LibRed API

Table of Contents

1. Obecný úvod . . . 3
2. Sestavení libRed . . . 3
3. Odkazy na hodnoty . . . 3
4. C API . . . 4
   4.1. Správa knihovny . . . 4
      4.1.1. redOpen() . . . 4
      4.1.2. redClose() . . . 4
   4.2. Provoz kódu Red . . . 5
      4.2.1. redDo() . . . 5
      4.2.2. redDoFile() . . . 5
      4.2.3. redDoBlock() . . . 5
      4.2.4. redCall() . . . 6
   4.3. Registrace callbackové funkce . . . 6
      4.3.1. redRoutine() . . . 6
   4.4. Vytváření Redových hodnot z C . . . 7
      4.4.1. redSymbol() . . . 7
      4.4.2. redUnset() . . . 7
      4.4.3. redNone() . . . 7
      4.4.4. redLogic() . . . 7
      4.4.5. redDatatype() . . . 8
      4.4.6. redInteger() . . . 8
      4.4.7. redFloat() . . . 8
      4.4.8. redPair() . . . 8
      4.4.9. redTuple() . . . 8
      4.4.10. redTuple4() . . . 8
      4.4.11. redBinary() . . . 8
      4.4.12. redImage() . . . 9
      4.4.13. redString() . . . 9
      4.4.14. redWord() . . . 9
      4.4.15. redBlock() . . . 9
      4.4.16. redPath() . . . 10
      4.4.17. redLoadPath() . . . 10
      4.4.18. redMakeSeries() . . . 10
   4.5. Vytvoření Céčkových hodnot z Redu . . . 10
      4.5.1. redCInt32() . . . 11
      4.5.2. redCDouble() . . . 11

4.5.3. redCString() . . . 11
4.5.4. redTypeOf() . . . 11
4.6. Volání akce v Redu . . . 11
4.6.1. redAppend() . . . 11
4.6.2. redChange() . . . 11
4.6.3. redClear() . . . 12
4.6.4. redCopy() . . . 12
4.6.5. redFind() . . . 12
4.6.6. redIndex() . . . 12
4.6.7. redLength() . . . 12
4.6.8. redMake() . . . 12
4.6.9. redMold() . . . 13
4.6.10. redPick() . . . 13
4.6.11. redPoke() . . . 13
4.6.12. redPut() . . . 13
4.6.13. redRemove() . . . 13
4.6.14. redSelect() . . . 13
4.6.15. redSkip() . . . 13
4.6.16. redTo() . . . 14
4.7. Přístup ke slovu Redu . . . 14
4.7.1. redSet() . . . 14
4.7.2. redGet() . . . 14
4.8. Přístup k cestě Redu . . . 14
4.8.1. redSetPath() . . . 14
4.8.2. redGetPath() . . . 14
4.9. Přístup k poli objektu . . . 15
4.9.1. redSetField() . . . 15
4.9.2. redGetField() . . . 15
4.10. Ladění . . . 15
4.10.1. redPrint() . . . 15
4.10.2. redProbe() . . . 15
4.10.3. redHasError() . . . 15
4.10.4. redFormError() . . . 16
4.10.5. redOpenLogWindow() . . . 16
4.10.6. redCloseLogWindow() . . . 16
4.10.7. redOpenLogFile() . . . 16
4.10.8. redCloseLogFile() . . . 16
4.11. Definice datových typů . . . 17
5. API pro Visual Basic . . . 17
5.1. Sestavení . . . 17
5.2. redLogic() . . . 18

5.3. redBlockVB() . . . 18
5.4. redPathVB() . . . 18
5.5. redCallVB() . . . 18
5.6. Registrace callbackové funkce . . . 19

1. Obecný úvod

LibRed je specielní verze knihovny 'runtime' a interpreta Redu , vhodná pro integraci se softwérem, vytvořeným s použitím jiného programovacího jazyka než Red. Aby se umožnila interakce jiného (non-Red) jazyka s Redem, poskytuje libRed dedikované low-level API (vyhovující standardům C, cdecl, či MS stdcall), jež je popsané v tomto dokumentu. Mezi podporovanými vlastnostmi jsou:

- Schopnost set/get hodnotu slova v globálním nebo lokálním kontextu.
- Zkrácené konstruktory pro většinu obvyklých datových typů Redu.
- Konverzní funkce pro datové typy Redu, kompatibilní s hostitelským jazykem (většinou C).
- Manipulaci s řadami z hostitelského jazyka.
- Callbacky, umožňující Redu volat funkce hostitelského jazyk.
- Ladící funkce, orientované na konzolu (terminál).

Terminologie: termín *hostitelský* označuje hostitelský jazyk nebo aplikaci s libRed.

Příklady použití libRed lze nalézt zde.

2. Sestavení libRed

Sestavení lokální verze libRed je jednoduché:

```
red build libRed
```

nebo z konzoly Rebolu či zdrojových kódů Redu:

```
rc "build libRed"
```

Tyto příkazy vytvoří verzi libRed pro jazyk C (s použitím `cdecl` ABI). Potřebujete-li `stdcall` ABI (kvůli kompatibilitě s aplikacemi Microsoft), musíte použít:

```
red build libRed stdcall
```

3. Odkazy na hodnoty

Hodnoty Redu mohou být vráceny voláním funkcí libRed. Jsou reprezentovány jako *opaktní* 32-

bitové reference. Tyto reference mají krátkou životnost, jsou tedy vhodné pouze pro lokální použití, jako zadání oné reference jinému volání funkce libRed. Přiřazení takové reference hostitelské proměnné je možné ale mělo be se provést **bezprostředně poté**. Tyto reference (odkazy) používají specifického správce paměti, jenž je podrží při životě pro asi dalších 50 volání API. Na příklad:

```
long a, blk;

a = redSymbol("a");
redSet(a, redBlock(0));                    // returned reference is used immediatly
here

blk = redGet(a);
redPrint(blk);                             // safe reference usage

for(i = 0; i < 100, i++) {
    // redAppend(blk, redNone());     // unsafe reference usage!
    redAppend(redGet("a"), redNone());    // safe version
}
```

4. C API

C API lze použít pro aplikace C/C++ ale také pro integraci Redu do libovolného jiného programovacího jazyka, majícího FFI, kompatibilní s C.

4.1. Správa knihovny

Aby bylo možné použít funkci z API, je nutné vytvořit *instanci* libRed.

NOTE	Aktuálně je povolena pouze jedna seance libRed v jednom procesu. Toto bude v budoucnosti rozšířeno na podporu více instancí.

4.1.1. redOpen()

```
void redOpen(void)
```

Inicializuje novou seanci knihovny Red runtime. Tato funkce musí být volána před voláním jakékoliv jiné funkce API. Je bezpečné, volat ji v témže procesu několikrát; tak jako tak se otevře pouze jedna seance.

NOTE	Je-li před `redOpen` volána jiná funkce, je vratnou hodnotou `-2` indikován ilegální pokus o přístup.

4.1.2. redClose()

```
void redClose(void)
```

Ukončí stávající seanci knihovny Red runtime, uvolňujíc všechny alokované zdroje.

4.2. Provoz kódu Red

Hostující software může spouštět kód Redu přímo s použitím různé úrovně kontroly, od poskytnutí kódu k vyhodnocení v textové formě až k volání libovolné funkce Redu přímo, poskytujíc argumenty vytvořené na straně hostitele.

4.2.1. redDo()

```
red_value redDo(const char* source)
```

Vyhodnotí výraz Redu, zadaný jako řetězec a vrací poslední hodnotu Redu.

Příklady

```
redDo("a: 123");

redDo("view [text {hello}]");

char *s = (char *) malloc(100);
const char *caption = "Hello";
redDo(sprintf(s, "view [text \"%s\"]", caption));
```

4.2.2. redDoFile()

```
red_value redDoFile(const char* filename)
```

Načte a vyhodnotí skript Redu, uvedený jako *filename* a vrátí poslední hodnotu. Název *filename* je formátován s použitím konvencí Redu, nezávislých na OS (basically Unix-style).

Příklady

```
redDoFile("hello.red");
redDoFile("/c/dev/red/demo.red");
```

4.2.3. redDoBlock()

```
red_value redDoBlock(red_block code)
```

Vyhodnotí blok argumentů a vrátí poslední hodnotu Redu.

Příklad

```
redDoBlock(redBlock(redWord("print"), redInteger(42)));
```

4.2.4. redCall()

```
red_value redCall(red_word name, ..., red_integer 0)
```

Invokuje funkci Redu (typu `any-function!`), uvedené názvem red_word, poskytujíc ji potřebné argumenty (jako hodnoty Redu). Vrací poslední hodnotu funkce. Seznam argumentů **musí** končit hodnotou `null` nebo `0` jako označení konce.

Příklad

```
redCall(redWord("random"), redInteger(6));     // returns a random integer! value
between 1 and 6
```

4.3. Registrace callbackové funkce

Reagování na událost, která se vyskytla v Redu nebo přesměrování některých volání na stranu hostitele (jako je přesměrování `print` nebo `ask`), vyžaduje možnost volat zpět hostitelskou funkci ze strany Redu. To umožňuje použití funkce `redRoutine()`.

4.3.1. redRoutine()

```
red_value redRoutine(red_word name, const char* spec, void* func_ptr)
```

Definuje novou rutinu, zvanou *name*, s blokem specifikací *spec* a s tělem, jímž je pointer (ukazovátko) na funkci C. Funkce C **musí** vracet hodnotu Redu (lze použít `redUnset()` k indikaci toho, že návratová hodnota není použita).

Příklad

```
#include "red.h"
#include <stdio.h>

red_integer add(red_integer a, red_integer b) {
    return redInteger(redCInt32(a) + redCInt32(b));
}

int main(void) {
    redRoutine(redWord("c-add"), "[a [integer!] b [integer!]]", (void*) &add);
```

```
    printf(redCInt32(redDo("c-add 2 3")));
    return 0;
}
```

4.4. Vytváření Redových hodnot z C

Mnohé funkce z libRed API vyžadují zadání Redových hodnot (as *references*). Následující funkce jsou jednoduchými konstruktory pro nejpoužívanější datové typy.

4.4.1. redSymbol()

```
long redSymbol(const char* word)
```

Vrací symbol ID spojený s načtemým *word* (poskytnutým jako řetězec v C). Toto ID může být potom zadáno jiným funkcím z libRed API functions, vyžadujícím ID místo hodnoty word.

Příklad

```
long a = redSymbol("a");
redSet(a, redInteger(42));
printf("%l\n", redGet(a));
```

4.4.2. redUnset()

```
red_unset redUnset(void)
```

Vrací hodnotu `unset!`.

4.4.3. redNone()

```
red_none redNone(void)
```

Vrací hodnotu `none!`.

4.4.4. redLogic()

```
red_logic redLogic(long logic)
```

Vrací hodnotu `logic!`. Logická hodnota `0` dává hodnotu `false`, všechny ostatní hodnoty dávají `true`.

4.4.5. redDatatype()

```
red_datatype redDatatype(long type)
```

Vrací hodnotu `datatype!`, korespondující *typu* ID, což je hodnota z výčtu (enumerace) `RedType`.

4.4.6. redInteger()

```
red_integer redInteger(long number)
```

Vrací hodnotu `integer!` z *number*.

4.4.7. redFloat()

```
red_float redFloat(double number)
```

Vrací hodnotu `float!` z *number*.

4.4.8. redPair()

```
red_pair redPair(long x, long y)
```

Vrací hodnotu `pair!` ze dvou celočíselných hodnot.

4.4.9. redTuple()

```
red_tuple redTuple(long r, long g, long b)
```

Vrací hodnotu `tuple!` ze tří celočíselných hodnot (obvykle pro prezentaci barev RGB). Zadané argumenty jsou zkráceny na 8-bitové hodnoty.

4.4.10. redTuple4()

```
red_tuple redTuple4(long r, long g, long b, long a)
```

Vrací hodnotu `tuple!` ze čtyř celočíselných hodot (obvykle pro prezentaci barev RGBA). Zadané argumenty jsou zkráceny na 8-bitové hodnoty.

4.4.11. redBinary()

```
red_binary redBinary(const char* buffer, long bytes)
```

Vrací hodnotu ve formátu `binary!`, vytvořenou z pointeru na vyrovnávací paměť (buffer). Zadaný buffer je kopírován interně.

4.4.12. redImage()

```
red_image redImage(long width, long height, const void* buffer, long format)
```

Vrací hodnotu ve formátu `image!`, vytvořenou z pointeru na vyrovnávací paměť. Velikost obrázku je určena `délkou` a `šířkou` v pixelech. Zadaný buffer je kopírován interně. Akceptované formáty bufferu jsou:

- `RED_IMAGE_FORMAT_RGB`: 24-bit per pixel.
- `RED_IMAGE_FORMAT_ARGB`: 32-bit per pixel, alpha channel leading.

4.4.13. redString()

```
red_string redString(const char* string)
```

Vrací hodnotu `string!` z ukazovátka (pointer) *string*. Očekávané kódování řetězcového argumentu je UTF-8. Jiné kódování lze definovat funkcí `redSetEncoding()`.

4.4.14. redWord()

```
red_word redWord(const char* word)
```

Vrací hodnotu `word!` z řetězce v C. Očekávané kódování řetězcového argumentu je UTF-8. Jiné kódování lze definovat funkcí `redSetEncoding()`. Řetězce, které nemohou být načteny jako slova, vracejí hodnotu `error!`.

4.4.15. redBlock()

```
red_block redBlock(red_value v,...)
```

Vrací novou řadu `block!`, vytvořenou ze seznamu argumentů. Seznam **musí** končit hodnotou `null` nebo `0`, jako označení konce.

Examples

```
redBlock(0);                                // Creates an empty block
redBlock(redInteger(42), redWord("hi"), 0);   // Creates [42 hi] block
```

4.4.16. redPath()

```
red_path redPath(red_value v, ...)
```

Vrací novou řadu `path!`, vytvořenou ze seznamu argumentů. Seznam **musí** končit hodnotou `null` nebo `0`, jako označení konce.

Příklad

```
redDo("a: [b 123]");
long res = redDo(redPath(redWord("a"), redWord("b"), 0);
printf("%l\n", redCInt32(res));    // will output 123
```

4.4.17. redLoadPath()

```
red_path redLoadPath(const char* path)
```

Vrací řadu `path!`, vytvořenou z cesty, vyjádřené jako řetězed v C. To poskytuje rychlý způsob sestavení cest bez jednotlivého vytváření každého elementu.

Příklad

```
redDo(redLoadPath("a/b"));    // Creates and evaluates the a/b path! value.
```

4.4.18. redMakeSeries()

```
red_value redMakeSeries(unsigned long type, unsigned long slots)
```

Vrací nový objekt `series!` typu *type* s dostatkem místa pro uložení *slotových* prvků. Toto je generická funkce pro tvorbu řad. Typ prvků musí být jedním z výčtových hodnot `RedType`.

Examples

```
redMakeSeries(RED_TYPE_PAREN, 2);  // Creates a paren! series

long path = redMakeSeries(RED_TYPE_SET_PATH, 2); // Creates a set-path!
redAppend(path, redWord("a"));
redAppend(path, redInteger(2));    // Now path is `a/2:`
```

4.5. Vytvoření Céčkových hodnot z Redu

Konverze Redových hodnot na hodnoty *hostitele* je možná, byť s omezením menšího počtu typů v

jazyce C.

4.5.1. redCInt32()

```
long redCInt32(red_integer number)
```

Vrací 32-bitové signované celé číslo z hodnoty `integer!` v Redu.

4.5.2. redCDouble()

```
double redCDouble(red_float number)
```

Vrací 'C_double_floating_point_value' z hodnoty `float!` v Redu.

4.5.3. redCString()

```
const char* redCString(red_string string)
```

Vrací 'UTF-8_string_buffer_pointer' z hodnoty `string!` v Redu. Jiná kódování lze definovat funkcí `redSetEncoding()`.

4.5.4. redTypeOf()

```
long redTypeOf(red_value value)
```

Vrací ID typu z hodnoty v Red. Hodnoty ID typů jsou definovány ve výčtu `RedType`. Viz Datatypes section.

4.6. Volání akce v Redu

Je možné volat libovolnou funkci Redu s použitím `redCall`. Pro většinu obvyklých akcí však je možné použít nejaké zkratky, poskytované pro větší pohodlí a lepší výkonost.

4.6.1. redAppend()

```
red_value redAppend(red_series series, red_value value)
```

Připojí *hodnotu* k *řadě* a vrací celou řadu (s ukazovátkem v čele).

4.6.2. redChange()

```
red_value redChange(red_series series, red_value value)
```

Změní *hodnotu* in *řadě* a vrací zbytek řady za změnou.

4.6.3. redClear()

```
red_value redClear(red_series series)
```

Přemístí hodnoty *řady* z aktuálního indexu na chvost (tail) a vrací řadu s novým chvostem.

4.6.4. redCopy()

```
red_value redCopy(red_value value)
```

Vrací kopii neskalární hodnoty.

4.6.5. redFind()

```
red_value redFind(red_series series, red_value value)
```

Vrací *řadu* od místa, kde byla nalezena *hodnota* nebo `none`.

4.6.6. redIndex()

```
red_value redIndex(red_series series)
```

Vrací aktuální index *řady* relativně k čelu nebo slovo v kontextu.

4.6.7. redLength()

```
red_value redLength(red_series series)
```

Vrací počet hodnot v *řadě*, od aktuálního indexu po chvost.

4.6.8. redMake()

```
red_value redMake(red_value proto, red_value spec)
```

Vrací novou hodnotu, vytvořenou ze *spec* konformní s typem *proto*.

4.6.9. redMold()

```
red_value redMold(red_value value)
```

Vrací prezentaci hodnoty jako formátovací řetězec zdroje.

4.6.10. redPick()

```
red_value redPick(red_series series, red_value value)
```

Vrací *řadu* v hodnotě daného indexu.

4.6.11. redPoke()

```
red_value redPoke(red_series series, red_value index, red_value value)
```

Nahradí indexem označenou hodnotu *řady* novou hodnotou a vrátí novou hodnotu.

4.6.12. redPut()

```
red_value redPut(red_series series, red_value index, red_value value)
```

Nahradí hodnotu za klíčem v *řadě* nebo v hodnotě `map!` a vrátí novou hodnotu.

4.6.13. redRemove()

```
red_value redRemove(red_series series)
```

Odebere hodnotu aktuálního indexu *řady* index a vrátí upravenou řadu.

4.6.14. redSelect()

```
red_value redSelect(red_series series, red_value value)
```

Nalezne *hodnotuy* v *řadě* a vrátí následující hodnotu nebo `none`.

4.6.15. redSkip()

```
red_value redSkip(red_series series, red_integer offset)
```

Vrací *řadu* relativně k aktuálnímu indexu.

4.6.16. redTo()

```
red_value redTo(red_value proto, red_value spec)
```

Konvertuje hodnotu *spec* na datový typ, specifikovaný v *proto*.

4.7. Přístup ke slovu Redu

Zadání nebo získání hodnoty slova v Red je nejpřímějším způsobem předání hodnoty mezi *hostitelským* a Redovým běhovým prostředím.

4.7.1. redSet()

```
red_value redSet(long id, red_value value)
```

Zadá *hodnotě* slovo, definované ze symbolu *id*. Ze symbolu vytvořené slovo má globální kontext. Vrací hodnotu *value*.

4.7.2. redGet()

```
red_value redGet(long id)
```

Vrací hodnotu slova, definovaného ze symbolu *id*. Slovo, vytvořené ze symbolu má globální kontext.

4.8. Přístup k cestě Redu

Cesty jsou velmi flexibilním způsobem přístupu k datům v Redu, takže mají své dedikované přístupové funkce v libRed. Především umožňují přístup ke slovům v objektových kontextech.

4.8.1. redSetPath()

```
red_value redSetPath(red_path path, red_value value)
```

Přiřadí *path* k *value* a vrací tuto *value*.

4.8.2. redGetPath()

```
red_value redGetPath(red_path path)
```

Vrací *hodnotu* označenou cestou (*path*).

4.9. Přístup k poli objektu

Je-li zapotřebí vícero přístupů `setting/getting` k polím objektu, doporučuje se použít přímo hodnotu objektu místo vytváření cesy. K tomu účelu slouží dvě následující funkce.

NOTE	Tyto accesory akceptují jakýkéli další asociované typy kolektorů (arrays), nikoliv pouze objekty typu `object!`. Zadání objektu typu `map!` je tedy také dovoleno.

4.9.1. redSetField()

```
red_value redSetField(red_value object, long field, red_value value)
```

Nastaví *pole objektu* na *hodnotu* a vrátí tuto *hodnotu*. Argument *pole* je ID symbol, vytvořený s použitím `redSymbol()`.

4.9.2. redGetField()

```
red_value redGetField(red_value obj, long field)
```

Vrátí *hodnotu* uloženou v *poli objektu*. Argument *pole* je ID symbol, vytvořený s použitím `redSymbol()`.

4.10. Ladění

Několik šikovných ladících funkcí se rovněž nabízí. Většina z nich vyžaduje okno systémové konzoly i když je možné vyvolat otevření logovacího okna nebo přesměrování výstupu do souboru.

4.10.1. redPrint()

```
void redPrint(red_value value)
```

Tiskne *value* do standardního výstupu nebo do ladící konzoly, je-li otevřena.

4.10.2. redProbe()

```
red_value redProbe(red_value value)
```

Přenese (probes) *value* do standardního výstupu, nebo do ladící konzoly, je-li otevřena. Volání této funkce vrací *value*.

4.10.3. redHasError()

```
red_value redHasError(void)
```

Vrací hodnotu `error!`, vyskytla-li se chyba v předchozím volání API nebo `null`, pakliže se žádná chyba nevyskytla.

4.10.4. redFormError()

```
const char* redFormError(void)
```

Vrací ukazovátko (pointer) řetězce v UTF-8, obsahující formátovanou chybu, pokud k ní došlo, případně `null`, pokud se žádná chyba nevyskytla.

4.10.5. redOpenLogWindow()

```
int redOpenLogWindow(void)
```

Otevře logovací okno a přesměruje do něho všechen tiskový výstup Redu. Tento nástroj je užitečný, není-li hostitelská aplikace spouštěna ze systémové konzoly, která se implicitně používá pro tiskový výstup. Opakované volání této funkce je bez účinku, je-li již logovací okno otevřeno. Vrací `1` při úspěchu, `0` při selhání.

NOTE Pouze pro platformy Windows.

4.10.6. redCloseLogWindow()

```
int redCloseLogWindow(void)
```

Zavírá logovací okno. Volání této funkce když je logovací okno již zavřené nemá žádný účinek. Vrací `1` při úspěchu, `0` při selhání.

NOTE Pouze pro platformy Windows.

4.10.7. redOpenLogFile()

```
void redOpenLogFile(const string *name)
```

Přesměruje výstup z tiskových funkcí Redu do souboru *name*. Součástí jména může být relativní nebo absolutní cesta při použití formátu, specifického pro OS.

4.10.8. redCloseLogFile()

```
void redCloseLogFile(void)
```

Closes the log file opened with `redOpenLogFile()`.

NOTE	V současné době **musí** být logovací soubor při exitu zavřen, jinak je nad ním držen zámek (lock) a to může způsobit zamrznutí nebo kolaps v některých hostitelských (např. MS Office) aplikacích.

4.11. Definice datových typů

Některé funkce z libRed API se mohou odkazovat na datové typy Redu: `redTypeOf()`, `redMakeSeries()` a `redDatatype()`. Tyto typy jsou na hostitelské straně prezentovány jako výčet (`RedType`), kde typy jsou zastoupeny jmény podle následujícího schematu:

```
RED_TYPE_<DATATYPE>
```

Vyčerpávající seznam je k nahlédnutí zde.

5. API pro Visual Basic

API Visual Basicu lze použít jak pro VB tak VBA (z aplikací MS Office). Je v podstatě stejné jako API pro C, takž v následujících odstavcích budou popsány pouze rozdíly. Rozdíly jsou většinou ve varidických funkcích, jež jsou rozděleny do dvou skupin:

- `redBlock()`, `redPath()`, `redCall()` přijímají pouze hodnoty Red a nevyžadují terminální `null` nebo `0`, jako u verze C.
- `redBlockVB()`, `redPathVB()`, `redCallVB()` přijímají pouze hodnoty VB, které jsou automaticky konvertovány podle následující tabulky:

VisualBasic	Red
`vbInteger`	`integer!`
`vbLong`	`integer!`
`vbSingle`	`float!`
`vbDouble`	`float!`
`vbString`	`string!`

5.1. Sestavení

K použití libRed s VB/VBA, potřebujete binární verzi libRed, která je kompilována pro `stdcall` ABI. Potřebujete-li takovou verze rekompilovat:

```
red build libRed stdcall
```

Potřebujete také do svého projektu importovat modulový soubor `libRed.bas`.

5.2. redLogic()

```
Function redLogic(bool As Boolean) As Long
```

Vrací Redovou hodnotu `logic!`, vytvořenou z VB hodnoty `boolean`.

5.3. redBlockVB()

```
Function redBlockVB(ParamArray args() As Variant) As Long
```

Vrací novou řadu typu `block!`, vytvořenou ze seznamu argumentů. Počet argumentů je proměnný je je složen pouze z hodnot VisualBasic.

Příklady

```
redProbe redBlockVB()              ' Creates an empty block
redProbe redBlockVB(42, "hello")   ' Creates the [42 "hello" hi] block
```

5.4. redPathVB()

```
Function redPathVB(ParamArray args() As Variant) As Long
```

Vrací novou řadu typu `path!`, vytvořenou ze seznamu argumentů. Počet argumentů je proměnný je je složen pouze z hodnot VisualBasic.

Příklady

```
redDo("a: [b 123]")
res = redDo(redPathVB("a", "b"))
Debug.print redCInt32(res))        ' will output 123
```

5.5. redCallVB()

```
Function redCallVB(ParamArray args() As Variant) As Long
```

Invokuje funkci Redu (typu `any-function!`), uvedenou zadaným řetězcem (první argument), doplňujíc ji případně dalšími argumenty (jako hodnoty VisualBasic). Vrací poslední hodnotu funkce. Počet argumentů je proměnný a je složen pouze z hodnot VisualBasic.

Příklad

```
redCallVB("random", 6);           ' returns a random integer! value between 1 and 6
```

5.6. Registrace callbackové funkce

Vytvoření funkce pro VisualBasic, která může být volána ze strany Redu, se provádí jako v C API, s použitím volání `redRoutine()`. Posledním argumentem pro tuto funkci je ukazovátko (pointer) funkce. Ve VB je takové ukazovátko možné získat pouze pro funkci, definované v *modulu* ale ne v *UserForm*.

Toto je callback použitý v demo Excelu "Red Console":

```
Sub RegisterConsoleCB()
    redRoutine redWord("print"), "[msg [string!]]", AddressOf onConsolePrint
End Sub

Function onConsolePrint(ByVal msg As Long) As Long
    If redTypeOf(msg) <> red_unset Then Sheet2.AppendOutput redCString(msg)
    onConsolePrint = redUnset
End Function
```