LibRed API

目次

1. 概要 . . . 3
2. libRedのビルド . . . 3
3. 値の参照 . . . 3
4. C API . . . 4
    4.1. ライブラリの管理 . . . 4
        4.1.1. redOpen() . . . 4
        4.1.2. redClose() . . . 4
    4.2. Redのコードの実行 . . . 4
        4.2.1. redDo() . . . 5
        4.2.2. redDoFile() . . . 5
        4.2.3. redDoBlock() . . . 5
        4.2.4. redCall() . . . 5
    4.3. コールバック関数の登録 . . . 6
        4.3.1. redRoutine() . . . 6
    4.4. CからRedの値を作る . . . 6
        4.4.1. redSymbol() . . . 6
        4.4.2. redUnset() . . . 7
        4.4.3. redNone() . . . 7
        4.4.4. redLogic() . . . 7
        4.4.5. redDatatype() . . . 7
        4.4.6. redInteger() . . . 7
        4.4.7. redFloat() . . . 7
        4.4.8. redPair() . . . 8
        4.4.9. redTuple() . . . 8
        4.4.10. redTuple4() . . . 8
        4.4.11. redBinary() . . . 8
        4.4.12. redImage() . . . 8
        4.4.13. redString() . . . 8
        4.4.14. redWord() . . . 9
        4.4.15. redBlock() . . . 9
        4.4.16. redPath() . . . 9
        4.4.17. redLoadPath() . . . 9
        4.4.18. redMakeSeries() . . . 10
    4.5. Cの値をRedから作成する . . . 10
        4.5.1. redCInt32() . . . 10
        4.5.2. redCDouble() . . . 10

4.5.3. redCString() . . . 10
4.5.4. redTypeOf() . . . 10
4.6. Redのactionの呼び出し . . . 11
4.6.1. redAppend() . . . 11
4.6.2. redChange() . . . 11
4.6.3. redClear() . . . 11
4.6.4. redCopy() . . . 11
4.6.5. redFind() . . . 11
4.6.6. redIndex() . . . 11
4.6.7. redLength() . . . 12
4.6.8. redMake() . . . 12
4.6.9. redMold() . . . 12
4.6.10. redPick() . . . 12
4.6.11. redPoke() . . . 12
4.6.12. redPut() . . . 12
4.6.13. redRemove() . . . 12
4.6.14. redSelect() . . . 13
4.6.15. redSkip() . . . 13
4.6.16. redTo() . . . 13
4.7. Redのwordへのアクセス . . . 13
4.7.1. redSet() . . . 13
4.7.2. redGet() . . . 13
4.8. Redのパスへのアクセス . . . 13
4.8.1. redSetPath() . . . 14
4.8.2. redGetPath() . . . 14
4.9. Redのオブジェクトフィールドへのアクセス . . . 14
4.9.1. redSetField() . . . 14
4.9.2. redGetField() . . . 14
4.10. デバッギング . . . 14
4.10.1. redPrint() . . . 14
4.10.2. redProbe() . . . 15
4.10.3. redHasError() . . . 15
4.10.4. redFormError() . . . 15
4.10.5. redOpenLogWindow() . . . 15
4.10.6. redCloseLogWindow() . . . 15
4.10.7. redOpenLogFile() . . . 15
4.10.8. redCloseLogFile() . . . 16
4.11. データ型の定義 . . . 16
5. Visual Basic API . . . 16
5.1. セットアップ . . . 16
5.2. redLogic() . . . 17

5.3. redBlockVB() 17
5.4. redPathVB() 17
5.5. redCallVB() 17
5.6. コールバック関数の登録 17

1. 概要

LibRedはRedのインタプリタとランタイムライブラリの特別なバージョンです。libRedはRed以外の言語でソフトウェアを開発する場合に適しています。Redではないソフトウェアが Redとやりとりすることを可能にするために、libRedはこのドキュメントに記載されている専用のローレベルAPIを（CのcdeclかMicrosoftのstadcallで）公開します。サポートされる機能は以下の通りです。

- グローバルまたはローカルなコンテキストのwordへの値の設定と取得
- 最も一般的なRedのデータ型のコンストラクタへのショートカット
- Cと互換のあるRedのデータ型に関する関数のコンバージョン
- ホスト言語からのseriesの操作
- Redがホスト言語の関数を呼び出し可能にするコールバック機能
- コンソール向けのデバッグ関数

用語：ホスト という用語はホスト言語またはlibRedを組み込んだアプリケーションのことを指します。

libRedを使用したサンプルは こちら で確認することができます。

2. libRedのビルド

自身のローカル環境にlibRedをビルドする方法はシンプルです。

```
red build libRed
```

または、RebolコンソールとRedのソースで

```
rc "build libRed"
```

これらのコマンドラインはC言語用（cdeclのABIを使用する）libRedをビルドします。もし（Microsoftアプリケーションとの互換のために）stdcall ABIが必要の場合、次のようにします。

```
red build libRed stdcall
```

3. 値の参照

libRedの関数呼び出しにより、Redの値が返されます。値は不透明な32ビットの参照で表現されます。これらの参照は短命で、別のlibRed関数の呼び出しに渡すなど、ローカルな使用に留めるべきです。ホストの変数にこれらの参照をセットすることは可能ですが、

□□□□□□□□□ □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□50□□API□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

```
long a, blk;

a = redSymbol("a");
redSet(a, redBlock(0));                     // □□□□□□□□□□□□□□□□□□□□□□□□

blk = redGet(a);
redPrint(blk);                              // □□□□□□□□

for(i = 0; i < 100, i++) {
    // redAppend(blk, redNone());           // □□□□□□□□□□□□□□
    redAppend(redGet("a"), redNone());      // □□□□□□□□
}
```

4. C API

C API□C/C++□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□C□□ FFI □□□□□□□□□□Red□□□□□□□□□□□□□□

4.1. □□□□□□□□□

libRed□ □□□□□□□ □□□□□□API□□□□□□□□□□□□□□

NOTE □□□□□□□□□□□□□□□□□□□□□libRed□□□

4.1.1. redOpen()

```
void redOpen(void)
```

□□□Red□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ □□□□□□*API*□□□□□□□□□ □□□□□□□□□□□□□□□□

NOTE □□redOpen□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□-2□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

4.1.2. redClose()

```
void redClose(void)
```

□□□Red□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

4.2. Red□□□□□□□□□

□□□□□□□□□□Red□□Red□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□Red□□□□□□□□□□□□□□□□□□□□□□□□□□□

4.2.1. redDo()

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

□□□□□□□□□□□Red□□□□□□□□□□□Red□□□□□□□□□□□□□□□

□

```
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)
```

filename □□□□□□Red□□□□□□□□□□□□□□□□□□□□□□□□□□□ *filename* □Red□OS□□□□□□□□□□□□□Unix□□□□□□□□□□□□□□□□□□□□□□□□□

□

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

4.2.3. redDoBlock()

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

□□□□□□□□□□□□□□□□□□□□□□□□

□

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

4.2.4. redCall()

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

name word□□□□□□Red□□□□any-function!型）を実行します。任意の数の（Redの値の）引数を渡すことができ、関数の最後の値を返します。引数の終わりを示すため、「null」か「0」で 終わらなければなりません。

例

```
redCall(redWord("random"), redInteger(6));     // 1から6の間のランダムなinteger!の値を返します。
```

4.3. コールバック関数の登録

Redで発生したイベントにレスポンスしたり、Redの呼び出しを（たとえば「print」や「ask」を）リダイレクトしたりするには、Redの側からホストの関数へコールバックする必要があります。これは「redRoutine()」関数を使うことで実現できます。

4.3.1. redRoutine()

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

name で指定された名前と、 *spec* をスペックブロック、 *func-ptr* で指定されたCの関数ポインタをボディとして、新しいRedのroutineを定義します。指定するCの関数は**Red**の値を返さなければなりません（戻り値が使われないことを示すために「redUnset()」を使うこともできます）。

例

```
#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. CからRedの値を作る

libRed APIの多くの関数は（ *references* として）Redの値を渡す必要があります。以下の関数は最もよく使われるデータ型のシンプルなコンストラクタです。

4.4.1. redSymbol()

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

（Cのstringとして与えられた）ロードされた *word* に関連づけられたシンボルIDを返します。このIDはwordの値のの代わりにシンボルIDを必要とする他のlibRed API関数に渡すことができます。

□

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

4.4.2. redUnset()

```
red_unset redUnset(void)
```

□ *unset!* □□□□□□

4.4.3. redNone()

```
red_none redNone(void)
```

□ *none!* □□□□□□

4.4.4. redLogic()

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

□logic!□□□□□□□□□□ *logic* □□□□0□□□□□□false□□□□□□□□□□□true□□□□□□□□□□□

4.4.5. redDatatype()

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

type □□□□□□ID□□□□□□datatype!□□□□□□□□□□ID□□RedType□□□□□□□□□□□□□□

4.4.6. redInteger()

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

number □□□□□□□□□□□integer!□□□□□□□□□

4.4.7. redFloat()

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

number □□□□□□□□□□□□float!□□□□□□□□□

4.4.8. redPair()

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

□□□integer□□□□pair!□□□□□□□□□

4.4.9. redTuple()

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

□□□integer□□□□□RGB□□□□□□□□□□□□□□□□□□□□□tuple!□□□□□□□□□□□□□□□□8□□□□□□□□□□□□□□

4.4.10. redTuple4()

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

□□□integer□□□□□RGB□□□□□□□□□□□□□□□□□□□□□tuple!□□□□□□□□□□□□□□□□8□□□□□□□□□□□□□□

4.4.11. redBinary()

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

□□□□□□□□□□□□□□□□□□□□bytes□□□□□□□□□□□□□□□□ `binary!` □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

4.4.12. redImage()

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

□□□□□□□□□□□□□□□□□□□□□ `image!` □□□□□□□□□□□□□□□□□□□□□□□□□□ `width` □ `height` □□

- `RED_IMAGE_FORMAT_RGB`: 24BPP□24-bit per pixel□□□□□□□□
- `RED_IMAGE_FORMAT_ARGB`: 32BPP□32-bit per pixel□□□□□□□□□□□□□□□□□□□

4.4.13. redString()

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

string □□□□□□□□string!□□□□□□□□□□□□□□□□□□□□□□□□□□□□UTF-8□□□redSetEncoding()□□□□□□□□□□□□□□□

4.4.14. redWord()

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

C□string□□□word!□□□□□□□□□□□□□□□□□□□□□□□□□□□UTF-8□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□redSetEncoding()□□□□□□□□□□□□□□word□□□□□□□□□□□□□□□□□□□□□□□□error!□□□□□□□□

4.4.15. redBlock()

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

□□□□□□□□□□□□block!□series□□□□□□□□□□□□□□□□□□□□□□□□□□□null□□□0□□ □□□□□□□□□□□□□□□□□□□□□□

□

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

4.4.16. redPath()

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

□□□□□□□□□□□□path!□series□□□□□□□□□□□□□□□□□□□□□□□□□□□null□□□0□□ □□□□□□□□□□□□□□□□□□□□□□

□

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

4.4.17. redLoadPath()

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

C□□□□□□□□□□□□□□□□path!□series□□□

□

```
redDo(redLoadPath("a/b"));    // a/b□□□path!□□□□□□□□□□□□□□
```

4.4.18. redMakeSeries()

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

type □□□□□□□□□□□series!□□ *slots* □□□□□□□□□□□□□□□□□□□□□□□□□□□series□□□□□□□□□□□□□□□□type□ `RedType` □□□□□□□□□□□□□□□□□□□□□□□□

□

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

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

4.5. C□□□Red□□□□□□□

Red□□□□□□□□□□□□□□□□□□□□□□□□□C□□□□□□□□□□□□□□□□□□□□□□

4.5.1. redCInt32()

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

Red□integer!□□□□32□□□□□□□□□□□□□□□□□

4.5.2. redCDouble()

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

Red□float!□□□□C□□□□□□□□□□□□□□□□□□□□

4.5.3. redCString()

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

Red□string!□□□□UTF-8□□□□□□□□□□□□□□□□□□□□□□□□□□□□ `redSetEncoding` □□□□□□□□□□

4.5.4. redTypeOf()

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

Red□□□□ID□□□□□□□ID□ `RedType` □□□□□□□□□□□□□□□□□□□□□ □□□□□□□□□□□□□□□□□□

4.6. Red□action□□□□□

`redCall`
□□□□□□□□□□□□Red□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□action□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

4.6.1. redAppend()

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

value □ *series* □□□□□□□□□□□□□□□□series□□□□□□

4.6.2. redChange()

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

series □□□ *value* □□□□□□□□□□□□□□series□□□□□□

4.6.3. redClear()

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

series □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□series□□□□□□

4.6.4. redCopy()

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

□□□□□□□□□□□□□□□□

4.6.5. redFind()

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

value □□□□□□□□□ *series* □□□□□NONE□□□□□□

4.6.6. redIndex()

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

□□□□□□□_series_ □□□□□□□□□□□□□□□□□□word□□□□□□□□□□□□□

4.6.7. redLength()

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

□□□□□□□□□□□□ *series* □□□□□□□□□□□□□

4.6.8. redMake()

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

spec □ *proto* □□□□□□□□□□□□□□□□□□□□

4.6.9. redMold()

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

□□Red□□□□□□□□□□□□□□□□□□□□□□□

4.6.10. redPick()

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

series □□□□□□□□□□□□□ *value* □□□□□□

4.6.11. redPoke()

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

series □□□□□□□□□□□□□ *value* □□□□□□□□□□□□□□□□□

4.6.12. redPut()

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

series □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

4.6.13. redRemove()

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

series □□□□□□□□□□□□□□□□□□□□□□series□□□□□□

4.6.14. redSelect()

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

series □□□ *value* □□□□□□□□□□□□□□□□□□□□□□□□□NONE□□□□□□

4.6.15. redSkip()

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

□□□□□□□□□□□□□□□□□□□□□□□□□ *series* □□□□□□□

4.6.16. redTo()

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

spec □□□ *proto* □□□□□□□□□□□□□□□□□□□□

4.7. Red□word□□□□□□

Red□word□□□□□Red□word□□□□□□□□□□□□□□Red□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

4.7.1. redSet()

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

id □□□□□□□□□□□word□ *value* □□□□□□□□□□□□□□□□□□□□□□□word□□□□□□□□□□□□□□□□□□□□□ *value* □□□□□□□□□□□□□□□□□□□□□□

4.7.2. redGet()

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

id □□□□□□□□□□□□word□□□□□□□□□□□□□□□□□□□□□□word□□□□□□□□□□□□□□□□□□□□□□□

4.8. Red□□□□□□□□□

□□□Red□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□libRed□□word□□□□□
□□□□□□□□□□□

4.8.1. redSetPath()

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

path に *value* をセットし、その *value* を返します。

4.8.2. redGetPath()

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

path で参照される *value* を返します。

4.9. Red のオブジェクトフィールドへのアクセス

オブジェクトのフィールドへの複数のセット／ゲットアクセスが必要な場合、パスを構築するよりもオブジェクトの値を直接使用する方がシンプルで木の間いいことがあります。以下の2つの関数はそのようなアクセスを可能にするために用意されています。

NOTE これらのアクセサは `object!` 型だけでなく、関連付けされた他の配列型にも機能します。従って `map!` の値に対して使うことも可能です。

4.9.1. redSetField()

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

object の *field* に *value* をセットし、その *value* を返します。*field* 引数は `redSymbol()` によって作られたシンボルIDです。

4.9.2. redGetField()

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

object の *field* に保持されている *value* を返します。*field* 引数は `redSymbol()` によって作られたシンボルIDです。

4.10. デバッギング

いくつかの便利なデバッグ機能も提供されています。ほとんどは出力のためにシステムシェルウィンドウを必要としますが、ログウィンドウを強制的に開いたり、ファイルに出力をリダイレクトすることもできます。

4.10.1. redPrint()

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

標準出力か、もし開かれている場合はデバッグコンソールに *value* を出力します。

4.10.2. redProbe()

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

□□□□□□□□□□□□□□□□□□□□□ *value* □probe□□□□□□□□ *value* □□□□□□□□□□□□□□□

4.10.3. redHasError()

```
red_value redHasError(void)
```

□□□API□□□□□□□□□□□□□ error! □□□□□□□□□□□□□ `null` □□□□□

4.10.4. redFormError()

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

□□□□□□□□ □□□□□□□□□□□□UTF-8□□□□□□□□□□□□□□□□□□ `null` □□□□□

4.10.5. redOpenLogWindow()

```
int redOpenLogWindow(void)
```

□□□□□□□□□□□Red□print□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□print□□□□□□□□□□□□□□□□□
□□□1□□□□□□□□0□□□

NOTE Windows□□□□□□□□□□□□□

4.10.6. redCloseLogWindow()

```
int redCloseLogWindow(void)
```

□□1□□□□□□□□0□□□

NOTE Windows□□□□□□□□□□□□□

4.10.7. redOpenLogFile()

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

name □□□□□□□□□Red□print□□□□□□□□□ *name* □□OS□□□□□□□□□□□□□□□□□□□□□□□□□□

4.10.8. redCloseLogFile()

```
void redCloseLogFile(void)
```

`redOpenLogFile()` によって開かれたログファイルを閉じます。

NOTE 現在のところ、ログファイルは終了時には必ず閉じておく必要があります。ログファイルを閉じずに終了した場合、ロックが残ってしまい、MS Officeアプリケーションなどの一部のホストではフリーズやクラッシュが起きる可能性があります。

4.11. データ型の定義

libRed APIのいくつかの関数はRedのデータ型を参照します。具体的には `redTypeOf`´ `redMakeSeries()`´`redDataType()`です。Redのデータ型はホストの側で`RedType`列挙体として表現され、次の命名規則で型を示します。

```
RED_TYPE_<DATATYPE>
```

網羅的なリストは ここ で見つかります。

5. Visual Basic API

VBとVBA（MS Office アプリケーション上）のためのVisual Basic APIが用意されています。これは基本的にはC APIと同等で、以下のセクションに記載の内容だけが異なります。差異のほとんどは可変長引数に関係するもので、以下の２つに分類できます。

- `redBlock()`, `redPath()`, `redCall()` はRedの値だけを受け取ることができ、Cのバージョンのように `null` または `0` で終わることを必要としません。
- `redBlockVB()`, `redPathVB()`, `redCallVB()` はVBの値だけを受け取ることができ、次に示すテーブルの内容で自動的に変換が行われます。

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

5.1. セットアップ

VBおよびVBAでlibRedを使用するためには、□`stdcall` のABIでコンパイルされたlibRedのバイナリが必要です。コンパイルするには、以下のコマンドを実行します。

```
red build libRed stdcall
```

また、 `libRed.bas` モジュールをプロジェクトにインポートする必要もあります。

5.2. redLogic()

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

VB□ boolean □□□□ Red□ logic! □□□□□□□□□□□□

5.3. redBlockVB()

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

□□□□□□□□□□□block!□□□□□□□□□□□□□□□□□□□□□VisualBasic□□□□□□□□□□□□□□□□□□□□

□

```
redProbe redBlockVB()                ' □□□□□□□□□□□□□
redProbe redBlockVB(42, "hello")     ' [42 "hello"] □□□□□□□□□□□□□□
```

5.4. redPathVB()

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

□□□□□□□□□□□path!□series□□□□□□□□□□□□□□□□□VisualBasic□□□□□□□□□□□□□□□□□□□□

□

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

5.5. redCallVB()

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

□□□□□□□□□□□□□□□□□□□□ any-function!
□□Red□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□VisualBasic□□□□□□□□□□□□□

□

```
redCallVB("random", 6);              ' 1□□6□□□□□□□□integer!□□□□□□□□
```

5.6. □□□□□□□□□□□□

Red□□□□□□□□□□□□□□□□VisualBasic□□□□□□□□□C API□□□□□□□ redRoutine()

を呼び出します。その関数の最後の引数は関数のポインタです。VBではそのようなポインタは *module* で定義された関数の場合のみ取得でき、 *UserForm* では取得できません。

以下はExcelの「Red Console」デモで使用されているコールバックです。

```
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
```