SpeeDBee Hive

カスタムコレクタ（C）

SALTYSTER
SPEED AND SMART WITH SPIRITS

改訂履歴

版	概要	更新日
1	新規作成	2021/4/2
2	Windows 版の追加	2021/7/29
3	誤記修正 カスタムコレクタ(Python)に関する記述を追記	2022/1/25

目次

1.カスタムコレクタとは

SpeeDBee Hive では、データの収集を行うモジュールのことを"コレクタ"と呼びます。
標準では PLC 等のデータを収集する"コレクタ"が用意されていますが、ユーザーが独自にコレクタを開発、運用することも可能です。コレクタの開発には C 言語、もしくは、Python を使用できます。
本ドキュメントでは C 言語によるカスタムコレクタの開発方法を説明します。

2.カスタムコレクタのサンプル

カスタムコレクタの作り方の説明のため、単純なデータ収集を行うシンプルなカスタムコレクタをサンプルとして提供しています。
SpeeDBee Hive をインストール済みの場合、カスタムコレクタサンプルも下記の場所にインストールされています。

Linux

```
$ ls /usr/local/share/speedbeehive/example
Makefile fixarray.c randomsin.c
```

Windows10

```
>dir "C:/Program Files/SALTYSTER/SpeeDBeeHive/share/speedbeehive/example"
2021/07/29  17:00    <DIR>          .
2021/07/29  17:00    <DIR>          ..
2021/07/29  16:49           5,324 fixarray.c
2021/07/29  16:49             439 Makefile
2021/07/29  16:49           3,694 randomsin.c
               3 個のファイル              9,457 バイト
```

2.1. カスタムコレクタのサンプルの内容

カスタムコレクタのサンプルとして 2 つのコレクタを提供しています。

- randomsin
  SIN 波の振幅値(-1～+1)と、ランダムな 0~100 までの整数値の 2 つのデータを登録するコレクタのサンプルです。
  データはいずれも単純なスカラ値で、前者は DOUBLE 型、後者は INT32 型としています。
- fixarray
  円運動する物体の X-Y-Z 座標値をデータとして登録するコレクタのサンプルです。

データは３要素の DOUBLE 型配列としています。

2.2. 準備

カスタムコレクタの開発のため、下記のソフトウェアを事前にインストールしてください。

2.2.1.Linux

- gcc
- make

2.2.2.Windows

- Visual Studio 2019

2.3. ビルド

2.3.1.Linux

カスタムコレクタをホームディレクトリにコピーした上で、make を実行してください。

```
$ cp -r /usr/local/share/speedbeehive/example ~/custom-example
$ cd ~/custom-example
$ make
cc -g -O2 -Wall -I/usr/local/include -shared -fPIC -L/usr/local/lib -o cltrfixarray.so
fixarray.c -pthread -lm
cc -g -O2 -Wall -I/usr/local/include -shared -fPIC -L/usr/local/lib -o cltrrandomsin.so
randomsin.c -pthread -lm
```

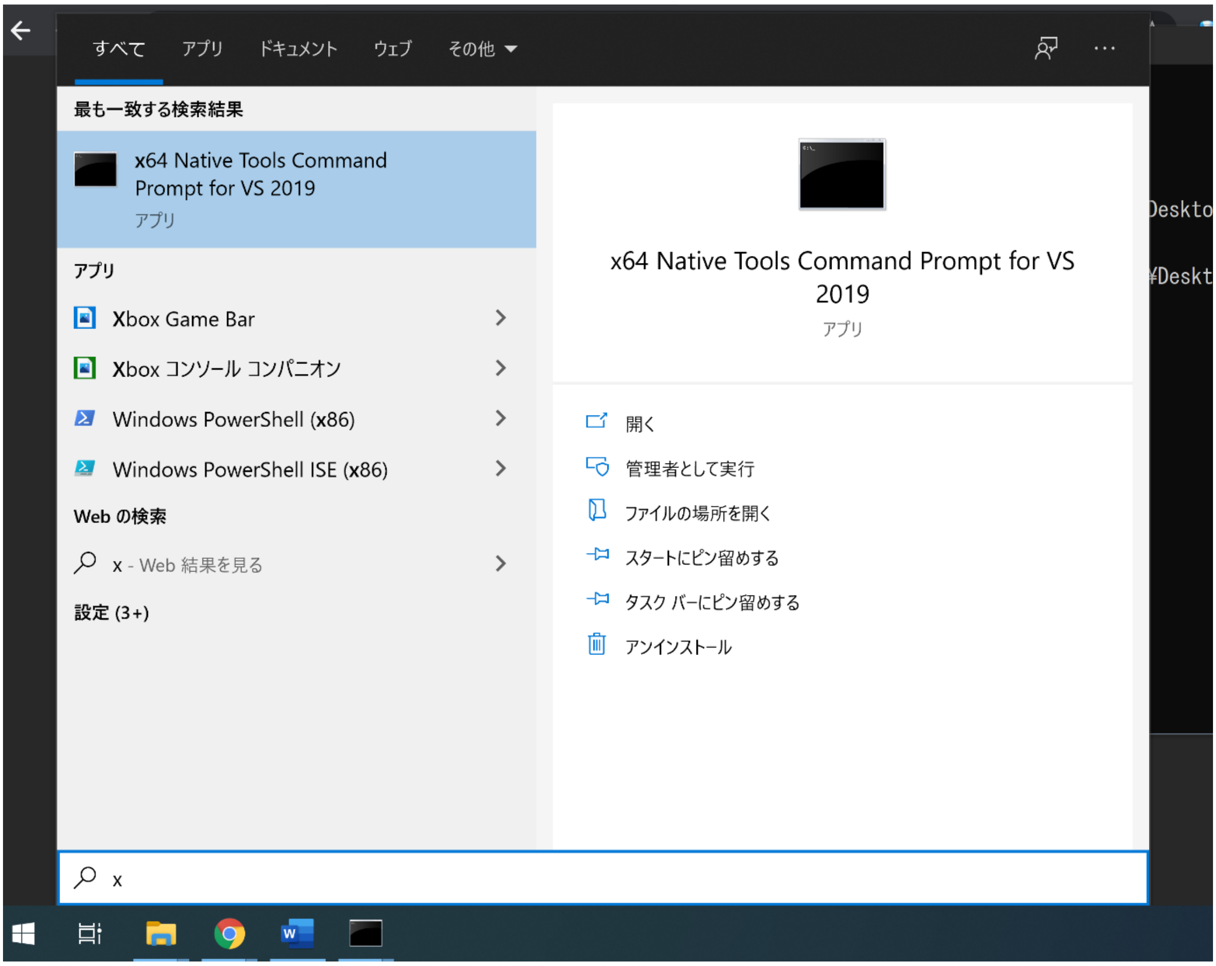
その後、生成された cltrfixarray.so, cltrrandomsin.so を/var/speedbeehive/libs にインストールします。

```
$ cd ~/custom-example
$ sudo make install
install -d /var/speedbeehive/libs
install -m 644 cltrfixarray.so cltrrandomsin.so /var/speedbeehive/libs
```

※この配置パス(/var/speedbeehive/libs)は必須の場所ではありません。

2.3.2.Windows

x64 Native Tools Command Prompt for VS 2019 を実行してください。



x64 Native Tools Command Prompt for VS 2019 上でカスタムコレクタをデスクトップにコピーし、nmake を実行してください。

```
> xcopy /I "C:¥Program Files¥SALTYSTER¥SpeeDBeeHive¥share¥example"
C:¥Users¥user_name¥Desktop¥test¥example
> cd  C:¥Users¥$user_name¥Desktop¥test¥example
> nmake
Microsoft(R) Program Maintenance Utility Version 14.28.29915.0
Copyright (C) Microsoft Corporation.  All rights reserved.
```

その後、生成された cltrfixarray.dll, cltrrandomsin.dll を C:¥ProgramData¥SALTYSTER¥SpeeDBeeHive¥libs にインストールします。

```
> xcopy cltrfixarray.dll C:¥ProgramData¥SALTYSTER¥SpeeDBeeHive¥libs
> xcopy cltrrandomsin.dll C:¥ProgramData¥SALTYSTER¥SpeeDBeeHive¥libs
```

※この配置パス(C:¥ProgramData¥SALTYSTER¥SpeeDBeeHive¥libs)は必須の場所ではありません。

2.3.3.配置場所変更方法

Web UI のカスタムモジュールパスを変更することで配置場所を変更する事が可能です。

※Windows 版ではカスタムモジュールパスの初期値は
「C:¥ProgramData¥SALTYSTER¥SpeeDBeeHive¥libs」となっています。

2. 4. 利用設定

インストールしたカスタムコレクタを利用する場合、設定画面から設定します。

＊ここではライブラリに「cltrrandomsin.dll(.so)」を使って説明します

2.4.1.カスタムコレクタ設定

① サブメニュー「カスタム」のをクリックすると新規登録画面が表示されます

② 各項目を入力し、「保存」をクリックし設定を保存します。

設定が必要な項目は以下の通りです

入力項目	説明
カスタム名	データを収集するコレクタ名　※コレクタ全体で重複不可
タイプ	シェアードライブラリのタイプ、Linux では「so」固定 ※保存後は変更不可
ライブラリ	カスタムモジュールパスに配置したシェアードライブラリ。 ※保存後は変更不可
パラメータ	ライブラリ内で利用される引数（文字列） cltrrandomsin.dll(.so)では入力の必要はありません ※clrtfixarray.dll(.so)では「0.0, 0.0, 1.0」などを入力してください。 実行時に上記で設定されたパラメータの文字列により収集するデータが変化します。

2.4.2.カスタムコレクタ 参照・更新・削除

① サブメニューから参照するカスタムコレクタをクリックします。

② 変更項目を入力後、「保存」をクリックし、設定を更新します。

③ データを削除したい場合は、「削除」をクリックし、設定を削除します。

> カスタムコレクタを削除すると、エミッタにて、そのデータ項目を利用している場合、自動削除されます。

2.4.3.データ初期化

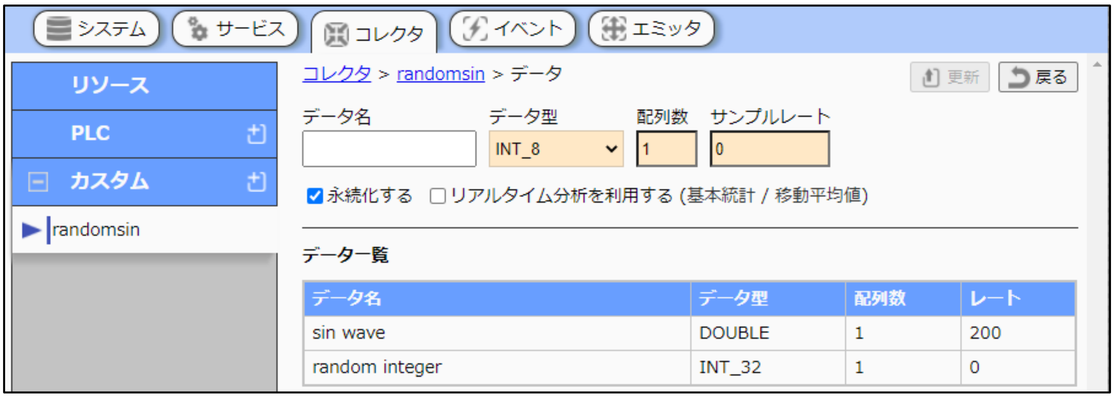
① 「初期化」をクリックします。

② データ一覧の表に「sin wave」「random integer」の二つが追加されます。

初期化済みのデータがすでに存在する場合、一度データすべてを削除してから、再初期化が行われます。そのため、エミッタにて、そのデータ項目を利用している場合、自動削除されます。

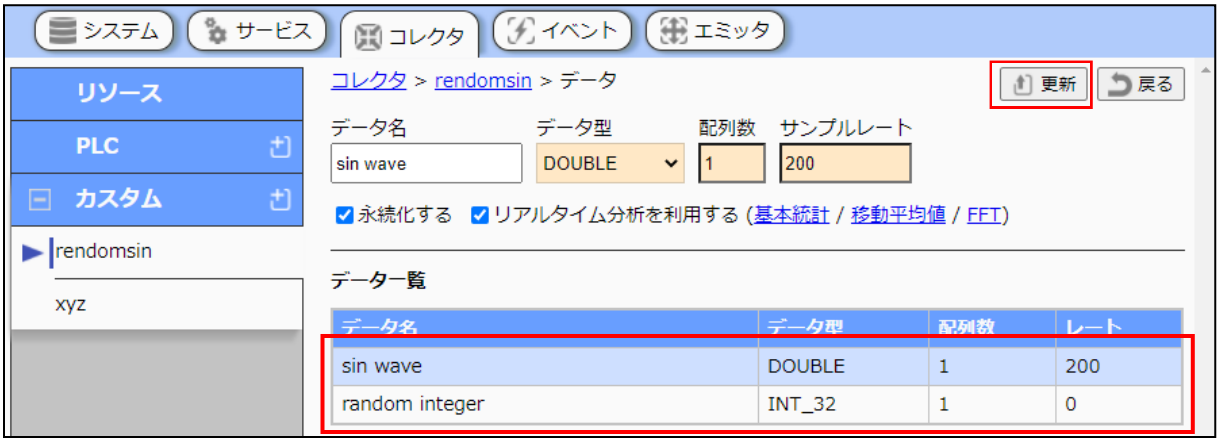
2.4.4.データ編集

① 「編集」をクリックし、データ編集画面に移動します。



② 下部のデーター覧から設定したい項目を選択します。

③ 各項目を変更し、「保存」をクリックし設定を保存します。



変更が可能な項目は以下の通りです

入力項目	説明
データ名	データ名称。コレクタ内で重複不可
データ型	取得するデータ型
配列数	取得する配列数
サンプルレート	1 秒間に取得する値の数
永続化する	ストレージ上に保存する
リアルタイム分析を利用する	リアルタイム分析を利用する チェック時には基本統計、移動平均、FFT をクリックすることで編集画面を表示（詳細は「ユーザマニュアル 8.補足」を参照） ＊FFT はサンプルレートが 1 以上の場合のみ選択可能

※データ型、配列数、サンプルレートは変更できません。

3.カスタムコレクタのソースコードの説明

この節ではカスタムコレクタサンプルの randomsin のソースコード randomsin.c をもとにカスタムコレクタの作りについて説明します。randomsin.c は以下の場所に配置されています。

Linux: /usr/local/share/speedbeehive/example/randomsin.c

Windows: C:/Program Files/SALTYSTER/SpeeDBeeHive/share/example/randomsin.c

3.1. インクルードヘッダ

カスタムコレクタを実装する際は下記のヘッダファイルをインクルードしてください。

このヘッダは以下の場所に配置されています。

Linux: /usr/local/include/

Windows: C:/Program Files/SALTYSTER/SpeeDBeeHive/include/

以降に説明する種々の構造体や関数のプロトタイプが宣言されていますので、詳細はこちらを参照してください。

```
#include "sdcltr_collector.h"
```

3.2. コレクタ情報定義

下記はカスタムコレクタの固有情報を SpeeDBee Hive に伝えるための定義データです。

COLLECTOR_INFO 構造体の実態定義、および、その定義を EXPORT_COLLECTOR_INFO() で SpeeDBee Hive に登録します。

```
/* コレクタ定義データ：名前、説明、パラメータ、出力データを記述する． */
const static COLLECTOR_INFO info = {
  .name         = "random&sin",          // コレクタの名称
  .description   = "ランダム値とサイン波", // コレクタの説明
  .multiple      = true,                 // 複数インスタンスを許すかどうか
  .functions     = {
    .constructor = constructor,
    .destructor  = NULL,
    .main        = mainloop,
  },
};

/* コレクタ定義データを外部シンボルに出力するマクロ（必須）．  */
EXPORT_COLLECTOR_INFO(info);
```

定義が必要な情報は下記のとおりです。

No	項目	説明

1	name	コレクタの名称をもつ文字列ポインタ。 マルチバイト文字の場合は UTF-8 の文字列としてください。
2	description	このコレクタの内容・仕様を説明する文字列。 OPC UA の機能を有効にした場合に、コレクタの情報を表示する際に使用されます。
3	multiple	このコレクタのモジュールを同時に複数実行することが可能か否かを示すフラグ。 これを true にする場合、下記の functions の関数が同時並列に実行される可能性を考慮した実装が必要です。 具体的にはグローバル領域の変数使用を避ける、もしくは、排他制御を行うなどを検討してください。
4	functions.constructor	SpeeDBee Hive がこのコレクタの実行を開始する際に最初に呼び出す関数です。 Web UI で設定したパラメータが引数として渡されます。 また、このコレクタが出力するデータの構造を関数の戻り値として返す必要があります。
5	functions.destructor	SpeeDBee Hive がこのコレクタの実行を終了する際に呼び出す関数です。 constructor でメモリの確保などを行った場合、ここで解放する必要があります。
6	functions.main	コレクタのメイン処理を行うための関数です。 この関数内で集めたいデータを取得し、指定の関数 I/F を利用してそのデータを SpeeDBee Hive に登録します。

3.3. コンストラクタ

このコレクタの開始時に最初に実行される関数です。

引数として"COLLECTOR *collector"と"params"を受け取っています。
collector はこの SpeeDBee Hive が用意している様々な関数をコールするために必要になります。
params は Web UI でパラメータとして定義した文字列です。
パラメータを受け取って動作を変えた場合は参照してください。

```
static const COLLECTOR_OUTPUT_TYPE *constructor(COLLECTOR *collector, const char *params)
{

  /* コレクタが出力するデータの定義．データ０は SIN 波の浮動小数点数、
     データ１はランダムな整数．最後は NONE にする． */
  const static COLLECTOR_OUTPUT_TYPE output_types[] = {
    // type            array type         description        sampling rate
    { SDCLTR_DOUBLE,    SDCLTR_SCALAR,      "sin wave",         200          },
    { SDCLTR_INT32,     SDCLTR_SCALAR,      "random integer",   0            },
    { SDCLTR_NOTYPE },  // 終端は必ず NOTYPE とする
  };
  return output_types;
}
```

コンストラクタは COLLECTOR_OUTPUT_TYPE の配列を戻り値として返す必要があります。
この構造体配列は、コレクタが収集するデータの個数やそれぞれの型、頻度を定義しています。

上記例の場合、データは２つあり、
　１つ目のデータは DOUBLE 型、頻度は 200Hz、
　２つ目のデータは INT32 型、頻度は不定期
となります。配列の最後は、必ず、NOTYPE で終端する必要があります。

3.4. メインループ

カスタムコレクタのデータ収集のメイン処理を記述する関数です。

このサンプルでは、単純に sdcltr_interval_call()関数を呼んでいます。
これにより、引数に与えられた proc 関数が定期的にコールされ、コレクタの終了通知が来ると自動的に終了します。

```
static int mainloop(COLLECTOR *collector) {
  double phase = 0.0;

  srand(time(NULL));
  int code = sdcltr_interval_call(collector, proc, 1000*1000, &phase); // １秒周期で proc をコール

  sdcltr_exit(collector, code);
}
```

下記の proc は、sdcltr_interval_call()の内部から、定期的にコールされる関数です。
ここで 1 周期分のデータを収集し、SpeeDBee Hive に登録しています。
このサンプルでは、特に外部からデータを取得するわけではなく、内部的に計算した結果をデータとして登録しています。

```
static int proc(COLLECTOR *collector, void *data) {

  // １つ目のデータ（１回で 200 個登録）
  double v0[200];
  double *phase = (double*)data;
  for (int i=0; i<200; i++) {
    v0[i] = cos(*phase);
    *phase += M_PI/7;
  }
  sdcltr_insert_data_array(collector, 0, &v0, 200);
  HIVE_DEBUG("insert 200 data for sin wave: [%lf, .. %lf].¥n", v0[0], v0[199]);

  // ２つ目のデータ
  int v1 = rand() % 100;
  sdcltr_insert_data(collector, 1, &v1);
  HIVE_DEBUG("insert 1 data for random integer: [%d].¥n", v1);

  return SDCLTR_EXIT_SUCCESS;
}
```

4.カスタムコレクタ内で利用可能な型・構造体・関数

4.1. カスタムコレクタ用関数

4.1.1.sdcltr_insert_data() … データ登録

コレクタが収集したデータを SpeeDBee Hive に登録する関数

書式

```
bool sdcltr_insert_data(COLLECTOR *collector, int dataNo, void *data);
```

引数

No.	引数	説明
1	COLLECTOR *collector	コレクタインスタンス constructor/mainloop/destructor の関数引数に渡されたコレクタインスタンスをそのまま渡し

		てください。
2	int dataNo	データ番号 登録するデータの番号を指定します。コンストラクタで定義したデータ配列の要素番号に相当します。
3	void *data	データポインタ 登録するデータの先頭アドレスを指定します。

戻り値

false: 登録失敗

true: 登録成功

説明

コレクタが収集したデータを SpeeDBee Hive に登録するための関数です。BINARY 型以外のデータはこの関数で登録できます。

登録するデータのタイムスタンプはコンストラクタで設定したデータの頻度によって異なります。

- 不定期なデータの場合 … この関数を呼出した時点の現在時刻がタイムスタンプとなります
- 固定周期データの場合 … 初回のデータ登録ではこの関数を呼出した時点の現在時刻、二回目以降のデータ登録の場合、前回のタイムスタンプ+１周期分の時間となります

データ型が配列の場合、その配列要素の先頭アドレスを引数'data'に渡してください。

4.1.2.sdcltr_insert_data_ts() … タイムスタンプ指定のデータ登録

コレクタが収集したデータをタイムスタンプとともに SpeeDBee Hive に登録する関数

書式

```
bool sdcltr_insert_data_ts(COLLECTOR *collector, int dataNo,
                           void *data, sdcltr_time_t ts);
```

引数

No.	引数	説明
1	COLLECTOR *collector	コレクタインスタンス constructor/mainloop/destructor の関数引数に渡されたコレクタインスタンスをそのまま渡してください。
2	int dataNo	データ番号 登録するデータの番号を指定します。コンストラクタで定義したデータ配列の要素番号に相当します。

3	void *data	データポインタ 登録するデータの先頭アドレスを指定します。
4	sdcltr_time_t ts	タイムスタンプ 登録するデータのタイムスタンプを指定します。

戻り値

false: 登録失敗

true: 登録成功

説明

コレクタが収集したデータを SpeeDBee Hive に登録するための関数です。BINARY 型以外のデータはこの関数で登録できます。

登録するデータのタイムスタンプは引数 ts で指定した時刻となります。

データ型が配列の場合、その配列要素の先頭アドレスを引数'data'に渡してください。

4.1.3.sdcltr_insert_data_array() … 配列データの登録

コレクタが収集した複数個のデータを纏めて登録する関数

書式

```
int sdcltr_insert_data_array(COLLECTOR *collector, int dataNo, void *data, int length);
```

引数

No.	引数	説明
1	COLLECTOR *collector	コレクタインスタンス constructor/mainloop/destructor の関数引数に渡されたコレクタインスタンスをそのまま渡してください。
2	int dataNo	データ番号 登録するデータの番号を指定します。コンストラクタで定義したデータ配列の要素番号に相当します。
3	void *data	データポインタ 登録するデータの先頭アドレスを指定します。
4	int length	配列の要素数 登録する配列の要素数を指定します。

戻り値

説明

コレクタが収集したデータを SpeeDBee Hive に登録するための関数です。高サンプルレートの際の複数のデータをまとめて登録できます。

登録するデータのタイムスタンプは配列の先頭が登録された際の現在時刻になります。

4.1.4.sdcltr_reset_timestamp() … タイムスタンプのリセット

周期データにおいてタイムスタンプをリセットする関数

書式

```
void sdcltr_reset_timestamp(COLLECTOR *collector, int dataNo, sdcltr_time_t ts);
```

引数

No.	引数	説明
1	COLLECTOR *collector	コレクタインスタンス constructor/mainloop/destructor の関数引数に渡されたコレクタインスタンスをそのまま渡してください。
2	int dataNo	データ番号 登録するデータの番号を指定します。コンストラクタで定義したデータ配列の要素番号に相当します。
3	sdcltr_time_t ts	次のデータ登録時のタイムスタンプ 登録するデータのタイムスタンプを指定します。

戻り値

なし

説明

周期データにおいて、エラーなど何らかの要因によりデータを定期的に登録できなかった場合、SpeeDBee Hive 側で持つ時刻の連続情報にずれが生じます。
コレクタ内で定期的な登録ができないことを検出した場合に、この関数をコールして、次回以降のデータ登録から再度タイムスタンプを連続保持することを SpeeDBee Hive に通知してください。 次回以降のデータ登録は、引数'ts'に指定した時刻から開始となります。 'ts'が 0 の場合、本関数コール時点の時刻を開始時刻とします。

4.1.5.sdcltr_interval_call() … 定期処理関数

一定の周期で指定した関数を継続的に実行する関数

書式

```
int sdcltr_interval_call(COLLECTOR *collector, sdcltr_interval_proc_t interval_proc,
                         int interval_usec, void *data);
```

引数

No.	引数	説明
1	COLLECTOR *collector	コレクタインスタンス constructor/mainloop/destructor の関数引数に渡されたコレクタインスタンスをそのまま渡してください。
2	sdcltr_interval_proc_t interval_proc	定期処理関数 定期的に実行する処理を定義した関数のポインタを指定します。
3	int interval_usec	定期処理関数実行の周期 interval_proc 関数をコールする周期を usec 単位で指定します。
4	void *data	ユーザーデータ 任意のデータポインタを指定できます。interval_proc がコールされる際に引数に渡されます。

戻り値

SDCLTR_EXIT_SUCCESS:

interval_proc が一度もエラーとならずに、SpeeDBee Hive がコレクタを終了させた場合にこのコードが返却されます

その他:

interval_proc 内でエラーが発生した場合、internval_proc の戻り値がそのままこの関数の戻り値となります。

説明

一定の周期で何らかの処理を実行する場合に、この関数を使用してください。

多くのコレクタは、定期的なデータ収集を行うため、データ収集の処理を定義した関数のポインタを'interval_proc'引数に渡すことで基本的な動きを実現できます。
（カスタムコレクタのソースコードの説明参照）

4.1.6.sdcltr_exit() … コレクタの強制終了

コレクタのメイン処理を強制終了する関数

書式

```
void sdcltr_exit(COLLECTOR *collector, int code);
```

引数

No.	引数	説明
1	COLLECTOR *collector	コレクタインスタンス

		constructor/mainloop/destructor の関数引数に渡されたコレクタインスタンスをそのまま渡してください。
2	int code	終了コード コレクタの終了状態を指定します。 SDCLTR_EXIT_SUCCESS (0): 正常終了 SDCLTR_EXIT_FAILURE (1): 異常終了

戻り値

この関数は呼び出し元に戻りません。

説明

コレクタのメイン処理を強制終了する関数です。

4.1.7.sdcltr_running() … コレクタの実行可否状態取得

コレクタの実行可否状態取得関数

書式

```
bool sdcltr_running(COLLECTOR *collector);
```

引数

No.	引数	説明
1	COLLECTOR *collector	コレクタインスタンス constructor/mainloop/destructor の関数引数に渡されたコレクタインスタンスをそのまま渡してください。

戻り値

true: このコレクタのメイン処理を継続可

false: このコレクタのメイン処理を継続不可

説明

コレクタの実行は SpeeDBee Hive により制御されているため、何らかの理由によりコレクタに停止要求が出ることがあります。

コレクタは定期的にこの関数をコールして状態をチェックし、この関数が false を返した場合には速やかにメイン処理を終了しなければなりません。

4.1.1.HIVE_XXXX() … コレクタ用のログ出力

コレクタ用のログ出力マクロ関数

書式

```
void HIVE_TRACE(const char *format, ...);
void HIVE_DEBUG(const char *format, ...);
void HIVE_INFO(const char *format, ...);
void HIVE_WARNING(const char *format, ...);
void HIVE_ERROR(const char *format, ...);
```

引数

No.	引数	説明
1	const char *format	ログ出力書式指定文字列 ログに出力する書式指定文字列を指定します。 標準 C ライブラリの printf と同じ形式です。
2..	-	書式指定文字列の変換指示子により変換されるパラメータ

戻り値

なし

説明

SpeeDBee Hive 本体のログ出力に文字列を出力するためのマクロ関数です。

引数'format'、および、それに続く可変長引数は、標準 C ライブラリの printf()と同様に使用できます。

マクロは 5 種類用意されており、それぞれログレベルに対応しています。