ソリトン・ランダム・ジェネレータ
ソフトウェア・デベロプメント・キット
プライム

Soliton Random Generator - Software Development Kit Prime

SRG-SDK Prime

マニュアル

-- 対象製品 --

SRG-SDK Prime Free Windows32bit版
SRG-SDK Prime Standard Windows32bit版
SRG-SDK Prime Enterprise Windows32bit版

有限会社電机本舗
http://www.dnki.co.jp/

使用承諾書

[1:使用承諾一般条項]
〇有限会社電机本舗(以下『弊社』といいます)は、本製品の使用者(以下『使用者』といいます)に対してこのパッケージに入っている製品を本書に従い使用することを承諾します。製品とは、本パッケージに入っているソフトウェア、プログラムディスク、マニュアルのことです。ただし、サードパーティ製の添付プログラム、マニュアルは製品に含めません。サードパーティ製のプログラムの使用については各メーカの規定に従うものとします。
〇製品に関する著作権およびその他の一切の権利は、本契約で明示的に付与したものを除き全て弊社に帰属します。
〇製品のサポートは購入日から1年間、弊社が実施します。ただし、「SRG-SDK Free 」、サンプル、試用品、オマケとして添付されたものは原則としてサポートの義務を伴わないものとします。

[2:使用制限]
〇使用者は製品を同時に複数のコンピュータで使用できません。また複数の使用者による使用も禁止します。但しマニュアル等において例外的に複数の使用を許可している場合はこの限りではありません。

[3:複製等の禁止]
〇使用者はマニュアルに記載されていない限り製品の全部または一部を複製、解析、改変はできません。
〇使用者は弊社の承諾なしに製品の一部ないし全部を売却、譲渡、貸与のいかなる方法でも第三者に使用できません。
〇使用者の製品の使用のための印刷・コピーは自由ですが、次のコピーライトを必ず各ページに記載してください。その他の転載などについては弊社まで御連絡ください。 コピーライト ”(C)2008, 有限会社電机本舗”

[4:保証]
〇使用者は製品の受け渡しから1ケ月以内に本製品や印刷物の物理的な障害を発見した時は交換を要求できます。

[5:免責]
〇弊社は前条に定める場合を除き製品に関していかなる保証も行いません。
〇弊社は使用者が製品の使用に関して直接または間接に生じる一切の損害(通常損害、特別損害およびその他一切の損害)について責任をおいません。

[6:使用権の消滅]
〇使用者が本承諾書に違反した場合または著作権法その他の命令に違反することによって、製品に関して弊社の著作権およびその他の権利を侵害した場合は[1:使用承諾一般条項]規定の使用権は消滅します。この時はただちに製品を弊社に返還しなければいけません。なお返却のための費用は使用者に負担していただきます。

有限会社電机本舗
〒108-0074 東京都港区高輪1-2-16鈴木ビル6A 電話(03)5449-7057
E-Mail:tec@dnki.co.jp
URL: http://www.dnki.co.jp/

目次

1. SRG-SDK Prime について

1.1 概要と特徴

「SRG-SDK Prime」は超長周期擬似乱数発生装置「ソリトン・ランダム・ジェネレータ(SRG)」により10の27000乗の乱数データを生成するソフトウェア開発用のキットです。※特許出願済。
最大で10の27000乗という数の乱数のシードを扱うことができます。

【SRGの特徴】

超長周期

内部に8Kバイトのメモリを使用し10の27000乗という超長周期を持ち、確保する作業用メモリサイズに合わせてスケーラブルに周期が伸びる特性があります。

超高速

乗算・除算命令というCPU負荷の高い命令を一切使用していないため、8ビットや16ビットCPUにおいても効率を落とさずMIPS値に比例した乱数生成を実現できます。 インテル製CPUセレロン1．7GHzのパソコンにおいて秒速1Gbps（125Mバイト）の乱数発生能力を持ちます。

精度

アメリカ政府が無料配布している乱数表の検定プログラムNIST「SP800－22」において。1Gビットファイルを10サンプル検定し、各項目で合格値を出す成績を上げています。
SRGはBlumBlumShub乱数発生方式と同系統の予測不可能型アルゴリズムを採用しており、一定のパターンが続くと先が予測できるという欠点がありません。暗号用途として高水準高品質の乱数を提供できます。

1.2 ラインナップ

SRG-SDK にはファイル操作用のシリーズと、それ以外の用途に用いる特別バージョンがあります。それぞれ Windwos32bit 版、Linux 版が存在しています。

1. SRG-SDK Prime

乱数の生成に特化したバージョンです。 1Gbpsを越す高速、高品質な乱数発生装置を提供します。 各種シュミレータ、モンテカルロ法、通信データの暗号化に最適です。
機能に応じて3つのラインナップがあります。

SRG-SDK Prime Free
無料配布版です。32ビット（4,294,967,296個）のランダム・シードを提供します。
シングルタスク、シングルスレッド版です。同時に2つ以上の乱数系列を作成できません。
対応OS：Windows32 のみ。

SRG-SDK Prime Standard
有償製品。32ビット（4,294,967,296個）のランダム・シードを提供します。
完全なマルチタスク、マルチスレッドにて使用できます。対応OS：Windows32、Linux

SRG-SDK Prime Enterprise
有償製品。最大10の27000乗のランダム・シード提供します。
完全なマルチタスク、マルチスレッドにて使用できます。対応OS：Windows32、Linux

2. SRG-SDK 3種 Free , Standard , Enterprise

ファイル操作向けに特化したSDKです。ファイルを「暗号化して保存（書き込み）」「復号化して開く（読み込む）」操作を行います。

3. SRG-SDK Environment

特別バージョンです。 C言語標準ファイルライブラリ上位互換の暗号化ライブラリです。 暗号化を意識する事無く、アプリケーションの暗号化を実現します。

1.3 製品パッケージ

製品はCDメディアまたはDVDメディアにて提供されます。本文中の「製品メディア」と書かれている場合には、CD メディアまたは DVD メディアのことを指しています。[SRG-SDK Prime Free]はダウンロード版のみです。本文中に「製品メディア」と書かれている場合には、ダウンロード後に解凍したフォルダのことを指しています。メディアの中には擬似乱数発生装置を簡単に使えるように機能をまとめたライブラリが格納され、コンピュータソフトウェアに乱数機能とこれを利用した暗号機能を提供します。 また本マニュアルも含まれます。

1.4 使用環境

対応 OS：32bit の WindowsOS （推奨：WindowsXP 環境）
対応言語：C 言語
必要容量：10 MB
供給形態：DLL ないし共有ライブラリ

2. インストール方法

1. 製品メディア内のFileEncSDKフォルダを開発環境にコピーします。
2. 各ファイルを各自の開発環境にコピーします。
   ■srg_sdk.dll・・・本SDK本体
   ■srg_sdk.lib・・・リンク情報を格納ファイル
   ■soliton_rnd_type.h・・・基本型ファイル
   ■soliton_rnd_prime.h・・・関数のプロトタイプ宣言
3. プロジェクタに登録後、利用可能になります。

3. 簡単な使い方

1. 共有ライブラリの使い方は次のプロジェクトを参考にしてください。
2. SRG-SDK Prime->TestSampleフォルダに「TestSample.dsw」というVC6用のプロジェクタがあります。
3. このファイルを起動し、ビルド、実行してください。サンプルプログラムが動作します。

4. 関数リファレンス

擬似乱数を生成する関数群、ならびに乱数の生成関数を説明します。

4.1. 関数一覧

【 Meta Level functions 】

_inf_soliton_lib_ver(void)
本ＳＤＫのバージョン情報を取得する。

_inf_soliton_lib_inz(void)
本ＳＤＫの初期化を行う。一番最初に必ず呼ぶ必要がある。

_inf_soliton_lib_meta_inz(unsigned char *tbl, int n)
_inf_soliton_lib_inz()の補助関数。
本関数は使用するシードの基本シードを事前に指定する。省略可能。

【Inz Level functions 】

_inf_soliton_seed32(_inf_soliton_rnd *rnd_tbl, unsigned long seed);
乱数を初期化する。乱数に３２ビットのシードを与える。

_inf_soliton_seed_n(_inf_soliton_rnd *rnd_tbl, unsigned char *seed_buf, int n);
乱数を初期化する。乱数に任意の長さのシードを与える。

【 Standard Level functions 】

_inf_soliton_rnd_gen(_inf_soliton_rnd *rnd_tbl);
１バイトの乱数を生成する。

_inf_soliton_rnd_gen__high(_inf_soliton_rnd *rnd_tbl);
１バイトの最高品質の乱数を生成する。

_inf_soliton_rnd_gen32(_inf_soliton_rnd *rnd_tbl);
３２ビット（４バイト）の乱数を生成する。

_inf_soliton_rnd_gen31(_inf_soliton_rnd *rnd_tbl);
３１ビット（４バイト）正符号の乱数を生成する。

_inf_soliton_rnd_gen_blk(_inf_soliton_rnd *rnd_tbl, unsigned char *buf, unsigned long no);
任意の長さの乱数を生成する。

_inf_soliton_rnd_enc_blk(_inf_soliton_rnd *rnd_tbl, unsigned char *buf, unsigned long no);
任意の長さの乱数を生成し、指定したメモリを暗号／復号化する。

_inf_soliton_rnd_gen_blk4(_inf_soliton_rnd *rnd_tbl, unsigned long *buf, unsigned long no);
任意の長さの乱数を生成する。生成する乱数は４バイト単位である。

_inf_soliton_rnd_gen1(_inf_soliton_rnd *rnd_tbl);
浮動小数点型でゼロ～１の乱数を生成。

_inf_soliton_rnd_gen2(_inf_soliton_rnd *rnd_tbl);
浮動小数点型でゼロ～１の乱数を生成。ただし、１を含まず。

_inf_soliton_rnd_gen3(_inf_soliton_rnd *rnd_tbl);
浮動小数点型でゼロ～１の乱数を生成。ただし、ゼロを含まず。

_inf_soliton_rnd_gen53(_inf_soliton_rnd *rnd_tbl);
浮動小数点型でゼロ～1の乱数を生成。ただし、ゼロと1を含まず。

unsigned long _inf_soliton_lib_ver(void)

本ＳＤＫのバージョン情報を取得する。
生成する乱数はSDKのバージョンにより異なる可能性がある。これより用意している。
データの暗号化に本SDKを使用する場合、事前にバージョンをチェックを推奨。
暗号・復号は同じバージョンのSDKで保障する。

■引数

none

■戻り値

バージョン番号を返す。１０進表記で次の意味を持つ。
VLFの形式をつ。
V=バージョン番号。
L=シードのリミット。ゼロ＝制限なし。４＝４バイト（３２ビット）シード。
F=フリー版か否か。有値の時にフリー版。ゼロの時は有料版。

例： １４１　の時は、バージョン１。４バイトシードまで有効。フリー版を示す。

■Note…

void _inf_soliton_lib_inz(void)

本SDKの初期化を行う。一番最初に必ず呼ぶ必要がある。
本関数の呼び出しにより本SDKは使用可能になる。

■引数

none

■戻り値

none

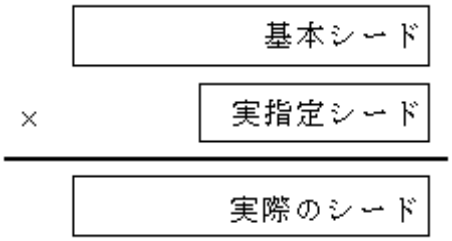
■Note…

void _inf_soliton_lib_meta_inz(unsigned char *tbl, int n)

[Enterprise]でのみ本関数は利用できます。他の版では使用できません。

_inf_soliton_lib_inz()の補助関数。
本関数は使用するシードの基本シードを事前に指定する。この指定。省略可能です。
マルチタスク、マルチスレッドにおいて複数の乱数を高速に生成したい時に使用します。

１０の２７０００乗のシードを指定するためには１２Kバイト以上のメモリが必要です。
乱数のシードを指定する毎に１２Kバイトで指定すると初期化に時間がかかります。
SDKは基本シードとし、実指定シードを用意し、この2つの複合で実際のシードを決定します。



リセットしたい場合は、本体アプリを再起動する必要があります。
本関数はライブラリ起動にあたり一度だけ呼ぶようにすることを推奨します。

■引数
unsigned char＊seed; // 最大９２１３バイトのバイナリデータ
int n; // シードの長さを指定する。
// １～９２１３バイトの範囲で指定可能です。
■戻り値
none
■Note…

```
#include        ″soliton_rnd_prime.h″
                .
                .
                .
    _inf_soliton_lib_inz( void )      // 本SDKの初期化
    _inf_soliton_lib_meta_inz( (unsigned char*)”abcd”, 4);
```

4.3. Inz Level functions

void _inf_soliton_seed32(_inf_soliton_rnd *rnd_tbl, unsigned long seed)

乱数に32ビットシードを与え初期化する。

■引数

```
_inf_soliton_rnd *rnd_tbl;      // 乱数の作業用メモリ
                                // [ Free ]では予約語。注1。
unsigned long seed;             // 32ビットのシードを与える
```

■戻り値

note

■Note…

本SDKは、マルチタスク、マルチスレッドで複数の乱数列を発生する。
本関数はこの乱数列を初期化する。
各乱数列は「_inf_soliton_rnd」により管理する。
本SDKは、使用に当たり必ず_inf_soliton_seed32()か_inf_soliton_seed_n()のいづれかを呼ぶ必要がある。
注意1:[Free]版は、この内容を無視し、内蔵する作業用メモリを使用する。この結果、複数の乱数を同時に管理できない。

```
---- Sample void _inf_soliton_seed32( _inf_soliton_rnd *rnd_tbl, unsigned long seed ) ---
#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    "soliton_rnd_prime.h"

void sample1()
{
    _inf_soliton_rnd rnd_tbl;

    _inf_soliton_lib_inz( void );                   // 本SDKの初期化
    // _inf_soliton_lib_meta_inz( (unsigned char*)"abcd", 4); 省略可。無理して使う必要なし。
    _inf_soliton_seed32( &rnd_tbl, 100 );           //  乱数を十進100にて初期化。
        .
        .
        .
}
```

void _inf_soliton_seed_n(_inf_soliton_rnd *rnd_tbl, unsigned char *seed_buf, int n)

乱数に最大９２１３バイトのシードを与え初期化する。

■引数

_inf_soliton_rnd *rnd_tbl; // 乱数の作業用メモリ
// [Free]では予約語。注１。
unsigned char *seed_buf; // シードとなるメモリ情報
int n; // シードの長さを指定する。
// [Free]では１～４バイトの範囲で指定可能です。
// [Standard]では１～４バイトの範囲で指定可能です。
// [Enterprise]では１～９２１３の範囲で指定可能です。

■戻り値

note

■Note…

本SDKは、マルチタスク、マルチスレッドで複数の乱数列を発生する。
本関数はこの乱数列を初期化する。
シードが長い程、初期化に時間がかかるので注意すること。
各乱数列は「_inf_soliton_rnd」により管理する。
本SDKは、使用に当たり必ず_inf_soliton_seed32()か_inf_soliton_seed_n()のいづれかを呼ぶ必要がある。

```
---- Sample _inf_soliton_seed_n( _inf_soliton_rnd *rnd_tbl, unsigned char *seed_buf, int n ) ---
#include	<stdio.h>
#include	<stdlib.h>
#include	<string.h>
#include	"soliton_rnd_prime.h"

void sample()
{
	_inf_soliton_rnd rnd_tbl;

	_inf_soliton_lib_inz( void );			// 本SDKの初期化
	// _inf_soliton_lib_meta_inz( (unsigned char*)"abcd", 4); 省略可。無理して使う必要なし。
	_inf_soliton_seed_n( (unsigned char*)"1234", 4 );  //  乱数を文字列"1234"にて初期化。
		.
		.
		.
}
```

4.4. Standard Level functions

最高品質の1バイトの乱数を生成する。

■引数

```
_inf_soliton_rnd *rnd_tbl;      // 乱数の作業用メモリ
                                // [ Free ]では予約機能とし使用しない。
```

■戻り値

unsigned char型の1バイトの乱数。
0～0xffの値。

■Note

unsigned char _inf_soliton_rnd_gen(_inf_soliton_rnd *rnd_tbl)
unsigned char _inf_soliton_rnd_gen_high(_inf_soliton_rnd *rnd_tbl)

１バイトの乱数を生成する。

■引数

_inf_soliton_rnd *rnd_tbl;　　// 乱数の作業用メモリ
// [Free]では予約機能とし使用しない。

■戻り値

unsigned char型の１バイトの乱数。
0～0xffの値。

■Note

_inf_soliton_rnd_gen( )は通常品質の乱数を生成する。
_inf_soliton_rnd_gen_high()は可能な限り高品質の乱数を生成する。
本SDKの他の乱数生成関数は、_inf_soliton_rnd_gen()をベースにしている。
可能な限り高い品質の乱数を必要とするときに、_inf_soliton_rnd_gen_high()を使用する。

```
---- Sample unsigned char  _inf_soliton_rnd_gen( _inf_soliton_rnd *rnd_tbl ) ---
#include	<stdio.h>
#include	<stdlib.h>
#include	<string.h>
#include	"soliton_rnd_prime.h"

void sample1()
{
	_inf_soliton_rnd rnd_tbl;
	int	i;
	unsigned char c;

	_inf_soliton_lib_inz();			// 本SDKの初期化
	// _inf_soliton_lib_meta_inz( (unsigned char*)"abcd", 4); 省略可。無理して使う必要なし。
	_inf_soliton_seed32( &rnd_tbl, 100 );		//  乱数を十進100にて初期化。

	for( i=0 ; i<10 ; i++ ) {
		c = _inf_soliton_rnd_gen( &rnd_tbl );
		// c = _inf_soliton_rnd_gen_high( &rnd_tbl ); // 高品位乱数を生成したい
								// 時はこちらを使う。
		printf( "RANDOM: %0x¥n", c );
	}
}
```

unsigned long _inf_soliton_rnd_gen32(_inf_soliton_rnd *rnd_tbl)

4バイトの乱数を生成する。

■引数

_inf_soliton_rnd *rnd_tbl;　　// 乱数の作業用メモリ
// [Free]では予約機能とし使用しない。

■戻り値

unsigned long型の4バイトの乱数。
0～0xffffffffの値。

■Note…

---- Sample unsigned char _inf_soliton_rnd_gen32(_inf_soliton_rnd *rnd_tbl) ---

```
#include	<stdio.h>
#include	<stdlib.h>
#include	<string.h>
#include	"soliton_rnd_prime.h"

void sample2()
{
	_inf_soliton_rnd rnd_tbl;
	int	i;
	unsigned long c;

	_inf_soliton_lib_inz();			// 本SDKの初期化
	// _inf_soliton_lib_meta_inz( (unsigned char*)"abcd", 4); 省略可。無理して使う必要なし。
	_inf_soliton_seed32( &rnd_tbl, 100 );		//  乱数を十進100にて初期化。

	for( i=0 ; i<10 ; i++ ) {
		c = _inf_soliton_rnd_gen32( &rnd_tbl );
		printf( "RANDOM: %0x¥n", c );
	}
}
```

unsigned long _inf_soliton_rnd_gen31(_inf_soliton_rnd *rnd_tbl)

31ビットの4バイトの乱数を生成する。

■引数

_inf_soliton_rnd *rnd_tbl;　　// 乱数の作業用メモリ
// [Free]では予約機能とし使用しない。

■戻り値

unsigned long型の4バイトの乱数。
0～0x7fffffffの値。

■Note…

---- Sample unsigned char _inf_soliton_rnd_gen31(_inf_soliton_rnd *rnd_tbl) ---

```
#include	<stdio.h>
#include	<stdlib.h>
#include	<string.h>
#include	"soliton_rnd_prime.h"

void sample3()
{
	_inf_soliton_rnd rnd_tbl;
	int	i;
	unsigned long c;

	_inf_soliton_lib_inz();			// 本SDKの初期化
	// _inf_soliton_lib_meta_inz( (unsigned char*)"abcd", 4); 省略可。無理して使う必要なし。
	_inf_soliton_seed32( &rnd_tbl, 100 );		//  乱数を十進100にて初期化。

	for( i=0 ; i<10 ; i++ ) {
		c = _inf_soliton_rnd_gen31( &rnd_tbl );
		printf( "RANDOM: %0x¥n", c );
	}
}
```

void _inf_soliton_rnd_gen_blk(_inf_soliton_rnd *rnd_tbl, unsigned char *buf, unsigned long no)

任意の乱数をブロック生成する。

■引数

```
_inf_soliton_rnd *rnd_tbl;      // 乱数の作業用メモリ
                                // [ Free ]では予約機能とし使用しない。
unsigned char *buf;             // 乱数の格納先のメモリ
unsigned long no;               // 生成したい乱数のバイト数
```

■戻り値

none

■Note…

---- Sample unsigned char _inf_soliton_rnd_gen_blk() ---

```
#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    "soliton_rnd_prime.h"

void sample4()
{
    _inf_soliton_rnd rnd_tbl;
    int     n;
    unsigned char       buf[32];

    _inf_soliton_lib_inz();                 // 本SDKの初期化
    // _inf_soliton_lib_meta_inz( (unsigned char*)"abcd", 4); 省略可。無理して使う必要なし。
    _inf_soliton_seed32( &rnd_tbl, 100 );       //  乱数を十進100にて初期化。

    _inf_soliton_rnd_gen_blk( &rnd_tbl, buf, 32 );
}
```

void _inf_soliton_rnd_enc_blk(_inf_soliton_rnd *rnd_tbl, unsigned char *buf, unsigned long no)

メモリブロックを暗号化／復号化する。

■引数

```
_inf_soliton_rnd *rnd_tbl;      // 乱数の作業用メモリ
                                // [ Free ]では予約機能とし使用しない。
unsigned char *buf;             // 暗号／復号化したいデータの格納先
unsigned long no;               // データの長さ
```

■戻り値

none

■Note

メモリデータに対して本関数を呼ぶと、データに乱数をXORをかけて暗号化する。
同じ条件で本関数を呼べば復号化する。
メモリ上での高速暗号化／復号化が必要な時に本関数を使用する。

---- Sample unsigned char _inf_soliton_rnd_enc_blk() ---

```
#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    "soliton_rnd_prime.h"

void sample5()
{
    _inf_soliton_rnd rnd_tbl;
    unsigned char       buf[32];

    memset( buf, 0, sizeof(buf) );
    strcpy( (char*)buf, "Hello World !" );
    _inf_soliton_lib_inz();                 // 本SDKの初期化
    _inf_soliton_seed32( &rnd_tbl, 100 );       //  乱数を十進100にて初期化。

    // 暗号化
    _inf_soliton_rnd_enc_blk( &rnd_tbl, buf, 32 );

    // 復号化
    _inf_soliton_seed32( &rnd_tbl, 100 );       //  乱数を十進100にて初期化。
    _inf_soliton_rnd_enc_blk( &rnd_tbl, buf, 32 );
    printf( "%s¥n", buf );
}
```

double _inf_soliton_rnd_gen1(_inf_soliton_rnd *rnd_tbl)

浮動小数点型でゼロ～１の乱数を生成。

■引数

_inf_soliton_rnd *rnd_tbl;　　// 乱数の作業用メモリ
// [Free]では予約機能とし使用しない。

■戻り値

double型の乱数。

■Note…

---- Sample unsigned double _inf_soliton_rnd_gen1() ---

```
#include	<stdio.h>
#include	<stdlib.h>
#include	<string.h>
#include	"soliton_rnd_prime.h"

void sample6()
{
	_inf_soliton_rnd rnd_tbl;
	int	i;
	double c;

	_inf_soliton_lib_inz();			// 本SDKの初期化
	// _inf_soliton_lib_meta_inz( (unsigned char*)"abcd", 4); 省略可。無理して使う必要なし。
	_inf_soliton_seed32( &rnd_tbl, 100 );		//  乱数を十進100にて初期化。

	for( i=0 ; i<10 ; i++ ) {
		c = _inf_soliton_rnd_gen1( &rnd_tbl );
		printf( "RANDOM: %f¥n", c );
	}
}
```

double _inf_soliton_rnd_gen2(_inf_soliton_rnd *rnd_tbl)

浮動小数点型でゼロ～１の乱数を生成。ただし、１を含まず。

■引数

_inf_soliton_rnd *rnd_tbl;　　// 乱数の作業用メモリ
// [Free]では予約機能とし使用しない。

■戻り値

double型の乱数。

■Note…

---- Sample unsigned double _inf_soliton_rnd_gen2() ---

```
#include	<stdio.h>
#include	<stdlib.h>
#include	<string.h>
#include	"soliton_rnd_prime.h"

void sample7()
{
	_inf_soliton_rnd rnd_tbl;
	int	i;
	double c;

	_inf_soliton_lib_inz();			// 本SDKの初期化
	// _inf_soliton_lib_meta_inz( (unsigned char*)"abcd", 4); 省略可。無理して使う必要なし。
	_inf_soliton_seed32( &rnd_tbl, 100 );		//  乱数を十進100にて初期化。

	for( i=0 ; i<10 ; i++ ) {
		c = _inf_soliton_rnd_gen2( &rnd_tbl );
		printf( "RANDOM: %f¥n", c );
	}
}
```

double _inf_soliton_rnd_gen3(_inf_soliton_rnd *rnd_tbl)

浮動小数点型でゼロ～１の乱数を生成。ただし、ゼロを含まず。

■引数

_inf_soliton_rnd *rnd_tbl;　　// 乱数の作業用メモリ
// [Free]では予約機能とし使用しない。

■戻り値

double型の乱数。

■Note…

---- Sample unsigned double _inf_soliton_rnd_gen3() ---

```
#include	<stdio.h>
#include	<stdlib.h>
#include	<string.h>
#include	"soliton_rnd_prime.h"

void sample8()
{
	_inf_soliton_rnd rnd_tbl;
	int	i;
	double c;

	_inf_soliton_lib_inz();			// 本SDKの初期化
	// _inf_soliton_lib_meta_inz( (unsigned char*)"abcd", 4); 省略可。無理して使う必要なし。
	_inf_soliton_seed32( &rnd_tbl, 100 );		//  乱数を十進100にて初期化。

	for( i=0 ; i<10 ; i++ ) {
		c = _inf_soliton_rnd_gen3( &rnd_tbl );
		printf( "RANDOM: %f¥n", c );
	}
}
```

double _inf_soliton_rnd_gen53(_inf_soliton_rnd *rnd_tbl)

浮動小数点型でゼロ～１の乱数を生成。ただし、ゼロと1を含まず。

■引数

_inf_soliton_rnd *rnd_tbl;　　// 乱数の作業用メモリ
// [Free]では予約機能とし使用しない。

■戻り値

double型の乱数。

■Note…

---- Sample unsigned double _inf_soliton_rnd_gen53() ---

```
#include	<stdio.h>
#include	<stdlib.h>
#include	<string.h>
#include	"soliton_rnd_prime.h"

void sample9()
{
	_inf_soliton_rnd rnd_tbl;
	int	i;
	double c;

	_inf_soliton_lib_inz();			// 本SDKの初期化
	// _inf_soliton_lib_meta_inz( (unsigned char*)"abcd", 4); 省略可。無理して使う必要なし。
	_inf_soliton_seed32( &rnd_tbl, 100 );		//  乱数を十進100にて初期化。

	for( i=0 ; i<10 ; i++ ) {
		c = _inf_soliton_rnd_gen53( &rnd_tbl );
		printf( "RANDOM: %f¥n", c );
	}
}
```

4. Appendix

Q1. .マルチタスク、マルチスレッドで乱数を発生させるにはどうしたら良いですか？
A1. 基本的に必要なタスク、スレッドの数だけ作業用メモリ「_inf_soliton_rnd」を割り振ります。
次のサンプルを参考にしてください。

```
_inf_soliton_rnd rnd_tbl1;      // 乱数１用の作業メモリを確保
_inf_soliton_rnd rnd_tbl1;      // 乱数２用の作業メモリを確保
        ・
        ・
        ・

_inf_soliton_lib_inz();             // 本SDKの初期化
_inf_soliton_seed32( &rnd_tbl1, 1 );  //  乱数をシード1にて初期化。
_inf_soliton_seed32( &rnd_tbl2, 2 );  //  乱数をシード2にて初期化。

// 以後、 rnd_tbl1にてシード1による乱数系列を生成する。
// 以後、 rnd_tbl2にてシード2による乱数系列を生成する。

r1  = _inf_soliton_rnd_gen( &rnd_tbl1 );       // rnd_tbl1にて乱数生成
r2  = _inf_soliton_rnd_gen( &rnd_tbl2 );       // rnd_tbl2にて乱数生成
```

本SDKは、サンプル中、個別に確保したrnd_tbl1とrnd_tbl2は独立して使用できます。従い、
・同時に必要とする乱数系列の数だけ、_inf_soliton_rnd型の作業用メモリを確保してください。
・マルチスレッド、マルチタスク環境で使用する場合は_inf_soliton_rnd型の作業用メモリをローカル変数として確保してください。

Q2. Visual Basicで使用できますか？
A2. 本SDKはWindows32標準のDLLで供給します。DLLはVisual Basicでも使用できます。
ただし、弊社では動作確認をしていません。サポート外です。動作を保障しません。

Q3. マルチタスク、マルチスレッドなプログラムに[Free]版で組み込んで使用できますか？
A3. 使用できません。[Free]版は乱数生成の作業用メモリを一つグローバル変数として保持します。
従い、マルチタスク、マルチスレッドで使用するとグローバル変数の呼び出し競合が発生します。
この結果、生成する乱数の品質が低下します。

(C)2008, 有限会社電机本舗