnej vyrovnávacej pamäte USB.

```
byte x = USBInputBufferOutPtr();
```

GetUSBOutputBuffer(adresa, počet, out data) Procedúra

Prečíta zadaný počet bajtov zo výstupnej vyrovnávacej pamäte USB na zadanej adrese a zapíše ich do premennej data.

```
GetUSBOutputBuffer(0, 10, buffer);
```

SetUSBOutputBuffer(adresa, počet, data) Procedúra

Zapíše zadaný počet bajtov z premennej data do výstupnej vyrovnávacej pamäte USB na zadanej adrese.

```
SetUSBOutputBuffer(0, 10, buffer);
```

SetUSBOutputBufferInPtr(n) Procedúra

Nastaví adresu čítania z výstupnej vyrovnávacej pamäte USB na zadanú hodnotu.

```
SetUSBOutputBufferInPtr(0);
```

USBOutputBufferInPtr() Operácia

Vráti adresu čítania z výstupnej vyrovnávacej pamäte USB.

```
byte x = USBOutputBufferInPtr();
```

SetUSBOutputBufferOutPtr(n) Procedúra

Nastaví adresu zapisovania do výstupnej vyrovnávacej pamäte USB na zadanú hodnotu.

```
SetUSBOutputBufferOutPtr(0);
```

USBOutputBufferOutPtr() Operácia

Vráti adresu zapisovania do výstupnej vyrovnávacej pamäte USB.

```
byte x = USBOutputBufferOutPtr();
```

GetUSBPollBuffer(adresa, počet, out data) Procedúra

Prečíta zadaný počet bajtov z „poll“-ovacej vyrovnávacej pamäte USB a zapíše ich do premennej data.

```
GetUSBPollBuffer(0, 10, buffer);
```

SetUSBPollBuffer(adresa, počet, data) Procedúra

Zapíše zadaný počet bajtov z premennej data do „poll“-ovacej vyrovnávacej pamäte USB na zadanej adrese.

```
SetUSBPollBuffer(0, 10, buffer);
```

SetUSBPollBufferInPtr(n) Procedúra

Nastaví adresu čítania z „poll“-ovacej vyrovnávacej pamäte USB na zadanú hodnotu.

```
SetUSBPollBufferInPtr(0);
```

USBPollBufferInPtr() Operácia

Vráti adresu čítania z „poll“-ovacej vyrovnávacej pamäte USB.

```
byte x = USBPollBufferInPtr();
```

SetUSBPollBufferOutPtr(n) Procedúra

Nastaví adresu zapisovania do „poll“-ovacej vyrovnávacej pamäte USB na zadanú hodnotu.

```
SetUSBPollBufferOutPtr(0);
```

USBPollBufferOutPtr() Operácia

Vráti adresu zapisovania do „poll“-ovacej vyrovnávacej pamäte USB.

```
byte x = USBPollBufferOutPtr();
```

SetUSBState(n) Procedúra

Nastaví stav USB na požadovanú hodnotu.

```
SetUSBState(0);
```

USBState() Operácia

Vráti stav USB.

```
byte x = USBPollBufferOutPtr();
```

3.13.2.2 Volania pre obsluhu vysokorýchlostného (High-speed) portu

GetHSInputBuffer(offset, count, out data) Procedúra

Prečíta zadaný počet bajtov zo vstupnej vyrovnávacej pamäte vysokorýchlostného portu (High-speed) a zapíše ich do premennej data.

```
GetHSInputBuffer(0, 10, buffer);
```

SetHSInputBuffer(offset, count, data) Procedúra

Zapíše zadaný počet bajtov z premennej data do vstupnej vyrovnávacej pamäte vysokorýchlostného portu (High-speed) na zadanej adrese.

```
SetHSInputBuffer(0, 10, buffer);
```

SetHSInputBufferInPtr(n) Procedúra

Nastaví adresu čítania zo vstupnej vyrovnávacej pamäte vysokorýchlostného portu (High-speed) na zadanú hodnotu.

```
SetHSInputBufferInPtr(0);
```

HSInputBufferInPtr() Operácia

Vráti adresu čítania z vstupnej vyrovnávacej pamäte vysokorýchlostného portu (High-speed).

```
byte x = HSInputBufferInPtr();
```

SetHSInputBufferOutPtr(n) Procedúra

Nastaví adresu zapisovania do vstupnej vyrovnávacej pamäte vysokorýchlostného portu (High-speed) na zadanú hodnotu.

```
SetHSInputBufferOutPtr(0);
```

HSInputBufferOutPtr() Operácia

Vráti adresu zapisovania do vstupnej vyrovnávacej pamäte vysokorýchlostného portu (High-speed).

```
byte x = HSInputBufferOutPtr();
```

GetHSOutputBuffer(offset, count, out data) Procedúra

Prečíta zadaný počet bajtov z výstupnej vyrovnávacej pamäte vysokorýchlostného portu (High-speed) a zapíše ich do premennej data.

```
GetHSOutputBuffer(0, 10, buffer);
```

SetHSOutputBuffer(offset, count, data) Procedúra

Zapíše zadaný počet bajtov z premennej data do výstupnej vyrovnávacej pamäte vysokorýchlostného portu (High-speed) na zadanej adrese.

```
SetHSOutputBuffer(0, 10, buffer);
```

SetHSOutputBufferInPtr(n) Procedúra

Nastaví adresu čítania z výstupnej vyrovnávacej pamäte vysokorýchlostného portu (High-speed) na zadanú hodnotu.

```
SetHSOutputBufferInPtr(0);
```

HSOutputBufferInPtr() Operácia

Vráti adresu čítania z výstupnej vyrovnávacej pamäte vysokorýchlostného portu (High-speed).

```
byte x = HSOutputBufferInPtr();
```

SetHSOutputBufferOutPtr(n) Procedúra

Nastaví adresu zápisu do výstupnej vyrovnávacej pamäte vysokorýchlostného portu (High-speed) na zadanú hodnotu.

```
SetHSOutputBufferOutPtr(0);
```

HSOutputBufferOutPtr() Operácia

Vráti adresu zápisu do výstupnej vyrovnávacej pamäte vysokorýchlostného portu (High-speed).

```
byte x = HSOutputBufferOutPtr();
```

SetHSFlags(n) Procedúra

Nastaví príznaky vysokorýchlostného portu (High-speed) na zadanú hodnotu.

```
SetHSFlags(0);
```

HSFlags() Operácia

Vráti príznaky vysokorýchlostného portu.

```
byte x = HSFlags();
```

SetHSSpeed(n) Procedúra

Nastaví rýchlosť vysokorýchlostného portu (High-speed) na zadanú hodnotu.

```
SetHSSpeed(1);
```

HSSpeed() Operácia

Vráti rýchlosť vysokorýchlostného portu (High-speed).

```
byte x = HSSpeed();
```

SetHSState(n) Procedúra

Nastaví stav vysokorýchlostného portu (High-speed) na zadanú hodnotu.

```
SetHSState(1);
```

HSState() Operácia

Vráti stav vysokorýchlostného portu (High-speed).

```
byte x = HSState();
```

3.13.2.3 Volania pre obsluhu Bluetooth

GetBTInputBuffer(offset, count, out data) Procedúra

Prečíta zadaný počet bajtov zo vstupnej vyrovnávacej pamäte BlueTooth a zapíše ich do premennej data.

```
GetBTInputBuffer(0, 10, buffer);
```

SetBTInputBuffer(offset, count, data) Procedúra

Zapíše zadaný počet bajtov z premennej data do vstupnej vyrovnávacej pamäte BlueTooth na zadanej adrese.

```
SetBTInputBuffer(0, 10, buffer);
```

SetBTInputBufferInPtr(n) Procedúra

Nastaví adresu čítania zo vstupnej vyrovnávacej pamäte BlueTooth na zadanú hodnotu.

```
SetBTInputBufferInPtr(0);
```

BTInputBufferInPtr() Operácia

Vráti adresu čítania zo vstupnej vyrovnávacej pamäte BlueTooth.

```
byte x = BTInputBufferInPtr();
```

SetBTInputBufferOutPtr(n) Procedúra

Nastaví adresu zápisu do vstupnej vyrovnávacej pamäte BlueTooth na zadanú hodnotu.

```
SetBTInputBufferOutPtr(0);
```

BTInputBufferOutPtr() Operácia

Vráti adresu zápisu do vstupnej vyrovnávacej pamäte BlueTooth.

```
byte x = BTInputBufferOutPtr();
```

GetBTOutputBuffer(offset, count, out data) Procedúra

Prečíta zadaný počet bajtov z výstupnej vyrovnávacej pamäte BlueTooth a zapíše ich do premennej data.

```
GetBTOutputBuffer(0, 10, buffer);
```

SetBTOutputBuffer(offset, count, data) Procedúra

Zapíše zadaný počet bajtov z premennej data do výstupnej vyrovnávacej pamäte BlueTooth na zadanej adrese.

```
SetBTOutputBuffer(0, 10, buffer);
```

SetBTOutputBufferInPtr(n) Procedúra

Nastaví adresu čítania z výstupnej vyrovnávacej pamäte BlueTooth na zadanú hodnotu.

```
SetBTOutputBufferInPtr(0);
```

BTOutputBufferInPtr() Operácia

Vráti adresu čítania z výstupnej vyrovnávacej pamäte BlueTooth.

```
byte x = BTOutputBufferInPtr();
```

SetBTOutputBufferOutPtr(n) Procedúra

Nastaví adresu zápisu do výstupnej vyrovnávacej pamäte BlueTooth na zadanú hodnotu.

```
SetBTOutputBufferOutPtr(0);
```

BTOutputBufferOutPtr() Operácia

Vráti adresu zápisu z výstupnej vyrovnávacej pamäte BlueTooth.

```
byte x = BTOutputBufferOutPtr();
```

BTDeviceCount() Operácia

Vráti počet zariadení definovaných v tabuľke zariadení BlueTooth.

```
byte x = BTDeviceCount();
```

BTDeviceNameCount() Operácia

Vráti počet mien zariadení definovaných v tabuľke zariadení BlueTooth. Táto hodnota je väčšinou rovnaká, ako BTDeviceCount, ale v niektorých prípadoch sa môže líšiť.

```
byte x = BTDeviceNameCount();
```

BTDeviceName(idx) Operácia

Vráti meno zariadenia na požadovanom indexe v tabuľke zariadení BlueTooth.

```
string name = BTDeviceName(0);
```

BTConnectionName(idx) Operácia

Vráti meno zariadenia na zadanom indexe v tabuľke spojení BlueTooth.

```
string name = BTConnectionName(0);
```

BTConnectionPinCode(idx) Operácia

Vráti pin kód zariadenia na zadanom indexe v tabuľke spojení BlueTooth.

```
string pincode = BTConnectionPinCode(0);
```

BrickDataName() Operácia

Vráti meno NXT.

```
string name = BrickDataName();
```

GetBTDeviceAddress(idx, out data) Procedúra

Vráti adresu zariadenia na zadanom indexe v tabuľke zariadení BlueTooth a uloží ju do premennej data.

```
GetBTDeviceAddress(0, buffer);
```

GetBTConnectionAddress(idx, out data) Procedúra

Vráti adresu zariadenia na zadanom indexe v tabuľke spojení BlueTooth a uloží ju do premennej data.

```
GetBTConnectionAddress(0, buffer);
```

GetBrickDataAddress(out data) Procedúra

Prečíta adresu kocky NXT a zapíše ju do premennej data.

```
GetBrickDataAddress(buffer);
```

BTDeviceClass(idx) Operácia

Prečíta triedu zariadenia (BlueTooth class) na zadanom indexe v tabuľke zariadení BlueTooth.

```
long class = BTDeviceClass(idx);
```

BTDeviceStatus(idx) Operácia

Vráti stav zariadenia na zadanom indexe v tabuľke zariadení BlueTooth.

```
byte status = BTDeviceStatus(idx);
```

BTConnectionClass(idx) Operácia

Vráti triedu zariadenia (BlueTooth class) na zadanom indexe v tabuľke spojení BlueTooth.

```
long class = BTConnectionClass(idx);
```

BTConnectionHandleNum(idx) Operácia

Vráti handle zariadenia na zadanom indexe v tabuľke spojení BlueTooth.

```
byte handlenum = BTConnectionHandleNum(idx);
```

BTConnectionStreamStatus(idx) Operácia

Vráti stav „stream“-u zariadenia na zadanom indexe v tabuľke spojení BlueTooth.

```
byte streamstatus = BTConnectionStreamStatus(idx);
```

BTConnectionLinkQuality(idx) Operácia

Vráti kvalitu spojenia zariadenia na zadanom indexe v tabuľke spojení BlueTooth.

```
byte linkquality = BTConnectionLinkQuality(idx);
```

BrickDataBluecoreVersion() Operácia

Vráti verziu bluecore v kocke NXT.

```
int bv = BrickDataBluecoreVersion();
```

BrickDataBtStateStatus() Operácia

Vráti stav Bluetooth kocky NXT.

```
int x = BrickDataBtStateStatus();
```

BrickDataBtHardwareStatus() Operácia

Vráti hardvérový stav Bluetooth kocky NXT.

```
int x = BrickDataBtHardwareStatus();
```

BrickDataTimeoutValue() Operácia

Vráti hodnotu timeout pre kocku NXT.

```
int x = BrickDataTimeoutValue();
```

3.13.3 Adresy položiek štruktúry IOMap (IOMap Offsets)

Konštanta	Hodnota	Veľkosť
CommOffsetPFunc	0	4
CommOffsetPFuncTwo	4	4
CommOffsetBtDeviceTableName(p)	(((p)*31)+8)	16
CommOffsetBtDeviceTableClassOfDevice(p)	(((p)*31)+24)	4
CommOffsetBtDeviceTableBdAddr(p)	(((p)*31)+28)	7
CommOffsetBtDeviceTableDeviceStatus(p)	(((p)*31)+35)	1
CommOffsetBtConnectTableName(p)	(((p)*47)+938)	16
CommOffsetBtConnectTableClassOfDevice	(p) (((p)*47)+954)	4
CommOffsetBtConnectTablePinCode(p)	(((p)*47)+958)	16
CommOffsetBtConnectTableBdAddr(p)	(((p)*47)+974)	7
CommOffsetBtConnectTableHandleNr(p)	(((p)*47)+981)	1
CommOffsetBtConnectTableStreamStatus(p)	(((p)*47)+982)	1
CommOffsetBtConnectTableLinkQuality(p)	(((p)*47)+983)	1
CommOffsetBrickDataName	1126	16
CommOffsetBrickDataBluecoreVersion	1142	2
CommOffsetBrickDataBdAddr	1144	7
CommOffsetBrickDataBtStateStatus	1151	1
CommOffsetBrickDataBtHwStatus	1152	1
CommOffsetBrickDataTimeOutValue	1153	1
CommOffsetBtInBufBuf	1157	128
CommOffsetBtInBufInPtr	1285	1
CommOffsetBtInBufOutPtr	1286	1

CommOffsetBtOutBufBuf	1289	128
CommOffsetBtOutBufInPtr	1417	1
CommOffsetBtOutBufOutPtr	1418	1
CommOffsetHsInBufBuf	1421	128
CommOffsetHsInBufInPtr	1549	1
CommOffsetHsInBufOutPtr	1550	1
CommOffsetHsOutBufBuf	1553	128
CommOffsetHsOutBufInPtr	1681	1
CommOffsetHsOutBufOutPtr	1682	1
CommOffsetUsbInBufBuf	1685	64
CommOffsetUsbInBufInPtr	1749	1
CommOffsetUsbInBufOutPtr	1750	1
CommOffsetUsbOutBufBuf	1753	64
CommOffsetUsbOutBufInPtr	1817	1
CommOffsetUsbOutBufOutPtr	1818	1
CommOffsetUsbPollBufBuf	1821	64
CommOffsetUsbPollBufInPtr	1885	1
CommOffsetUsbPollBufOutPtr	1886	1
CommOffsetBtDeviceCnt	1889	1
CommOffsetBtDeviceNameCnt	1890	1
CommOffsetHsFlags	1891	1
CommOffsetHsSpeed	1892	1
CommOffsetHsState	1893	1
CommOffsetUsbState	1894	1

Tabuľka 65. Offsety modulu Comm

3.14 Volania API pre zariadenia HiTechnic

SensorHTCompass(port) Operácia

Prečíta smer z kompasového senzora HiTechnic na zadanom porte.

```
x = SensorHTCompass(S1);
```

SensorHTIRSeekerDir(port) Operácia

Prečíta smer zo senzora HiTechnic na hladanie infračerveného signálu (IR Seeker) na zadanom porte.

```
x = SensorHTIRSeekerDir(S1);
```

SensorHTColorNum(port) Operácia

Prečíta číslo farby z farebného senzora HiTechnic na zadanom porte.

```
x = SensorHTColorNum(S1);
```

SetSensorHTGyro(port) Procedúra

Nastaví senzor na zadanom porte ako HiTechnic Gyro.

```
SetSensorHTGyro(S1);
```

SensorHTGyro(port, priemer) Operácia

Prečíta hodnotu z HiTechnic Gyro senzora na zadanom porte. Hodnota priemeru musí byť vypočítaná spriemerovaním viacerých hodnôt pokiaľ je senzor v dokonalom pokoji, prečítaných s nulovou hodnotou priemeru.

```
x = SensorHTGyro(S1, gyroOffset);
```

ReadSensorHTAccel(port, x, y, z) Operácia

Prečíta hodnoty zrýchlenia podľa osí X, Y, a Z zo senzora HiTechnic Accelerometer. Vráti logickú hodnotu, ktorá udáva, či operácia čítania bola úspešná.

```
bVal = ReadSensorHTAccel(S1, x, y, z);
```

ReadSensorHTColor(port, ČísloFarby, Červená, Zelená, Modrá) Operácia

Prečíta číslo farby, červenú, zelenú a modrú zložku z farebného senzora HiTechnic. Vráti logickú hodnotu, ktorá udáva, či operácia čítania bola úspešná.

```
bVal = ReadSensorHTColor(S1, c, r, g, b);
```

ReadSensorHTRawColor(port, Červená, Zelená, Modrá) Operácia

Prečíta nespracovanú červenú, zelenú a modrú zložku z farebného senzora HiTechnic. Vráti logickú hodnotu, ktorá udáva, či operácia čítania bola úspešná.

```
bVal = ReadSensorHTRawColor(S1, r, g, b);
```

ReadSensorHTNormalizedColor(port, Idx, Červená, Zelená, Modrá) Operácia

Prečíta číslo farby, a normalizovanú červenú, zelenú a modrú zložku z farebného senzora HiTechnic. Vráti logickú hodnotu, ktorá udáva, či operácia čítania bola úspešná.

```
bVal = ReadSensorHTNormalizedColor(S1, c, r, g, b);
```

ReadSensorHTIRSeeker(port, smer, s1, s3, s5, s7, s9) Operácia

Prečíta smer a päť hodnôt intenzity IR signálu zo senzora HiTechnic na vyhľadávanie infračerveného signálu (IR Seeker). Vráti logickú hodnotu, ktorá udáva, či operácia čítania bola úspešná.

```
bVal = ReadSensorHTIRSeeker(port, dir, s1, s3, s5, s7, s9);
```

HTPowerFunctionCommand(port, kanál, príkaz1, príkaz2) Procedúra

Vykoná dvojicu motorových príkazov (Power Function) na zadanom kanáli použitím zariadenia HiTechnic iRLink. Možné príkazy sú HTPF_CMD_STOP, HTPF_CMD_REV, HTPF_CMD_FWD, a HTPF_CMD_BRAKE. Prípustné kanály sú HTPF_CHANNEL_1 až HTPF_CHANNEL_4.

```
HTPowerFunctionCommand(S1, HTPF_CHANNEL_1, HTPF_CMD_STOP,
     HTPF_CMD_FWD);
```

HTRCXSetIRLinkPort(port) — Procedúra

Vopred nastaví globálny port pre volania API s menami HTRCX* a HTScout*, ktoré slúžia na komunikáciu s RCX alebo Scout cez zariadenie HiTechnic iRLink.

```
HTRCXSetIRLinkPort(S1);
```

HTRCXPoll(src, hodnota) — Operácia

Vyšle na RCX príkaz Poll za účelom prečítania dvojbajtovej hodnoty na zadanej kombinácii src a hodnota.

```
x = HTRCXPoll(RCX_VariableSrc, 0);
```

HTRCXBatteryLevel() — Operácia

Vyšle na RCX príkaz BatteryLevel a prečíta aktuálny stav batérie.

```
x = HTRCXBatteryLevel();
```

HTRCXPing() — Procedúra

Vyšle na RCX príkaz Ping.

```
HTRCXPing();
```

HTRCXDeleteTasks() — Procedúra

Vyšle na RCX príkaz DeleteTasks.

```
HTRCXDeleteTasks();
```

HTRCXStopAllTasks() — Procedúra

Vyšle na RCX príkaz StopAllTasks.

```
HTRCXStopAllTasks();
```

HTRCXPBTurnOff() — Procedúra

Vyšle na RCX príkaz PBTurnOff.

```
HTRCXPBTurnOff();
```

HTRCXDeleteSubs() — Procedúra

Vyšle na RCX príkaz DeleteSubs.

```
HTRCXDeleteSubs();
```

HTRCXClearSound() — Procedúra

Vyšle na RCX príkaz ClearSound.

```
HTRCXClearSound();
```

HTRCXClearMsg() — Procedúra

Vyšle na RCX príkaz ClearMsg.

```
HTRCXClearMsg();
```

HTRCXMuteSound() — Procedúra

Vyšle na RCX príkaz MuteSound.

```
HTRCXMuteSound();
```

HTRCXUnmuteSound() Procedúra

Vyšle na RCX príkaz UnmuteSound.

```
HTRCXUnmuteSound();
```

HTRCXClearAllEvents() Procedúra

Vyšle na RCX príkaz ClearAllEvents.

```
HTRCXClearAllEvents();
```

HTRCXSetOutput(výstupy, mód) Procedúra

Vyšle na RCX príkaz SetOutput na nastavenie módu na zadaných výstupoch.

```
HTRCXSetOutput(RCX_OUT_A, RCX_OUT_ON);
```

HTRCXSetDirection(výstupy, smer) Procedúra

Vyšle na RCX príkaz SetDirection na nastavenie smeru na zadaných výstupoch.

```
HTRCXSetDirection(RCX_OUT_A, RCX_OUT_FWD);
```

HTRCXSetPower(výstupy, pwrsrc, pwrval) Procedúra

Vyšle na RCX príkaz SetPower na nastavenie sily na zadaných výstupoch.

```
HTRCXSetPower(RCX_OUT_A, RCX_ConstantSrc, RCX_OUT_FULL);
```

HTRCXOn(výstupy) Procedúra

Vyšle na RCX príkazy na zapnutie požadovaných výstupov.

```
HTRCXOn(RCX_OUT_A);
```

HTRCXOff(výstupy) Procedúra

Vyšle na RCX príkazy na vypnutie (s brzdou) požadovaných výstupov.

```
HTRCXOff(RCX_OUT_A);
```

HTRCXFloat(výstupy) Procedúra

Vyšle na RCX príkazy na vypnutie (so zotrvačnosťou) požadovaných výstupov.

```
HTRCXFloat(RCX_OUT_A);
```

HTRCXToggle(výstupy) Procedúra

Vyšle na RCX príkazy na zmenu smeru na požadovaných výstupoch.

```
HTRCXToggle(RCX_OUT_A);
```

HTRCXFwd(výstupy) Procedúra

Vyšle na RCX príkazy na nastavenie požadovaných výstupov na dopredný chod.

```
HTRCXFwd(RCX_OUT_A);
```

HTRCXRev(výstupy) Procedúra

Vyšle na RCX príkazy na nastavenie požadovaných výstupov na spätný chod.

```
HTRCXRev(RCX_OUT_A);
```

HTRCXOnFwd(výstupy) Procedúra

Vyšle na RCX príkazy na zapnutie požadovaných výstupov, pričom sa motory budú otáčať v doprednom smere.

```
HTRCXOnFwd(RCX_OUT_A);
```

HTRCXOnRev(výstupy) Procedúra

Vyšle na RCX príkazy na zapnutie požadovaných výstupov, pričom sa motory budú otáčať v spätnom smere.

```
HTRCXOnRev(RCX_OUT_A);
```

HTRCXOnFor(výstupy, trvanie) Procedúra

Vyšle na RCX príkazy na zapnutie požadovaných výstupov na zadanu dobu.

```
HTRCXOnFor(RCX_OUT_A, 100);
```

HTRCXSetTxPower(pwr) Procedúra

Vyšle na RCX príkaz SetTxPower.

```
HTRCXSetTxPower(0);
```

HTRCXPlaySound(snd) Procedúra

Vyšle na RCX príkaz PlaySound.

```
HTRCXPlaySound(RCX_SOUND_UP);
```

HTRCXDeleteTask(n) Procedúra

Vyšle na RCX príkaz DeleteTask.

```
HTRCXDeleteTask(3);
```

HTRCXStartTask(n) Procedúra

Vyšle na RCX príkaz StartTask.

```
HTRCXStartTask(2);
```

HTRCXStopTask(n) Procedúra

Vyšle na RCX príkaz StopTask.

```
HTRCXStopTask(1);
```

HTRCXSelectProgram(prog) Procedúra

Vyšle na RCX príkaz SelectProgram.

```
HTRCXSelectProgram(3);
```

HTRCXClearTimer(timer) Procedúra

Vyšle na RCX príkaz ClearTimer.

```
HTRCXClearTimer(0);
```

HTRCXSetSleepTime(t) Procedúra

Vyšle na RCX príkaz SetSleepTime.

```
HTRCXSetSleepTime(4);
```

HTRCXDeleteSub(s) — Procedúra

Vyšle na RCX príkaz DeleteSub.

```
HTRCXDeleteSub(2);
```

HTRCXClearSensor(port) — Procedúra

Vyšle na RCX príkaz ClearSensor.

```
HTRCXClearSensor(S1);
```

HTRCXPlayToneVar(čísloPremennej, trvanie) — Procedúra

Vyšle na RCX príkaz PlayToneVar.

```
HTRCXPlayToneVar(0, 50);
```

HTRCXSetWatch(hodiny, minúty) — Procedúra

Vyšle na RCX príkaz SetWatch.

```
HTRCXSetWatch(3, 30);
```

HTRCXSetSensorType(port, typ) — Procedúra

Vyšle na RCX príkaz SetSensorType.

```
HTRCXSetSensorType(S1, SENSOR_TYPE_TOUCH);
```

HTRCXSetSensorMode(port, mód) — Procedúra

Vyšle na RCX príkaz SetSensorMode.

```
HTRCXSetSensorMode(S1, SENSOR_MODE_BOOL);
```

HTRCXCreateDatalog(veľkosť) — Procedúra

Vyšle na RCX príkaz CreateDatalog.

```
HTRCXCreateDatalog(50);
```

HTRCXAddToDatalog(src, value) — Procedúra

Vyšle na RCX príkaz AddToDatalog.

```
HTRCXAddToDatalog(RCX_InputValueSrc, S1);
```

HTRCXSendSerial(first, count) — Procedúra

Vyšle na RCX príkaz SendSerial.

```
HTRCXSendSerial(0, 10);
```

HTRCXRemote(cmd) — Procedúra

Vyšle na RCX príkaz Remote.

```
HTRCXRemote(RCX_RemotePlayASound);
```

HTRCXEvent(zdroj, hodnota) — Procedúra

Vyšle na RCX príkaz Event.

```
HTRCXEvent(RCX_ConstantSrc, 2);
```

HTRCXPlayTone(frekvencia, trvanie) — Procedúra

Vyšle na RCX príkaz PlayTone.

```
HTRCXPlayTone(440, 100);
```

HTRCXSelectDisplay(zdroj, hodnota) — Procedúra

Vyšle na RCX príkaz SelectDisplay.

```
HTRCXSelectDisplay(RCX_VariableSrc, 2);
```

HTRCXPollMemory(adresa, počet) — Procedúra

Vyšle na RCX príkaz PollMemory.

```
HTRCXPollMemory(0, 10);
```

HTRCXSetEvent(event, zdroj, typ) — Procedúra

Vyšle na RCX príkaz SetEvent.

```
HTRCXSetEvent(0, RCX_ConstantSrc, 5);
```

HTRCXSetGlobalOutput(výstupy, mód) — Procedúra

Vyšle na RCX príkaz SetGlobalOutput.

```
HTRCXSetGlobalOutput(RCX_OUT_A, RCX_OUT_ON);
```

HTRCXSetGlobalDirection(výstupy, smer) — Procedúra

Vyšle na RCX príkaz SetGlobalDirection.

```
HTRCXSetGlobalDirection(RCX_OUT_A, RCX_OUT_FWD);
```

HTRCXSetMaxPower(výstupy, pwrsrc, pwrval) — Procedúra

Vyšle na RCX príkaz SetMaxPower.

```
HTRCXSetMaxPower(RCX_OUT_A, RCX_ConstantSrc, 5);
```

HTRCXEnableOutput(výstupy) — Procedúra

Vyšle na RCX príkaz EnableOutput.

```
HTRCXEnableOutput(RCX_OUT_A);
```

HTRCXDisableOutput(výstupy) — Procedúra

Vyšle na RCX príkaz DisableOutput.

```
HTRCXDisableOutput(RCX_OUT_A);
```

HTRCXInvertOutput(výstupy) — Procedúra

Vyšle na RCX príkaz InvertOutput.

```
HTRCXInvertOutput(RCX_OUT_A);
```

HTRCXObvertOutput(výstupy) — Procedúra

Vyšle na RCX príkaz ObvertOutput.

```
HTRCXObvertOutput(RCX_OUT_A);
```

HTRCXCalibrateEvent(event, dolnýPrah, hornýPrah, hysterézia) Procedúra

Vyšle na RCX príkaz CalibrateEvent.

```
HTRCXCalibrateEvent(0, 200, 500, 50);
```

HTRCXSetVar(čísloPremennej, zdroj, hodnota) Procedúra

Vyšle na RCX príkaz SetVar.

```
HTRCXSetVar(0, RCX_VariableSrc, 1);
```

HTRCXSumVar(čísloPremenej, zdroj, hodnota) Procedúra

Vyšle na RCX príkaz SumVar.

```
HTRCXSumVar(0, RCX_InputValueSrc, S1);
```

HTRCXSubVar(čísloPremennej, zdroj, hodnota) Procedúra

Vyšle na RCX príkaz SubVar.

```
HTRCXSubVar(0, RCX_RandomSrc, 10);
```

HTRCXDivVar(čísloPremennej, zdroj, hodnota) Procedúra

Vyšle na RCX príkaz DivVar.

```
HTRCXDivVar(0, RCX_ConstantSrc, 2);
```

HTRCXMulVar(čísloPremennej, zdroj, hodnota) Procedúra

Vyšle na RCX príkaz MulVar.

```
HTRCXMulVar(0, RCX_VariableSrc, 4);
```

HTRCXSgnVar(čísloPremennej, zdroj, hodnota) Procedúra

Vyšle na RCX príkaz SgnVar.

```
HTRCXSgnVar(0, RCX_VariableSrc, 0);
```

HTRCXAbsVar(čísloPremennej, zdroj, hodnota) Procedúra

Vyšle na RCX príkaz AbsVar.

```
HTRCXAbsVar(0, RCX_VariableSrc, 0);
```

HTRCXAndVar(čísloPremennej, zdroj, hodnota) Procedúra

Vyšle na RCX príkaz AndVar.

```
HTRCXAndVar(0, RCX_ConstantSrc, 0x7f);
```

HTRCXOrVar(čísloPremennej, zdroj, hodnota) Procedúra

Vyšle na RCX príkaz OrVar.

```
HTRCXOrVar(0, RCX_ConstantSrc, 0xCC);
```

HTRCXSet(dstsrc, dstval, zdroj, hodnota) Procedúra

Vyšle na RCX príkaz Set.

```
HTRCXSet(RCX_VariableSrc, 0, RCX_RandomSrc, 10000);
```

HTRCXUnlock() Procedúra

Vyšle na RCX príkaz Unlock.

```
HTRCXUnlock();
```

HTRCXReset() Procedúra

Vyšle na RCX príkaz Reset.

```
HTRCXReset();
```

HTRCXBoot() Procedúra

Vyšle na RCX príkaz Boot.

```
HTRCXBoot();
```

HTRCXSetUserDisplay(zdroj, hodnota, presnosť) Procedúra

Vyšle na RCX príkaz SetUserDisplay.

```
HTRCXSetUserDisplay(RCX_VariableSrc, 0, 2);
```

HTRCXIncCounter(počítadlo) Procedúra

Vyšle na RCX príkaz IncCounter.

```
HTRCXIncCounter(0);
```

HTRCXDecCounter(počítadlo) Procedúra

Vyšle na RCX príkaz DecCounter.

```
HTRCXDecCounter(0);
```

HTRCXClearCounter(počítadlo) Procedúra

Vyšle na RCX príkaz ClearCounter.

```
HTRCXClearCounter(0);
```

HTRCXSetPriority(p) Procedúra

Vyšle na RCX príkaz SetPriority.

```
HTRCXSetPriority(2);
```

HTRCXSetMessage(správa) Procedúra

Vyšle na RCX príkaz SetMessage.

```
HTRCXSetMessage(20);
```

HTScoutCalibrateSensor() Procedúra

Vyšle na Scout príkaz CalibrateSensor.

```
HTScoutCalibrateSensor();
```

HTScoutMuteSound() Procedúra

Vyšle na Scout príkaz MuteSound.

```
HTScoutMuteSound();
```

HTScoutUnmuteSound() — Procedúra

Vyšle na Scout príkaz UnmuteSound.

```
HTScoutUnmuteSound();
```

HTScoutSelectSounds(skupina) — Procedúra

Vyšle na Scout príkaz SelectSounds.

```
HTScoutSelectSounds(0);
```

HTScoutSetLight(mód) — Procedúra

Vyšle na Scout príkaz SetLight.

```
HTScoutSetLight(SCOUT_LIGHT_ON);
```

HTScoutSetCounterLimit(počítadlo, zdroj, hodnota) — Procedúra

Vyšle na Scout príkaz SetCounterLimit.

```
HTScoutSetCounterLimit(0, RCX_ConstantSrc, 2000);
```

HTScoutSetTimerLimit(časovač, zdroj, hodnota) — Procedúra

Vyšle na Scout príkaz SetTimerLimit.

```
HTScoutSetTimerLimit(0, RCX_ConstantSrc, 10000);
```

HTScoutSetSensorClickTime(zdroj, hodnota) — Procedúra

Vyšle na Scout príkaz SetSensorClickTime.

```
HTScoutSetSensorClickTime(RCX_ConstantSrc, 200);
```

HTScoutSetSensorHysteresis(zdroj, hodnota) — Procedúra

Vyšle na Scout príkaz SetSensorHysteresis.

```
HTScoutSetSensorHysteresis(RCX_ConstantSrc, 50);
```

HTScoutSetSensorLower Limit(zdroj, hodnota) — Procedúra

Vyšle na Scout príkaz SetSensorLowerLimit.

```
HTScoutSetSensorLower Limit(RCX_ConstantSrc, 100);
```

HTScoutSetSensorUpper Limit(zdroj, hodnota) — Procedúra

Vyšle na Scout príkaz SetSensorUpperLimit.

```
HTScoutSetSensorUpper Limit(RCX_ConstantSrc, 400);
```

HTScoutSetEventFeedback(zdroj, hodnota) — Procedúra

Vyšle na Scout príkaz SetEventFeedback.

```
HTScoutSetEventFeedback(RCX_ConstantSrc, 10);
```

HTScoutSendVLL(zdroj, hodnota) — Procedúra

Vyšle na Scout príkaz SendVLL.

```
HTScoutSendVLL(RCX_ConstantSrc, 0x30);
```

HTScoutSetScoutRules(motion, touch, light, time, effect) Procedúra

Vyšle na Scout príkaz SetScoutRules.

```
HTScoutSetScoutRules(SCOUT_MR_FORWARD, SCOUT_TR_REVERSE,
      SCOUT_LR_IGNORE, SCOUT_TGS_SHORT, SCOUT_FXR_BUG);
```

HTScoutSetScoutMode(mód) Procedúra

Vyšle na Scout príkaz SetScoutMode.

```
HTScoutSetScoutMode(SCOUT_MODE_POWER);
```

3.15 Volania API pre zariadenia Mindsensors

ReadSensorMSRTClock(port, ss, mm, hh, dow, dd, MM, yy) Procedúra

Prečíta hodnotu presného času zo zariadenia Mindsensors RTClock sensor. Vráti logickú hodnotu, ktorá určuje či operácia prebehla úspešne.

```
ReadSensorMSRTClock(S1, ss, mm, hh, dow, dd, mon, yy);
```

SensorMSCompass(port) Procedúra

Vráti hodnotu z kompasového senzora Mindsensors.

```
x = SensorMSCompass(S1);
```