

***SpeedDBee***

Version 6.2

SDTS-API リファレンス

株式会社ソルティスター

## - ● 改変履歴

DOC ver	日付	改変内容
1.0	2020/07/07	新規作成

# 目次

1. はじめに.....	5
2. 概要.....	5
3. 構成.....	5
4. 制限および注意事項.....	8
4.1. 制限事項.....	8
4.2. 注意事項.....	8
5. リアルタイム処理.....	9
5.1. ウィンドウ.....	9
5.2. ウィンドウタイプ.....	9
5.3. ウィンドウ処理.....	10
5.4. 集計処理.....	11
5.5. 通知処理.....	11
5.6. ウィンドウ定義.....	12
6. API 仕様.....	13
6.1. インターフェース初期化/終了.....	13
6.1.1. sd_init()... インターフェース初期化.....	13
6.1.2. sd_end()... インターフェース終了.....	13
6.2. DB 操作.....	14
6.2.1. sdts_open_db() ... DB 作成/オープン.....	14
6.2.2. sdts_close_db() ... DB クローズ.....	16
6.2.3. sdts_drop_db() ... DB 削除.....	16
6.2.4. sdts_delete_db() ... DB ストレージ再利用.....	17
6.3. DB 永続化同期操作.....	19
6.3.1. sdts_sync_db() ... DB 永続化同期操作.....	19
6.4. カラム（データ列）操作.....	23
6.4.1. sdts_create_col() ... カラム作成.....	23
6.4.2. sdts_create_col_batch() ... 複数カラム一括作成.....	26
6.4.3. sdts_get_col() ... カラム ID 取得.....	28
6.4.4. sdts_drop_col() ... カラム削除.....	28
6.4.5. sdts_activate_col() ... カラムアクティベート設定.....	29
6.4.6. sdts_set_smpl_rate() ... サンプリングレートの設定・変更.....	30
6.5. 基本データ操作.....	30
6.5.1. sdts_insert()... データ登録.....	30
6.5.2. sdts_get_col_data()... 単一カラムのデータ配列を取得.....	32
6.5.3. sdts_get_col_latest()... 単一カラムの最新値を取得.....	33
6.6. 集約検索.....	34

6.6.1. <code>sdtls_open_cur()</code> ...	集約用検索カーソルを作成	35
6.6.2. <code>sdtls_fetch_cur()</code> ...	カーソルの移動	51
6.6.3. <code>sdtls_get_cur_aggr()</code> ...	集約検索カーソルから各カラムの集約値を取得	51
6.6.4. <code>sdtls_extend_cur()</code> ...	検索カーソルの終了条件拡張	54
6.6.5. <code>sdtls_get_cur_dat()</code> ...	シリアルライズ検索カーソルからカラムの値を取得	57
6.6.6. <code>sdtls_close_cur()</code> ...	カーソルのクローズ	57
6.7. リアルタイム処理		58
6.7.1. <code>sdtls_set_win()</code> ...	リアルタイム処理用のウィンドウ定義の設定	58
6.7.2. <code>sdtls_unset_win()</code> ...	リアルタイム処理用の指定ウィンドウ定義の削除	61
6.7.3. <code>sdtls_drop_win()</code> ...	リアルタイム処理用のウィンドウ定義の全削除	61
6.8. カラムデータ取得 API		62
6.8.1. <code>spcf_get_col_dat()</code> ...	カラムデータ取得	62
6.8.2. <code>spcf_clean_col_dat()</code> ...	検索リソースの解放	65
6.9. 情報取得 API		66
6.9.1. <code>sdtls_get_db_info()</code> ...	DB 情報取得	66
6.9.2. <code>sdtls_free_db_info()</code> ...	DB 情報領域の解放	67
6.9.3. <code>sdtls_get_col_info()</code> ...	カラム情報の取得	68
6.9.4. <code>sdtls_free_col_info()</code> ...	カラム情報領域の解放	70
6.9.5. <code>sdtls_get_col_simp_info()</code> ...	簡易版カラム情報の取得	71
6.10. エラー API		72
6.10.1. <code>sd_get_err()</code> ...	エラーコードの取得	72
6.10.2. <code>sd_clear_err()</code> ...	エラーコードのクリア	72
7. エラー		73
7.1. エラー種別		73
7.2. エラーコード		74

## 1.はじめに

本書は SpeedBee における時系列 DB 用の SDTS-API 仕様を定義したものです。

## 2.概要

SpeedBee における時系列 DB は、インメモリ DB 機能を利用して高速にデータを時間に紐づけて登録します。DB はデータ登録可能なカラム（列）を複数保持し、カラムへのデータ登録は各カラム単位で、時間、値（もしくは値の配列）を設定し行います。登録されたデータは集約検索を利用して、同一タイムスタンプ上に集約して取得する事が可能です。

登録データは、リアルタイム分析機能を用い、ウィンドウ単位でのデータ分析およびイベント通知を送ることができます。

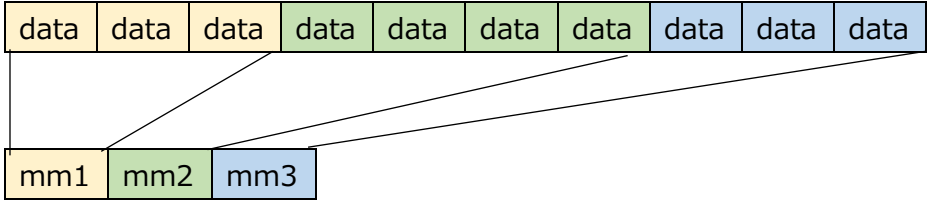
登録データは、指定によりファイル保存（永続化）することができます。また、永続化データはインメモリ DB 機能と同じ集約検索 API を用いて検索することができます。

## 3.構成

時系列 DB は、1つの DB 内に、設定（サンプリングレート、データサイズなど）の違う複数のカラム（データ列）を作成する事ができます。

カラムは下記のタイプが用意されています。用途に応じて使い分けてください。

カラムタイプ	特徴										
H（High）	<ul style="list-style-type: none"><li>● カラム作成時に、サンプリングレートとデータサイズを指定。（以降の変更不可）</li><li>● データは配列にて一括登録</li><li>● 登録時、初回はタイムスタンプを指定し、連続登録時はデータ配列のみ指定</li><li>● 内部制御が最もシンプルなためカラムタイプの中で最速</li><li>● データの取りこぼしがないアナログデータなどで利用</li><li>● 途中でタイムスタンプを設定された場合、メモリ上のデータはクリアされる</li></ul> <p>【データ管理イメージ】</p> <table><tr><td>data</td><td>data</td><td>data</td><td>data</td><td>data</td><td>data</td><td>data</td><td>data</td><td>data</td><td>data</td></tr></table> <p>データ管理は配列で、タイムスタンプを持っていない。データ取得時に初期設定タイムスタンプとサンプリングレートから計算でタイムスタンプを出力する。タイムスタンプを持たないため、メモリ使用効率はよい。</p>	data	data	data	data	data	data	data	data	data	data
data	data	data	data	data	data	data	data	data	data		

M (Middle)	<ul style="list-style-type: none"><li>● カラム作成時に、サンプリングレートとデータサイズを指定</li><li>● サンプリングレートは途中で変更可能</li><li>● データは配列にて一括登録</li><li>● 登録時、初回はタイムスタンプを指定し、連続登録時はデータ配列のみ指定</li><li>● 登録の都度タイムスタンプを指定した場合は、内部にて前回からの連続データか判断して格納（計算負荷は増加するため性能低下）</li><li>● 管理領域を別途参照しながらの登録になるため、Hと比較すると性能は劣る</li><li>● 計測中にサンプリングレートを変更する可能性のあるデータ収集に利用</li><li>● データの中断間隔を管理</li></ul> <p>【管理イメージ】</p> <p>データ配列</p>  <p>データ管理領域</p> <p>Hと同様データは配列で、データのみを管理し、タイムスタンプは持っていない。 M は、途中でサンプリングレートの変更やデータの登録中断の発生を対応するため、データの管理領域を持つ。 タイムスタンプは、データ取得時に各データ管理領域のタイムスタンプとサンプリングレートから計算でタイムスタンプを出力する。タイムスタンプを持たないため、メモリ使用効率はよい。</p>
------------	--

## L (Low)

- LOW タイプは、タイムスタンプとデータを常に一緒に格納するため、不定期なデータを扱う場合に向いている。H,M のようなサンプリングレート指定はない。また、タイムスタンプを格納するため、メモリリソースは H, M より必要になる。
- LOW タイプは、固定長データを扱う LOW FIX と可変長データを扱う LOW VAR の 2つのタイプがある。
- データ登録は、1 件ごとに行い、配列指定はない。

## 【管理イメージ】

time	data	time	data	time	data	time	data	time	data
------	------	------	------	------	------	------	------	------	------

L は、データとともにタイムスタンプをセットで持つ。(タイムスタンプは 8 バイト)

データは、固定長と可変長は、データフォーマットが異なる。固定長データの場合は、データのみが格納されるが、可変長データの場合、データの先頭にデータサイズ情報 (2 バイト) を付加し、格納する。

## 固定長データフォーマット

time(8)	data(n)
---------	---------

## 可変長データフォーマット

time(8)	size(2)	data(n)
---------	---------	---------

## 4.制限および注意事項

### 4.1.制限事項

- カラム数の最大

最大 2,147,483,647 個まで

※ただしシステム（メモリ容量）に依存する。また、カラムタイプ L-V は、Mutex 資源数に依存する。

- 登録データサイズの最大長

カラムタイプ	最大登録データサイズ (byte)
H (High)	64 byte
M (Middle)	256 byte
L-F (Low-Fix)	65535 Kbyte
L-V (Low-Var)	デフォルト 256 byte カラム作成パラメータ LV_MAX 指定で 65535 byte まで変更可能

- カーソル検索カラム数の最大

1024 個まで

### 4.2.注意事項

時系列 DB は、速度を優先し、API 内部の処理負荷を削減するために、極力、排他制御を行わないようにしているため、多くの API がスレッドセーフではありません。よって、下記の事を注意して利用する必要があります。

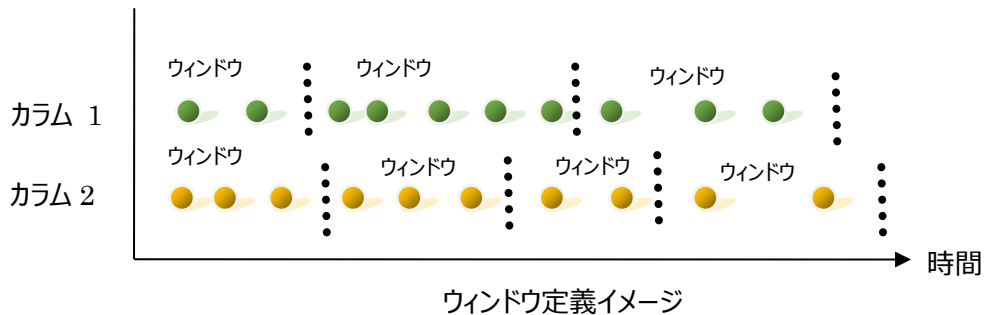
- IF 初期化用の API は、プロセスまたはカーネル起動時と終了時に行うこと。
- DB 作成、DB オープン、カラム作成など DB 管理操作は、データ登録・検索が行われる前に実行し、同時に実行しないよう制御すること。
- DB 削除、DB カラム削除操作の実行において、削除対象の DB、カラム、ファイルに他のプロセス・スレッドのアクセスがないことをアプリ側で確認、また、アクセスしないよう制御すること。
- 一意のカラムへのデータ登録は、1 タスク（スレッド）で行う。複数のタスク（スレッド）から同一カラムへ登録を行う場合には、アプリケーション側で同期をとり、並列実行しないようにすること。



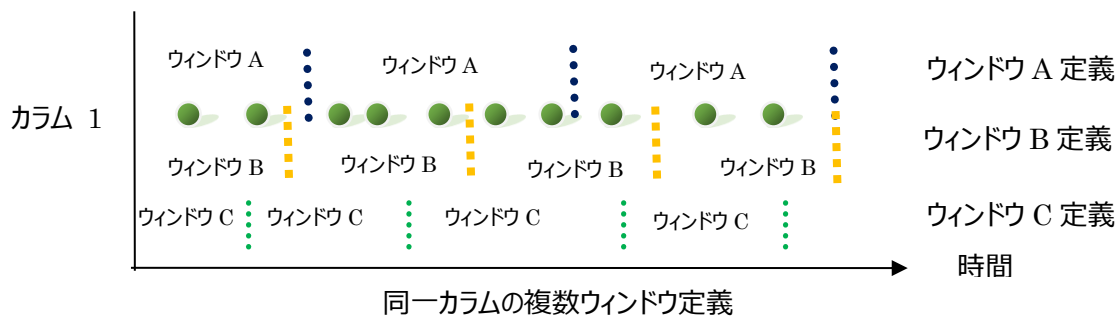
## 5.リアルタイム処理

### 5.1.ウィンドウ

ストリームデータをリアルタイム処理する上でデータを区切り処理を行います。この区切った間をウィンドウと呼び、集計、通知などをウィンドウ定義することで、各ウィンドウの境界で自動に処理が実行されます。



ウィンドウはカラムごとに定義し、1つのカラムのデータ群において複数のウィンドウ定義をすることができます。



### 5.2.ウィンドウタイプ

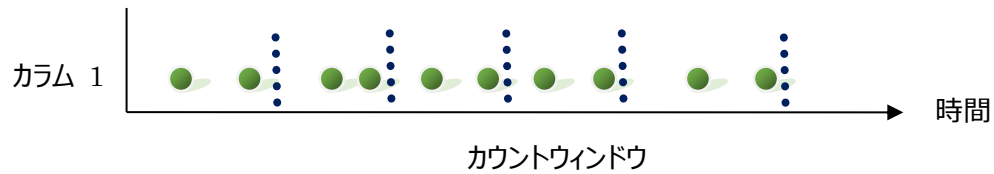
ウィンドウは以下のタイプをサポートします。

#	ウィンドウタイプ	説明
1	カウントウィンドウ	指定件数単位のウィンドウ
2	FFT ウィンドウ	FFT 分析のためのウィンドウ

次に各ウィンドウについて、図で説明します。

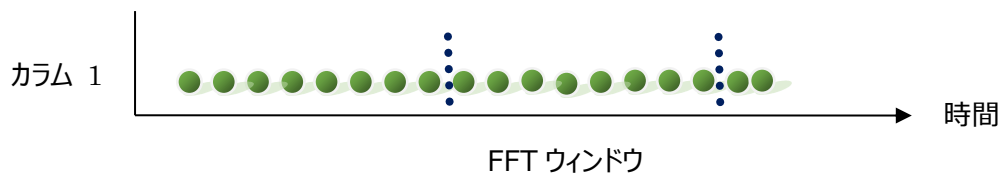
## 1) カウントウィンドウ

カウントウィンドウは、指定件数単位のウィンドウです。図では、2 件単位のウィンドウ定義になります。



## 2) FFT ウィンドウ

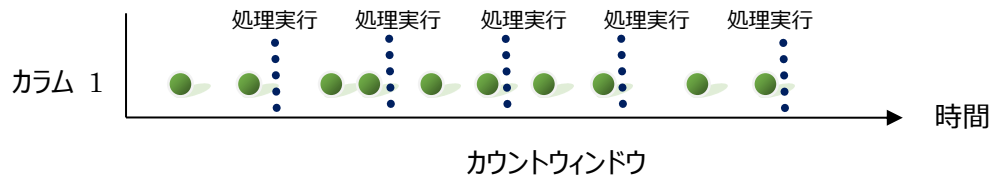
FFT 分析専用のウィンドウです。カウントウィンドウと同じで、FFT 処理件数単位を定義します。図では 8 件単位のウィンドウになります。



## 5.3.ウィンドウ処理

## 1) 実行タイミング

ウィンドウの境界で定義された処理を実行します。例えば 2 件単位のカウントウィンドウの場合、2 件目が入力された際に処理を実行します。



## 2) ウィンドウ処理

ウィンドウ処理は、以下の処理が可能です。

#	ウィンドウ処理	説明
1	集計処理	・基本統計量
2	通知処理	・コールバック関数による通知

次にそれぞれの処理について説明します。

## 5.4.集計処理

集計処理は、以下の基本統計量を取得することができます。

#	集計処理情報
1	最大値
2	最小値
3	総和
4	2乗和
5	平均
6	分散
7	標準偏差
8	不偏分散
9	標本不偏標準偏差
10	標準誤差
11	ウィンドウ開始時刻
12	ウィンドウ終了時刻
13	ウィンドウ内データ件数

## 5.5.通知処理

コールバック関数を登録することで通知することができます。

## 5.6. ウィンドウ定義

### 1) ウィンドウ定義

ウィンドウ定義は、ウィンドウタイプとウィンドウ処理を定義します。

以下の表は、ウィンドウタイプとサポートする機能です。

ウィンドウタイプ	集計処理	通知処理
カウントウィンドウ	○	○
FFT ウィンドウ	○	○

### 2) 複数ウィンドウの設定

1 カラムに対して複数のウィンドウ定義が可能です。違ったウィンドウタイプやウィンドウタイプ重複も可能です。

例えば、同一カラムに対して以下のように複数のウィンドウ定義が可能です

- ① 100 件単位のカウントウィンドウと 200 件単位のカウントウィンドウ
- ② 100 件単位のカウントウィンドウと 512 件単位の FFT ウィンドウ

### 3) ウィンドウ定義 API

ウィンドウ定義の操作は以下の API を使用します。

	API 名	説明
1	sdts_set_win	ウィンドウ定義の設定
2	sdts_unset_win	指定ウィンドウ定義の削除
3	sdts_drop_win	ウィンドウ定義の全削除

※詳細は、それぞれの API を参照してください。

## 6.API 仕様

### 6.1.インターフェース初期化/終了

SDTS-API を使用するにあたり、アプリケーション開始／終了時にインターフェースの初期化及び終了処理を行う必要があります。

プロセスアプリケーション	プロセスの起動、終了にて実行
カーネルアプリケーション	カーネル起動時と終了時に実行

#### 6.1.1.sd\_init()… インターフェース初期化

##### 【書式】

```
#include <speedbee.h>
int sd_init(char *env)
```

##### 【引数】

char* env	…	環境パラメータ
-----------	---	---------

##### 【戻り値】

0	:	正常終了
-1	:	異常終了

##### 【説明】

SDTS-API を利用する環境の初期化を行います。(SpeedBee 共通)

環境パラメータは、“パラメータ名=値”の形態で指定します。複数指定する場合は、;(セミコロン)に続けて指定してください。

#### 6.1.2.sd\_end()… インターフェース終了

##### 【書式】

```
#include <speedbee.h>
void sd_end(void)
```

##### 【説明】

SDTS-API のインターフェース終了処理を行います。

## 6.2. DB 操作

### 6.2.1.sdts\_open\_db() ... DB 作成/オープン

#### 【書式】

```
#include <speedbee.h>
sdtsdb_t sdts_open_db(char *dbpar)
```

#### 【引数】

dbpar	...	DB パラメータ
-------	-----	----------

#### 【戻り値】

NULL 以外	:	正常終了 (DB 構造体ポインタを返却)
NULL	:	異常終了

#### 【説明】

データベースをオープンします。DB パラメータ “DB\_PATH”が指定された場合、ストレージへの永続化を行います。指定されたパスに DB ディレクトリが、存在しない場合はそのディレクトリに DB を新規に作成します。存在する場合、データベースをオープンします。“DB\_PATH”を指定しない場合は、永続化は行わず、メモリ DB 機能のみ使用となります。

DB パラメータは、“パラメータ名=値”のフォーマットで指定します。複数指定する場合は、;(セミコロン)に続けて指定してください。

パラメータ名	デフォルト	※1	説明
DB_PATH	なし	○	DB ディレクトリパスの指定(絶対パス)。 指定したディレクトリが存在しない場合、新規に DB を作成します。 存在する場合は、DB をオープンする。 この DB_PATH を指定しない場合、メモリ DB のみの使用となり、永続化は行わない。
COL_KEY_ID_SIZE	説明参照	×	カラム名称としてバイナリキーを使用する場合にバイナリキーサイズ(単位バイト)を指定。バイナリキーサイズの最大は 16 バイト。 COL_KEY_ID_SIZE を指定しない場合は、カラム名称は文字列キーになり、指定可能な最大文字数は 15 文字まで。 例えば、以下のような構造体をキーとして指定可能。 <pre>struct {     ui16_t    did;        // ID 1     ui16_t    cid;        // ID 2     ui32_t    sid;        // ID 3 };</pre> アプリ側で定義済みの構造体を利用することで、カラム名の定義が不要となる。

COL_SPEC_ID	No	×	カラム名称として任意のカラム ID を指定する場合、"Yes"を指定。 Yes 指定した場合のカラム名称のデータ型は 4 バイト整数。 COL_SPEC_ID と COL_KEY_ID_SIZE の両方を指定した場合、COL_SPEC_ID が優先され、内部でキーサイズを 4 バイトに指定する。
SYNC_DIR_UTC	Yes	×	DB を永続化した場合にデータを日時ごとにディレクトリに管理するがその際の日時ディレクトリ名として UTC 時間かローカル時間(日本では JST)のどちらを使うかの指定 (Yes/No) デフォルトは"Yes"で UTC、"No"はローカル時間
SYNC_DAT_COMP	No	×	DB を永続化した場合にデータを圧縮するかどうかの指定 (Yes/No) "Yes"は圧縮、"No"は圧縮しない。 ※この機能は DB ライブラリの圧縮機能が使える場合のみ有効
SYNC_WRBSZ	2MB	○	DB を永続化した場合にファイル書き込みに使用する書き込みバッファサイズを指定。単位はバイトサイズ サイズは、1 ファイル分のデータを 1 度を書くことで書き込み性能向上を行っているため、1 ファイルの最大サイズを指定する
SYNC_PERIOD_H SYNC_PERIOD_M SYNC_PERIOD_L	2 5 30	×	DB を永続化する場合の 1 データファイルに格納するデータ期間を指定する。単位は秒。 カラムタイプごとの指定でパラメータ名の最後の_(H M L)はカラムタイプを示す。詳細は、「6.3.1.sdts_sync_db () ... DB 永続化同期操作【同期関連 DB パラメータ】」を参照
SYNC_LAG_H SYNC_LAG_M SYNC_LAG_L	5 5 5	○	同期 API の実行(sdts_sync_db)で現在時間から指定秒数の前のデータを書き込む対象とする指定。単位は秒。 カラムタイプごとの指定でパラメータ名の最後の_(H M L)はカラムタイプを示す。詳細は、「6.3.1.sdts_sync_db () ... DB 永続化同期操作【同期関連 DB パラメータ】」を参照
SYNC_MAXF_H SYNC_MAXF_M SYNC_MAXF_L	1 1 1	○	1 回の同期 API の実行(sdts_sync_db)で最大作成データファイル数を指定する。 カラムタイプごとの指定でパラメータ名の最後の_(H M L)はカラムタイプを示す。詳細は、「6.3.1.sdts_sync_db () ... DB 永続化同期操作【同期関連 DB パラメータ】」を参照
READ_MAX_CACHE	2	○	データファイルキャッシュ数の指定 ファイルからデータを読み込む場合の最大ファイルキャッシュ数を指定する。キャッシュは、同一 DB ディスクリプタ内で共有する。
READ_MAX_SEARC H_CACHE	5	○	ファイル検索結果件数の指定 ファイル検索した結果を保管し、次回以降、ファイル検索条件が同じであれば、キャッシュされた結果から値を返す。

※1 ストレージ DB のオープン(すでに存在する DB に対してのオープン)時に設定可能なパラメータは ○、設定できないものは×で示します。

### 6.2.2.sdts\_close\_db () … DB クローズ

**【書式】**

```
#include <speedbee.h>
int sdts_close_db(sdtsdb_t db);
```

**【引数】**

sdtsdb_t db	…	DB 構造体ポインタ
-------------	---	------------

**【戻り値】**

0	:	正常終了
-1	:	異常終了

**【説明】**

DB をクローズします。

DB を永続化している場合、同期されていないデータはすべて同期します。また、インメモリのみ の DB の場合、全データは破棄されます。

### 6.2.3.sdts\_drop\_db () … DB 削除

**【書式】**

```
#include <speedbee.h>
int sdts_drop_db(char* dbpar);
```

**【引数】**

char* dbpar	…	DB パラメータ 利用可能な DB パラメータは、「DB_PATH」のみ、それ以外は無視される。
-------------	---	---

**【戻り値】**

0	:	正常終了
-1	:	異常終了

**【説明】**

ストレージ上の DB ディレクトリを削除します。

API の実行により DB ディレクトリ以下を削除しますが、DB ディレクトリ中にユーザーが作成したディレクトリおよびファイルが存在した場合、これらは削除しません。

**【留意事項】**

**必ず、DB クローズ、および他のプロセスおよびタスクが DB にアクセスしていないことを確認してから実行してください。**（これはデータベースアクセスを高速化するため、内部において DB ロック処理を行っておりません。この API を実行する際は、DB アクセスの排他制御はアプリ側で考慮してください。）



### 6.2.4.sdts\_delete\_db () … DB ストレージ再利用

#### 【書式】

```
#include <speedbee.h>
int sdts_delete_db (sdtsdb_t db, int min);
```

#### 【引数】

sdtsdb_t db	…	DB 構造体ポインタ
int min	…	保存時間（分） 指定例として 10 分間保存は、10 を指定、1 時間は 60 を指定、 1 日は、1440 を指定する。 また、0 は、指定できない。

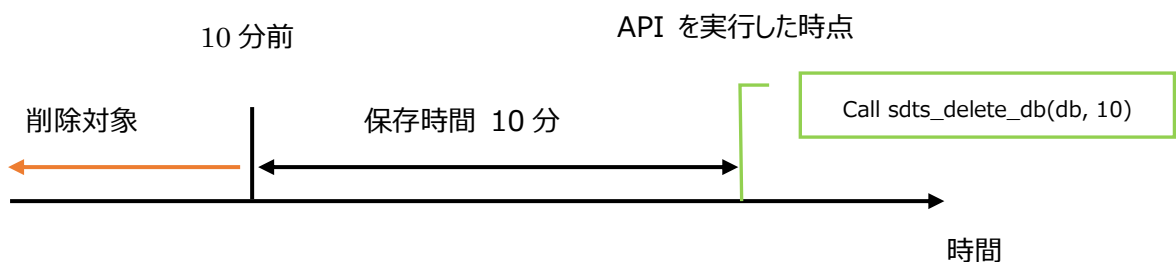
#### 【戻り値】

0	:	正常終了
-1	:	異常終了

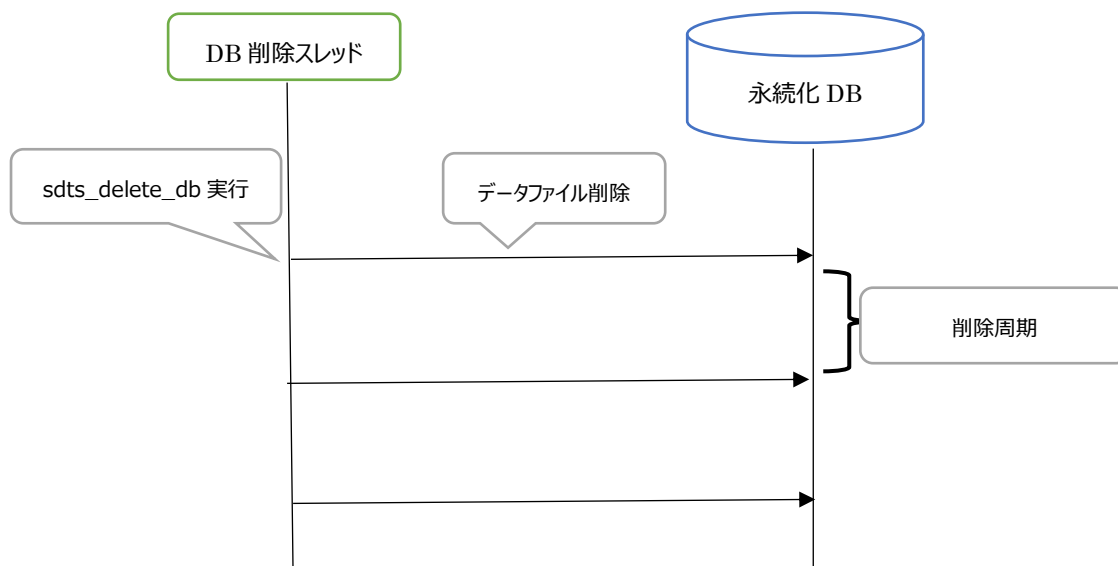
#### 【説明】

永続化した DB に対して、限られたストレージリソースの再利用するためにコールした時点から引数に指定された保存時間（分）より前のデータファイルを削除します。

以下の図で例えば API の保存時間 10 と指定して実行すると API を実行した時点から 10 分前までを保存期間とします。それ以前が削除対象となります。



この API を利用方法として、同期操作の API と同様に専用のスレッドを用意し、定期的に行います。実行周期は、システムリソース、システム負荷を考慮し、適切な周期で実行してください。



API の実行により DB ディレクトリ以下の指定ファイルおよびディレクトリを削除しますが、対象 DB ディレクトリ中にユーザーが作成したディレクトリおよびファイルが存在した場合、これらは削除しません。

また、API は、データファイルのみを削除し、データベース自体の削除は行わないため、DB 全体を削除する場合は API `sdts_drop_db()` を使用してください。

**【留意事項】**

他のプロセス・スレッドが対象 DB にアクセスしている場合でも対象の DB ファイルの削除を行います。API の実行の前提として削除対象になっているファイルを他のプロセス・スレッドが参照していないことになるため、実行前に他が DB にアクセスしていないことを確認してください。（これはデータベースアクセスを高速化するため、内部において DB ロック処理を行っておりません。そのため、この API を実行する際は、上記をアプリ側で考慮してください。）

## 6.3.DB 永続化同期操作

### 6.3.1.sdts\_sync\_db () … DB 永続化同期操作

#### 【書式】

```
#include <speedbee.h>
int sdts_sync_db (sdtsdb_t db, int cmd);
```

#### 【引数】

sdtsdb_t db	…	DB 構造体ポインタ
int cmd	…	同期操作指定 SDTS_SYNC_CMD_REG 通常同期操作 SDTS_SYNC_CMD_FORCE 強制同期操作

#### 【戻り値】

0	:	正常終了
-1	:	異常終了

#### 【マクロ】

```
#define sdts_sync(a)      sdts_sync_db((a),SDTS_SYNC_CMD_REG)
#define sdts_sync_force(a) sdts_sync_db((a),SDTS_SYNC_CMD_FORCE)
```

#### 【説明】

メモリ DB からストレージ上の DB に対してデータ同期(保存)を行います。

SDTS\_SYNC\_CMD\_REG(マクロ: sdts\_sync)のデータ同期の方法は、API 利用側で同期スレッドを作成し、周期的にこの API をコールすることによりデータ保存を行います。

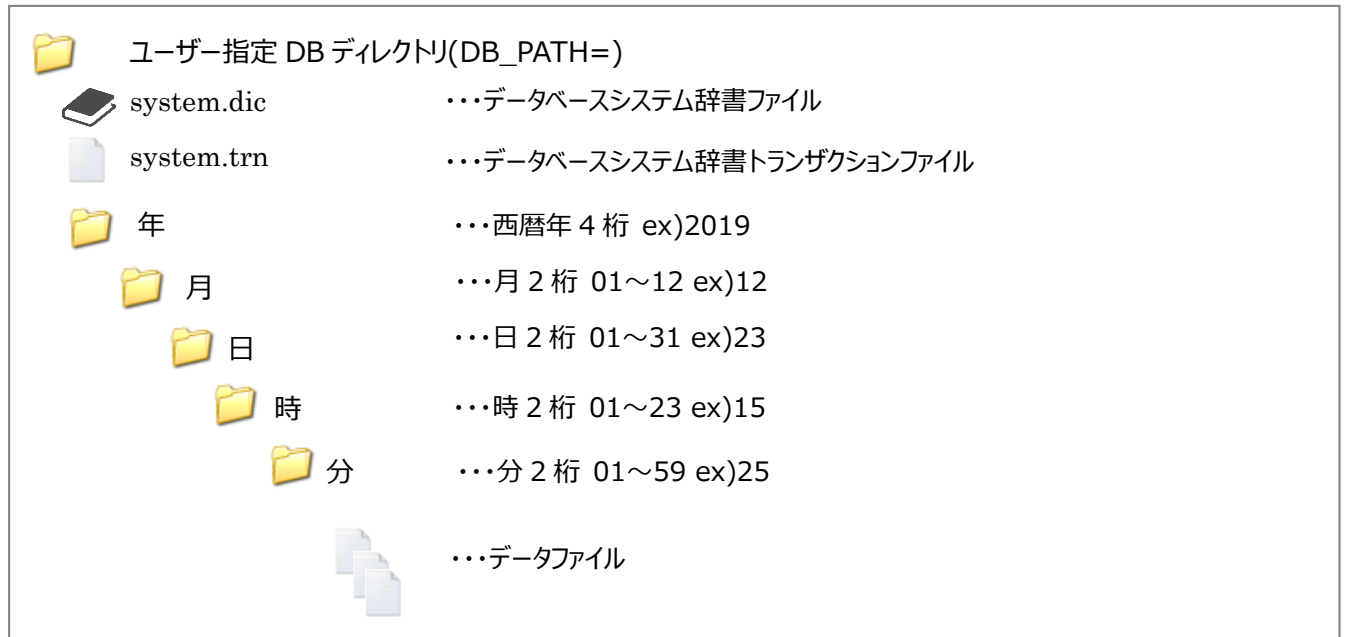
SDTS\_SYNC\_CMD\_FORCE(マクロ:sdts\_sync\_force)のデータ同期の方法は、ユーザーが任意のタイミングでメモリ上のデータをストレージへ強制同期を実行します。

#### 【留意事項】

・SDTS\_SYNC\_CMD\_FORCE(マクロ:sdts\_sync\_force)を実行する場合、他スレッドからの同一 DB に対する登録や同期操作がないことが前提となります。

## 【永続化 DB のディレクトリ・ファイル構成】

永続化 DB のデータファイルは、登録データの日時のディレクトリに格納され、構成は、以下になります。



データファイル名の定義は以下になります。

データ時間(yyyymmddHHMMSS)+データファイル番号(01~99)+'\_'+間隔秒数(01~59) + ' \_'  
+ カラムタイプ(H, M, S)+ 拡張子 ".sdt"

永続化データは、カラムタイプごとに対象データのタイムスタンプが同じ範囲データをまとめ対象データファイルに格納します。対象時間にデータがない場合は、ファイルは作成されません。

登録データは、データ登録時間の年月日時分ディレクトリの下にデータファイルに格納されます。

例えば以下のデータの場合

- ・DB 名 db1
- ・カラムタイプ M,
- ・登録タイムスタンプ 2019 年 12 月 23 日 16:20.39
- ・DB パラメータ SYNC\_PERIOD\_M 10 秒

格納するデータファイルパスは、以下になります。

db1/2019/12/23/16/20/2019122316203001\_10\_M.sdt

## 【同期関連 DB パラメータ】

## 1) SYNC\_PERIOD\_H, SYNC\_PERIOD\_M, SYNC\_PERIOD\_L

データファイルへ登録する期間（間隔）を指定します。単位は秒。

カラムタイプごとの指定でパラメータ名の最後の\_(H|M|L)はカラムタイプを示します。

例えば、SYNC\_PERIOD\_M = 1 とした場合、1 秒ごとのデータファイルが作成されます。

指定する値は、60 を割り切れる数で、60 を超える値は指定できません。

指定可能な値：1,2,3,4,5,6,10,15,20,30,60

## 2) SYNC\_LAG\_H, SYNC\_LAG\_M, SYNC\_LAG\_L

同期 API の実行時の時間から指定秒数の前のデータを同期対象とする指定になります。単位は秒。

カラムタイプごとの指定でパラメータ名の最後の\_(H|M|L)はカラムタイプを示します。

例えば 5 秒とした場合、API の実行時の時間から 5 秒前のデータまでが同期対象データとなり、

直前のデータは、同期対象データとはなりません。これを行う理由は、カラムによって登録タイミングがバラバラであるため、同期するデータを揃えるためです。

## 3) SYNC\_MAXF\_H, SYNC\_MAXF\_M, SYNC\_MAXF\_L

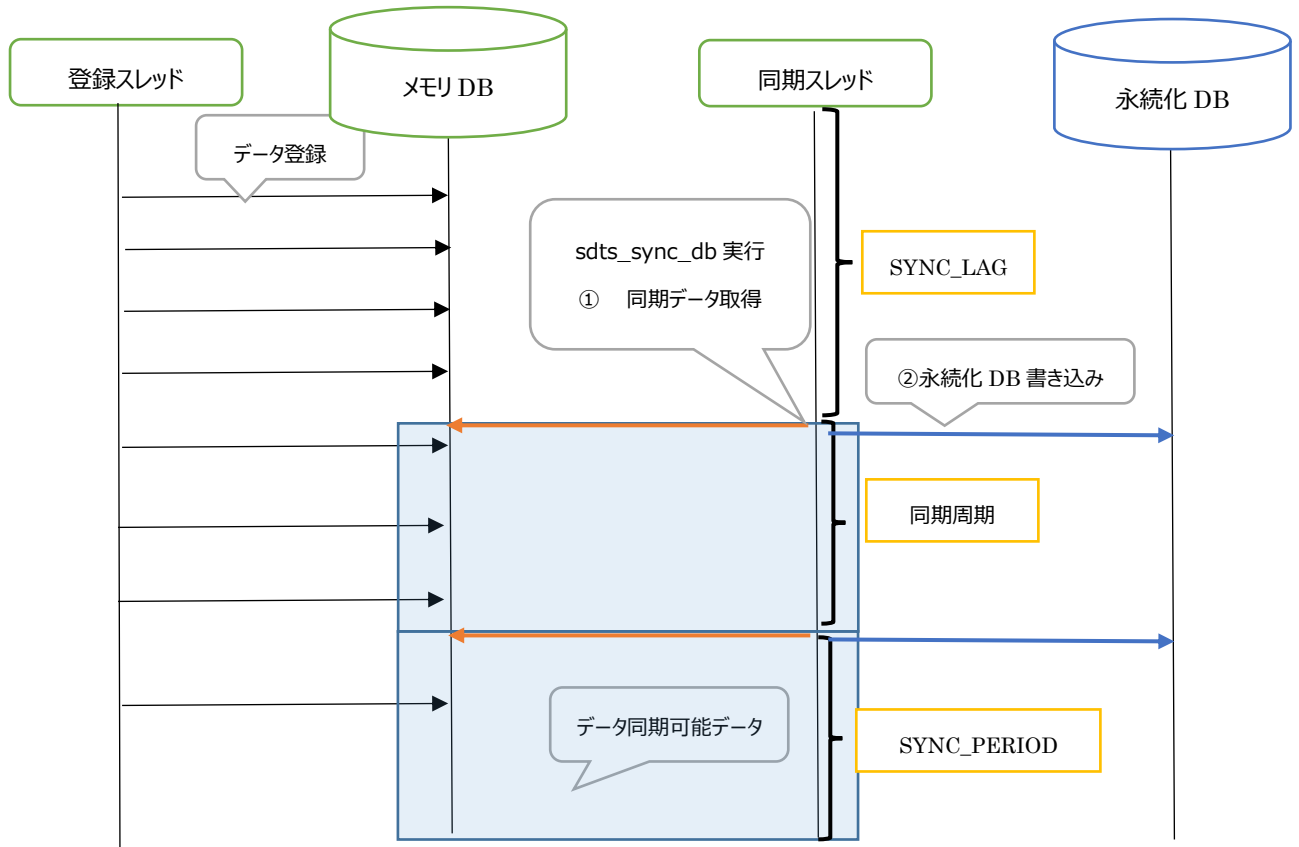
1 回の同期 API の実行で作成する最大データファイル数を指定します。カラムタイプごとの指定でパラメータ名の最後の\_(H|M|L)はカラムタイプを示します。

例えばすべてのタイプに 2 と指定した場合、1 回の同期実行でそれぞれのタイプで最大で 2 ファイル作成を許可します。1 回の実行で作成数を制限する理由は、同期データがメモリにたまっていた場合に、複数のファイルを作成しようとしませんが、ストレージへの書き込みは最も遅い処理であるため、次回の周期的な同期実行に間に合わなくなる可能性があり、そのため、作成数を制限し、同期の調整を行います。

## 【データ同期実行スレッド】

本製品は、DB ライブラリであり、サーバー、サービスプロセスまたはスレッドを持ちません。そのため、ストレージに永続化するために利用者側でデータ同期 API を実行します。

データ同期 API の実行は、以下のように登録スレッドと別のスレッドにし、周期的に実行してください。



## 6.4.カラム（データ列）操作

### 6.4.1.sdts\_create\_col () … カラム作成

#### 【書式】

```
#include <speedbee.h>
sdtsid_t sdts_create_col(sdtsdb_t db, sdid_t cname, char *colpar)
```

#### 【引数】

sdtsdb_t db	…	DB 構造体ポインタ
sdid_t cname	…	カラム名（またはカラム識別 ID）
char *colpar	…	カラムパラメータ

#### 【戻り値】

1 以上	:	正常終了（カラム ID を返却）
-1	:	異常終了

#### 【説明】

カラム（データ列）を作成します。カラムパラメータは、“パラメータ名=値”の形態で指定します。複数指定する場合は、;(セミコロン)に続けて指定してください。

パラメータ名	○必須 △任意 ×不要	単位 デフォル ト	説明
COL_TYPE	H ○ M ○ L ○		カラムタイプを指定 'H' 'M' 'L' のいずれか ex) H を指定 COL_TYPE='H';
COL_ACT	H △ M △ L △	'A'	カラムアクティベート指定 'A' アクティブ'I' 非アクティブのいずれか 指定がない場合のデフォルトは、アクティブ ex) 非アクティブ COL_ACT='I' ※カラムのアクティベートについて次ページで補足します。
DATA_SIZE	H ○ M ○ L 固 ○ L 可 ×	byte	1 データのサイズを指定 COL_TYPE=L で可変長の場合は指定しない。 ex) データサイズ 4 バイト DATA_SIZE=4;
SMPL_RATE	H ○ M △ L ×	Hz	サンプリングレートを指定 COL_TYPE=M の場合、必須ではないが初期サンプリングレートの指定が可能。 ex)サンプリングレート 10kHz SMPL_RATE=10000 少数点以下の指定は、以下のように指定する。 サンプリングレート 31.25Hz SMPL_RATE=31.25
SAVE_COUNT	H ○	件数	メモリ保存データ件数を指定

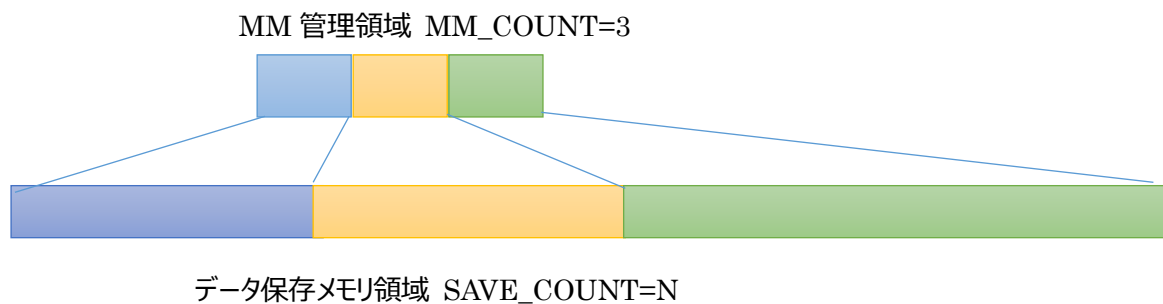
	M ○ L ○		ex)メモリ保存件数 10000 SAVE_COUNT=10000;
MM_COUNT	H X M △ L X	個数 100	<p>COL_TYPE=M での管理領域の個数を指定。</p> <p>管理領域は、登録途中でサンプリングレートが変わった場合やデータ取得時のエラーによって空白時間が存在した場合において、データ位置情報などの管理情報を保存する領域のことで、SAVE_COUNT で指定した保存件数内での管理領域の上限である。</p> <p>例えば、MM_COUNT=10 とした場合、SAVE_COUNT 数内において 10 回までサンプリングレートを変更し、登録が可能である。</p> <p>これは、SAVE_COUNT 数内での制限であり、MM_COUNT 数内であれば、サンプリングレートの変更登録数に制限はない。</p> <p>管理領域は、指定個数内であれば再利用されるため、SAVE_COUNT 内の上限を越えなければ、問題はないが、この上限を上回った場合 ERR_USR_SDTS_MM_LIMIT_COUNT(182208) エラーとなる。</p> <p>ex) 管理領域を 10 件 MM_COUNT=10; ※MM_COUNT について次ページで補足。</p>
WR_DATA	H △ M △ L △	No	<p>ストレージ上にデータを永続化するか指定 (Yes/No)</p> <p>デフォルトは No であり、保存する場合は Yes を指定する。</p> <p>ex) 登録データを永続化する WR_DATA='No';</p>
LV_MAX	H X M X L 固 X L 可 △	byte 256	<p>LO 可変長タイプのデータ最大長の指定</p> <p>LO 可変長タイプの登録可能なデータ最大長のデフォルトは 256 バイトである。それを超えるデータを登録する場合は、このパラメータを指定する。</p> <p>ex) 最大データ長を 512 byte にする LV_MAX=512</p> <p>使用例として、文字列のデータでデータ長が固定でないものは LO 可変長型を使用して管理する。</p> <p>(その他のカラムタイプでは、データ長は、カラム生成時に指定したサイズで固定)</p> <p>最大値を設定することで、最大値以上のデータが登録されることを防止できる。仕様上の上限は、65535byte。</p>



## 【MM\_COUNT の指定について】

メモリ上では、カラムタイプ Middle のデータは、MM 管理領域とデータ保存領域で管理されています。  
 データの保存領域に格納されたデータの塊※ 1 ごとに MM 管理領域を 1 つ保持します。  
 ※ 1 データの塊とは、サンプリングレートが異なるデータ、およびエラーなどによってサンプリングレートの連続性が途切れたデータ単位のこと

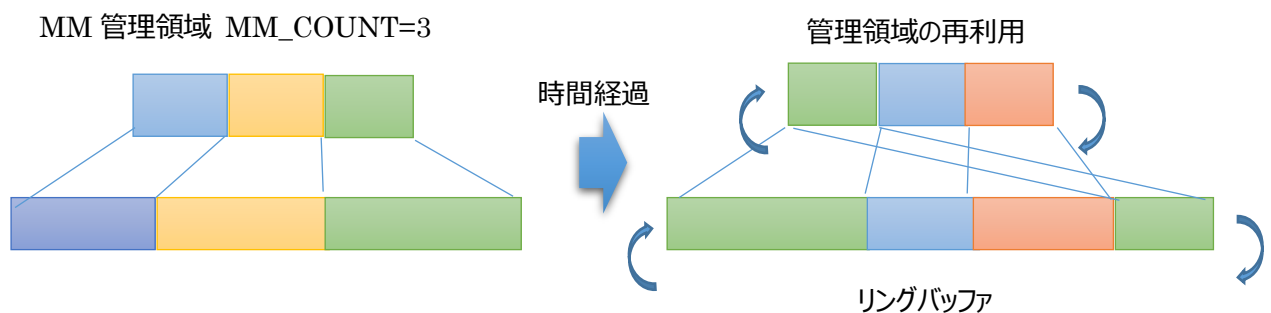
管理領域では、サンプリングレート、データ開始時間、データ個数、データ保存領域のインデックスなどの管理情報を持ちます。(管理領域の 1 つのサイズは約 50 バイト)  
 この管理により、データごとのタイムスタンプが不要であり、メモリリソースを効率的に利用します。



例えば MM\_COUNT=3 を指定した場合(上図)、3 つの管理領域を保持し、データ保存メモリ領域では 3 つのデータの塊を管理することができます。  
 MM\_COUNT を、上限を超えるデータの塊をデータ保存メモリ領域に登録することはできません。超えた場合は、ERR\_USR\_SDTS\_MM\_LIMIT\_COUNT(182208)エラーとなります。

MM\_COUNT 数内であれば、時間が経過したとしても管理領域はリングバッファにより再利用されるため、常に指定数分は管理が可能です。

(下の右の図は、データ保存メモリ領域が 4 つに分かれてはいますが、緑のデータは後方から先頭に続いているので、データの塊としては 3 つになります。)



### 【カラムのアクティブ化と非アクティブ化】

カラムは、アクティブ、非アクティブの 2 つの状態を持ちます。

- ・アクティブ状態は、メモリ上にデータ領域を持ち、カラムデータの操作（登録、検索）が可能です。
- ・非アクティブ状態は、メモリにデータ領域を持たず、カラム定義のみ保持している状態でカラムデータの操作はできません。

アクティブと非アクティブがあることで、アクティブ状態の使用していないカラムが存在する場合、非アクティブにすることでメモリリソースを節約し、カラム定義のみであるため、将来的に作成せずに、アクティブにすることで利用することができます。

アクティブ、非アクティブは、カラム作成 API `sdts_create_col()` およびカラムアクティベート API `sdts_activate_col()` で指定することができます。

### 6.4.2.sdts\_create\_col\_bat () … 複数カラム一括作成

#### 【書式】

```
#include <speedbee.h>
int sdts_create_col_bat(sdtsdb_t db, sdtsbrcol_t *bp, int bcnt)
```

#### 【引数】

sdtsdb_t db	...	DB 構造体ポインタ
sdtsbrcol_t *bt	...	カラム作成パラメータ配列
int bcnt	...	作成カラム数

#### 【戻り値】

0	:	正常終了
-1	:	異常終了

#### 【説明】

複数カラムを一括作成します。

大量カラムを作成する場合、1 カラムごと `sdts_create_col` で作成するより、この API を使用し、内部において一括してカラムを作成することで作成速度が向上します。特に、ストレージ DB を使用する場合において、ストレージアクセスが遅いケースに有効です。

パラメータの第 2 引数には、カラム作成パラメータの配列を用意し、各カラムの作成のパラメータをセットします。

第 3 引数には、カラム作成数（作成パラメータ配列の要素数）をセットします。

#### 第 2 引数のカラム作成パラメータ

typedef struct {	名前	説明
sdid_t	name	[in] 名前ポインタ sdts_create_col の第 2 引数と同じ。 ※sdts_create_col の cname を参照
cha	*colpar;	[in] カラム作成パラメータ sdts_create_col の第 3 引数と同じ。 ※sdts_create_col の colpar を参照

sdtscid_t	rscid	[OUT]作成後のカラム ID sdt_create_col の戻り値のカラム ID をセット
} sdtsbrcol_t;		

## 【カラム作成定義例】

```
// 10 カラムの一括作成
#define COL_CNT      10
static sdtsbrcol_t  brcol[COL_CNT] = {
{"c1", "COL_TYPE=M;DATA_SIZE=4;SMPL_RATE=100;SAVE_COUNT=1000", 0},
{"c2", "COL_TYPE=M;DATA_SIZE=4;SMPL_RATE=200;SAVE_COUNT=2000", 0},
{"c3", "COL_TYPE=M;DATA_SIZE=4;SMPL_RATE=300;SAVE_COUNT=3000", 0},
{"c4", "COL_TYPE=M;DATA_SIZE=4;SMPL_RATE=400;SAVE_COUNT=4000", 0},
{"c5", "COL_TYPE=M;DATA_SIZE=4;SMPL_RATE=500;SAVE_COUNT=5000", 0},
{"c6", "COL_TYPE=M;DATA_SIZE=4;SMPL_RATE=600;SAVE_COUNT=6000", 0},
{"c7", "COL_TYPE=M;DATA_SIZE=4;SMPL_RATE=700;SAVE_COUNT=7000", 0},
{"c8", "COL_TYPE=M;DATA_SIZE=4;SMPL_RATE=800;SAVE_COUNT=8000", 0},
{"c9", "COL_TYPE=M;DATA_SIZE=4;SMPL_RATE=900;SAVE_COUNT=9000", 0},
{"c10", "COL_TYPE=M;DATA_SIZE=4;SMPL_RATE=1000;SAVE_COUNT=10000", 0}
};
// 上の例ではカラム名を文字列としていますが、バイナリ指定する場合は、パラメータ構造体の name にバイナリのポインタを指定します。
// 例) "COL_KEY_ID_SIZE=4"の場合
// uint32_t id[COL_CNT] = {1,2,3,4,5,6,7,8,9,10};
// for (i = 0; i < COL_CNT;i++) brcol.name = (sdid_t)&id[i];
// 上記の設定で API 呼び出す場合は以下になります。
sdt_create_col_bat(db, brcol, COL_CNT)
```

### 6.4.3.sdts\_get\_col() … カラム ID 取得

**【書式】**

```
#include <speedbee.h>
sdtsid_t sdts_get_col(sdtsdb_t db, sdid_t cname);
```

**【引数】**

sdtsdb_t db	…	DB 構造体ポインタ
sdid_t cname	…	カラム名（またはカラム識別 ID）

**【戻り値】**

1 以上	:	カラム ID を返却
-1	:	指定したカラム名(またはカラム識別 ID) は存在しない

**【説明】**

カラム ID を取得します。カラム ID はデータ登録、検索などで利用します。登録の都度、カラム ID を取得するのではなく、事前に操作するカラム ID を取得し、システム全体で共有する事を推奨します。

### 6.4.4.sdts\_drop\_col () … カラム削除

**【書式】**

```
#include <speedbee.h>
sdtsid_t sdts_drop_col(sdtsdb_t db, sdid_t cname)
```

**【引数】**

sdtsdb_t db	…	DB 構造体ポインタ
sdid_t cname	…	カラム名（またはカラム識別 ID） カラム名を文字列で指定するか、バイナリの ID で指定するかは DB 作成時に決定します。

**【戻り値】**

0	:	正常終了
-1	:	異常終了

**【説明】**

カラムを削除します。指定されたメモリ DB 上のカラムの全データは削除されます。

カラムデータをストレージ保存した場合（WR\_DATA=y のケース）は、保存データは削除せずに保持したままとなり、削除対象のカラムは、非アクティブ状態として管理します。（維持する理由は、保存したデータのカラム情報が失われるため）また、保存データを取得する場合は、カラムをアクティブ状態に変更（sdts\_activate\_col）することで、取得が可能となります。

### 6.4.5.sdts\_activate\_col () ... カラムアクティベート設定

#### 【書式】

```
#include <speedbee.h>
int sdts_activate_col(sdtsdb_t db, sdid_t cname, bool act)
```

#### 【引数】

sdtsdb_t db	...	DB 構造体ポインタ
sdid_t cname	...	カラム名（またはカラム識別 ID） カラム名を文字列で指定するか、バイナリの ID で指定するかは DB 作成時に決定します。
bool act	...	アクティブ・非アクティブ指定 true(1) アクティブ false(0) 非アクティブ

#### 【戻り値】

0	:	正常終了
-1	:	異常終了

#### 【説明】

カラムをアクティブ・非アクティブに状態に変更します。ストレージ DB に保存をしている場合、設定したアクティブ・非アクティブ状態は永続化されます。

非アクティブ状態は、カラム属性は保持されますが、カラムメモリ DB 領域は解放され、カラムデータの登録および検索（ファイル検索を含む）はできません。

#### 6.4.6.sdts\_set\_smpl\_rate () … サンプルングレートの設定・変更

##### 【書式】

```
#include <speedbee.h>
int sdts_set_smpl_rate(sdtsdb_t db, sdid_t cname, double rate)
```

##### 【引数】

sdtsdb_t db	…	DB 構造体ポインタ
sdid_t cname	…	カラム名（またはカラム識別 ID） カラム名を文字列で指定するか、バイナリの ID で指定するかは DB 作成時に決定します。
double rate	…	サンプルングレート（単位：Hz） ※小数点以下の指定可能 ex) 31.25

##### 【戻り値】

0	:	正常終了
-1	:	異常終了

##### 【説明】

COL\_TYPE=M のカラムに対して、サンプルングレートを設定（変更）します。

##### 【留意事項】

COL\_TYPE=M 以外のカラムには実行できません。  
登録処理を終了（中断）してから行う必要があります

### 6.5.基本データ操作

#### 6.5.1.sdts\_insert ()… データ登録

##### 【書式】

```
#include <speedbee.h>
int sdts_insert(sdtsdb_t db, sdtsid_t cid, sdntime_t ts, char *d, int dcnt)
```

##### 【引数】

sdtsdb_t db	…	DB 構造体ポインタ
sdtsid_t cid	…	カラム ID
sdntime_t ts	…	タイムスタンプ(UTC 1970-01-01 00:00:00 からの通算ナノ秒)
char *d	…	データ配列（or データ）の先頭アドレス
int nt	…	データ配列数（or データサイズ）

##### 【戻り値】

1 以上	:	正常終了・登録件数
-1	:	異常終了

## 【説明】

データを登録します。各カラムタイプ単位で引数を下記に従い指定します。

カラムタイプ	パラメータ	説明
H [固定長]	ts	初回のみ指定。(初回で 0 を指定した場合、システムタイムをセットする) 初回以降で継続したデータ配列を登録する場合は 0 指定。 0 以外の ts を指定した場合、その値から再スタート(リセット)となる
	d	データ配列の先頭アドレス
	dcnt	配列数 (1 データサイズは、カラム設定に従う)
M [固定長]	ts	初回のみ指定。(初回で 0 を指定した場合、システムタイムをセットする) 初回以降で継続したデータ配列を登録する場合は 0 指定。 0 以外の ts を指定した場合、前回からのサンプリングレートによる経過時間と一致したときのみ、前回の登録の続きと認識する。それ以外の場合、前回からデータの連続性は途切れ、次のデータシリーズの初回登録と認識する。
	d	データ配列の先頭アドレス
	dcnt	配列数 (1 データサイズは、カラム設定に従う)
L [固定長/ 可変長]	ts	0 の場合は、システムタイムを適用
	d	データの先頭ポインタを指定
	dcnt	固定長は 0 (指定しても内部で無視) 可変長はデータサイズ (Byte)

## 6.5.2.sdts\_get\_col\_dat ()… 単一カラムのデータ配列を取得

## 【書式】

```
#include <speedbee.h>

int sdts_get_col_dat(sdtsdb_t db, sdtsid_t cid,
                    sdntime_t st, sdntime_t et, char *rb, int rbsz, sdntime_t *rst)
```

## 【引数】

sdtsdb_t db	…	DB 構造体ポインタ
sdtsid_t cid	…	カラム ID
sdntime_t st	…	開始タイムスタンプ(UTC 1970-01-01 00:00:00 からの通算ナノ秒) 指定した開始タイムスタンプは、条件として含む。
sdntime_t et	…	終了タイムスタンプ(UTC 1970-01-01 00:00:00 からの通算ナノ秒) 指定した終了タイムスタンプは、条件として含む。
char *rb	…	出力バッファ
int rbsz	…	出力バッファサイズ
sdntime_t *rst	…	タイムスタンプ (COL_TYPE=H のみ返却)

## 【戻り値】

0 以上	:	出力されたデータ件数
上記以外	:	異常終了

## 【説明】

指定された一つのカラムから時間条件に従い、データを出力バッファにコピーします。

カラムタイプごとに出力される書式は、下記のように異なります

H	<div>rb(出力バッファ)書式</div> <table><tr><td>値 1</td><td>値 2</td><td>値 3</td><td>値 4</td><td>……</td><td>値 N</td></tr></table> <div>rst に値 1 のタイムスタンプを出力</div> <div>※ 値のデータサイズはカラムに設定したデータサイズ（固定長）</div>	値 1	値 2	値 3	値 4	……	値 N	
値 1	値 2	値 3	値 4	……	値 N			
M、L（固定長）	<div>rb(出力バッファ)書式</div> <table><tr><td>ts1</td><td>値 1</td><td>ts2</td><td>値 2</td><td>……</td><td>tsN</td><td>値 N</td></tr></table> <div>※ タイムスタンプ（8byte）と値を一組で格納</div> <div>※ タイムスタンプは UTC 1970-01-01 00:00:00 からの通算ナノ秒</div> <div>※ 値のデータサイズはカラムに設定したデータサイズ（固定長）</div> <div>※ rst には値を出力しない</div>	ts1	値 1	ts2	値 2	……	tsN	値 N
ts1	値 1	ts2	値 2	……	tsN	値 N		
L（可変長）	<div>rb(出力バッファ)書式</div> <table><tr><td>ts1</td><td>長 1</td><td>値 1</td><td>ts2</td><td>……</td><td></td></tr></table> <div>※ タイムスタンプ（8byte）と、データ長(2Byte)、値を一組で格納</div> <div>※ タイムスタンプは UTC 1970-01-01 00:00:00 からの通算ナノ秒</div> <div>rst には値を出力しない</div>	ts1	長 1	値 1	ts2	……		
ts1	長 1	値 1	ts2	……				



### 6.5.3.sdts\_get\_col\_lval ()… 単一カラムの最新値を取得

#### 【書式】

```
#include <speedbee.h>

int sdts_get_col_lval(sdtsdb_t db, sdtsqid_t cid,
                    sdntime_t *rst, char *rb, int rbsz)
```

#### 【引数】

sdtsdb_t db	…	DB 構造体ポインタ
sdtsqid_t cid	…	カラム ID
sdntime_t *rst	…	出力タイムスタンプ(UTC 1970-01-01 00:00:00 からの通算ナノ秒)
char *rb	…	出力バッファ
int rbsz	…	出力バッファサイズ

#### 【戻り値】

1 以上	:	取得成功・最新値 データサイズ (byte)
0	:	最新値が存在しない
-1	:	異常終了

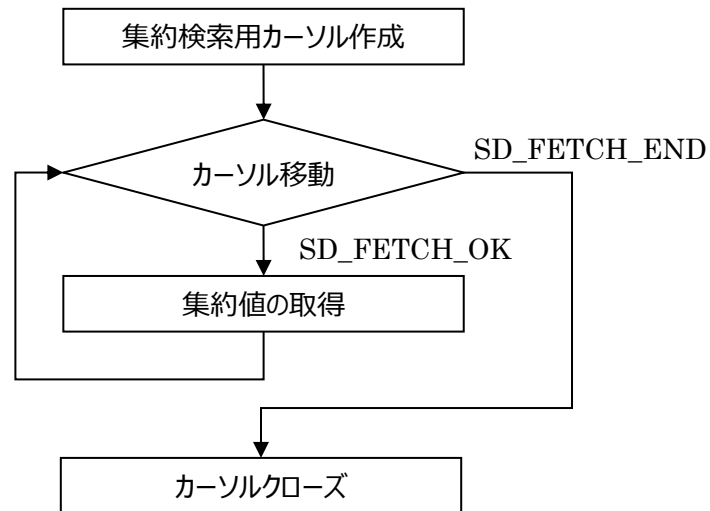
#### 【説明】

指定された一つのカラムから最新値データを出力バッファにコピーします。

## 6.6.集約検索

### 1) 集約検索フロー

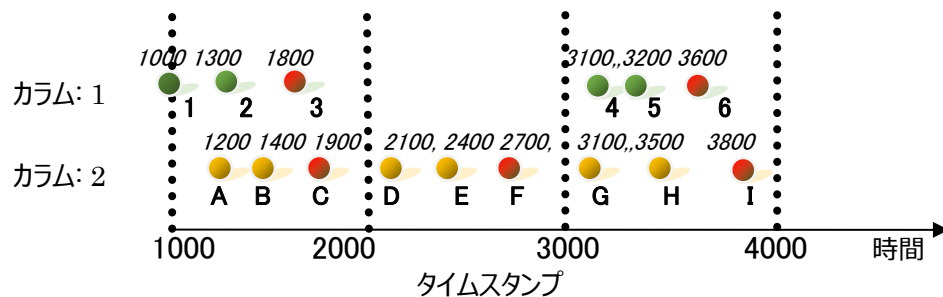
下記のフローにて集約検索を行います。



### 2) 集約検索結果

集約検索の結果について以下の図のようにデータが登録された場合を例に説明します。

(データの上はタイムスタンプ, 下の 1, 2, ... および A, B, ... は値)



集約検索の条件例として刻み値を 1000 として タイムスタンプを 1000 ~ 4000 までを検索します。

sdt\_s\_open\_cur の引数として、以下を指定します。

引数	値
1	db ディスクリプタ
2	検索対象カラム ID 配列 [カラム 1 ID, カラム 2 ID]
3	検索対象カラム I カラム数 2
4	検索開始タイムスタンプ 1000
5	検索終了タイムスタンプ 4000
6	刻み値 1000
7	集約オプション なし 0

レコード数は、検索条件のタイムスタンプ区間 1000～4000 と刻み値により決定されます。

集約オプションが指定しない場合、レコード数は、1000, 2000, 3000, 4000 の 4 件になります。刻み値のタイムスタンプの範囲は、～1000, 1001～2000, 2001～3000, 3001～4000 で、集約オプションが指定されないデフォルトでは、データが存在しない区間がある場合、その前にセットされた値を繰り返します。対象データは図の赤色で示されたデータになります。

レコード	タイムスタンプ	カラム 1	カラム 2	タイムスタンプ区間
1	1000	1	NULL	～1000
2	2000	3	C	1001～2000
3	3000	3	F	2001～3000
4	4000	6	I	3001～4000

### 6.6.1.sdts\_open\_cur ()・・・ 集約用検索カーソルを作成

#### 【書式】

```
#include <speedbee.h>

sdtscur_t sdts_open_cur(sdtsdb_t db, sdtsqid_t *cids, int ccc,
                        sdntime_t st, sdntime_t et, sdntime_t iv, ui32_t opt);
```

#### 【引数】

sdtsdb_t db	...	DB 構造体ポインタ
sdtsqid_t *cids	...	カラム ID 配列の先頭ポインタ
int ccc	...	カラム ID 配列数 カーソル検索カラム数の最大 1024 個まで
sdntime_t st	...	開始タイムスタンプ(UTC 1970-01-01 00:00:00 からの通算ナノ秒) 指定した開始タイムスタンプは、条件として含む。(inclusive)
sdntime_t et	...	終了タイムスタンプ(UTC 1970-01-01 00:00:00 からの通算ナノ秒) 指定した終了タイムスタンプは、SDTS_CUR_OPT_EXTEND オプションを指定の有り無しで、条件に含むか含まないかが変わる。 SDTS_CUR_OPT_EXTEND オプション指定なし:条件として含む。 SDTS_CUR_OPT_EXTEND オプション指定あり:条件として含まない。
sdntime_t iv	...	タイムスライスする刻値 (ナノ秒)
ui32_t opt	...	検索オプション 設定しない場合は 0 を指定

#### 【戻り値】

NULL 以外	:	正常終了 (検索用カーソルを返却)
NULL	:	異常終了

#### 【説明】

集約検索に利用するためのカーソルを作成します。

## 【検索オプション】

検索オプションは、以下のタイプがあります。

- ・検索対象ストレージ指定オプション
- ・区間内データの取得ルール集約オプション
- ・その他のオプション

## 【検索対象ストレージ指定オプション】

オプション種別	オプション名	※1	説明
メモリ検索	SDTS_STO_OPT_MEM	○	メモリ DB の参照
ファイル検索	SDTS_STO_OPT_FILE		ファイル DB の参照

※1 デフォルト

検索対象ストレージを指定します。指定がない場合のデフォルトは、メモリ検索(SDTS\_STO\_OPT\_MEM)になります。ファイル検索を行う場合は、SDTS\_STO\_OPT\_FILE を指定する必要があります。

ファイルとメモリの両方を検索したい場合は、SDTS\_STO\_OPT\_MEM | SDTS\_STO\_OPT\_FILE を指定します。この指定では、メモリとファイルの両方に取得対象データが存在する場合、メモリ優先でメモリから取得します。

## 【集約オプション】

オプション種別	オプション名	※1	説明
代表のデータを区間の最初か最後かを指定	SDTS_CUR_OPT_AGGR_VAL_TOP		代表値は区間の最初のデータ
	SDTS_CUR_OPT_AGGR_VAL_LAST	○	代表値は区間の最後のデータ
出力タイムスタンプを区間の始まりか終わりを指定	SDTS_CUR_OPT_AGGR_TS_PREV		出力タイムスタンプは区間の始まり
	SDTS_CUR_OPT_AGGR_TS_NEXT	○	出力タイムスタンプは区間の終わり
区間内に値がない場合、前の値を引き継ぐかの指定	SDTS_CUR_OPT_AGGR_VAL_CONT	○	前の値を引き継ぐ
	SDTS_CUR_OPT_AGGR_VAL_NULL		値がない場合、NULL
カーソルの終了条件の拡張	SDTS_CUR_OPT_EXTEND		CURSOR END においてカーソルの終了条件を拡張
区間内に値がない場合、前の値を引き継ぐかの指定の追加オプション	SDTS_CUR_OPT_AGGR_VAL_TOP		SDTS_CUR_OPT_AGGR_VAL_CONT 指定され、このオプションが指定された場合、先頭データ区間に値がない際に指定条件外の1つ前データの値を引き継ぐ

※1 デフォルト

集約オプションの指定方法は、OR（|）で繋ぎ、複数指定することができます。

ただし、同じ種別のオプションを同時に指定することはできません。

例えば、SDTS\_CUR\_OPT\_AGGR\_VAL\_TOP | SDTS\_CUR\_OPT\_AGGR\_VAL\_LAST は指定できません。

集約オプションが、指定がない場合(0)のデフォルトは、次のオプションが指定されていると解釈されます。

SDTS\_CUR\_OPT\_AGGR\_VAL\_LAST

SDTS\_CUR\_OPT\_AGGR\_VAL\_CONT

SDTS\_CUR\_OPT\_AGGR\_TS\_PREV

また、オプション種別で指定がないものはデフォルト値を使用します。（上記表のデフォルト）

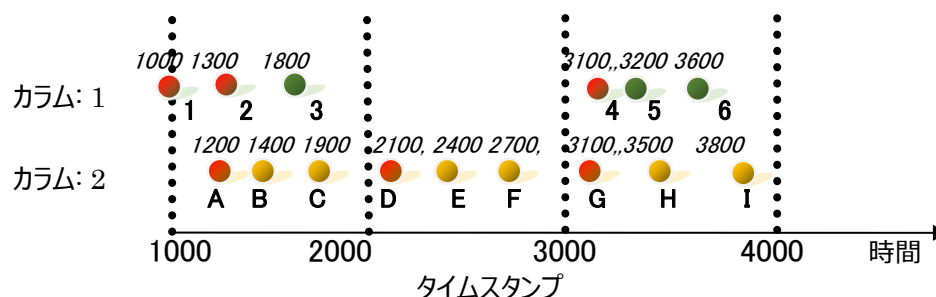
次にそれぞれのオプションの動作について説明します。

## 1) SDTS\_CUR\_OPT\_AGGR\_VAL\_TOP

対象データとして、区間の最初のデータを取得します。

以下の図のようにデータが登録された場合を例に説明します。

(データの上はタイムスタンプ, 下の 1, 2, ... および A, B, ... は値)



例として刻み値を 1000、タイムスタンプを 1000 ~ 4000、  
集約オプション SDTS\_CUR\_OPT\_AGGR\_VAL\_TOP を指定し、検索します。  
sdt\_open\_cur の引数は、以下になります。

引数	値
1	db ディスクリプタ
2	検索対象カラム ID 配列 [カラム 1 ID, カラム 2 ID]
3	検索対象カラム I カラム数 2
4	検索開始タイムスタンプ 1000
5	検索終了タイムスタンプ 4000
6	刻み値 1000
7	集約オプション SDTS_CUR_OPT_AGGR_VAL_TOP SDTS_CUR_OPT_AGGR_TS_NEXT SDTS_CUR_OPT_AGGR_VAL_NULL

刻み値のタイムスタンプの範囲は、~1000, 1001~2000, 2001~3000, 3001~4000 で  
集約オプション SDTS\_CUR\_OPT\_AGGR\_VAL\_TOP の指定により、その区間の最初のデータをカラムのデータとします。図では、区間の対象データは、赤色で示された最初のデータです。

レコード	タイムスタンプ	カラム 1	カラム 2	タイムスタンプ区間
1	1000	1 ※1	NULL	~1000
2	2000	2 ※1	A	1001~2000
3	3000	NULL	D	2001~3000
4	4000	4	G	3001~4000

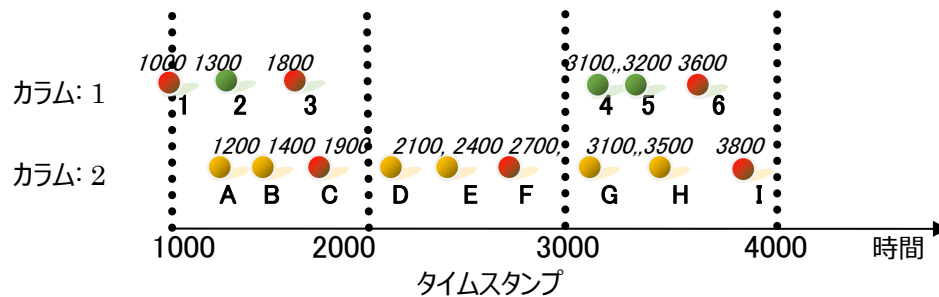
※1 刻み値上のデータ(カラム 1:1000,1) のデータは、タイムスタンプ 1000 のレコードに含み、次の 2000 のレコードは 1001 からの先頭データ (カラム 1:1300,2) がセットされます。

## 2) SDTS\_CUR\_OPT\_AGGR\_VAL\_LAST

対象データとして、区間の最後のデータを取得します。オプションに指定がない場合のデフォルトです。

以下の図ようにデータが登録された場合を例に説明します。

(データの上はタイムスタンプ, 下の 1, 2, ... および A, B, ... は値)



集約検索の条件例として刻み値を 1000 として タイムスタンプを 1000 ～ 4000 までを検索します。  
sdt Open\_cur の引数として、以下を指定します。

引数	値
1	db ディスクリプタ
2	検索対象カラム ID 配列 [カラム 1 ID, カラム 2 ID]
3	検索対象カラム I カラム数 2
4	検索開始タイムスタンプ 1000
5	検索終了タイムスタンプ 4000
6	刻み値 1000
7	集約オプション SDTS_CUR_OPT_AGGR_VAL_LAST SDTS_CUR_OPT_AGGR_TS_NEXT SDTS_CUR_OPT_AGGR_VAL_NULL

レコード数は、検索条件のタイムスタンプ区間 1000～4000 と刻み値により決定されます。

集約オプションが指定しない場合、レコード数は、1000, 2000, 3000, 4000 の 4 件になります。刻み値のタイムスタンプの範囲は、～1000, 1001～2000, 2001～3000, 3001～4000 で、集約オプションが指定されないデフォルトでは、その区間の最後のデータをカラムのデータとし、また、タイムスタンプ値は、区間の終わりの値とします。対象データは図の赤色で示されたデータになります。

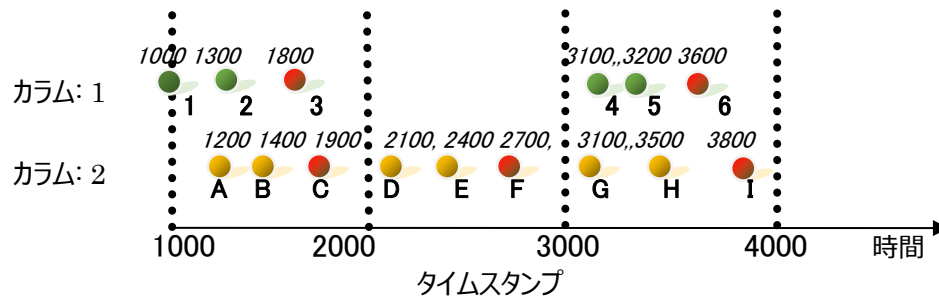
レコード	タイムスタンプ	カラム 1	カラム 2	タイムスタンプ区間
1	1000	1	NULL	～1000
2	2000	3	C	1001～2000
3	3000	NULL	F	2001～3000
4	4000	6	I	3001～4000

### 3) SDTS\_CUR\_OPT\_AGGR\_TS\_PREV

タイムスタンプを区間の始まりの値とします。

以下の図ようにデータが登録された場合を例に説明します。

(データの上はタイムスタンプ,下の 1,2,... および A,B,...は値)



例として刻み値を 1000 として タイムスタンプを 1000 ～ 4000、  
集約オプション SDTS\_CUR\_OPT\_AGGR\_TS\_PREV を指定し、検索します。  
sdt\_open\_cur の引数は、以下になります。

引数	値
1	db ディスクリプタ
2	検索対象カラム ID 配列 [カラム 1 ID, カラム 2 ID]
3	検索対象カラム I カラム数 2
4	検索開始タイムスタンプ 1000
5	検索終了タイムスタンプ 4000
6	刻み値 1000
7	集約オプション SDTS_CUR_OPT_AGGR_VAL_LAST SDTS_CUR_OPT_AGGR_TS_PREV SDTS_CUR_OPT_AGGR_VAL_NULL

SDTS\_CUR\_OPT\_AGGR\_TS\_PREV の指定により、タイムスタンプの範囲は、1000～1999, 2000～2999,3000～3999,4000～4999 となり、タイムスタンプは区間の始まりの値になります。

(タイムスタンプの区間がデフォルトと異なることに留意してください。)

例では、区間の終わりのデータを取得します。

レコード	タイムスタンプ	カラム 1	カラム 2	タイムスタンプ区間
1	1000	3	C	1000～1999
2	2000	NULL	F	2000～2999
3	3000	6	I	3000～3999
4	4000	NULL	NULL	4000～4999



#### 4) SDTS\_CUR\_OPT\_AGGR\_TS\_NEXT

タイムスタンプを区間の終わりの値とします。オプションに指定がない場合のデフォルトです。

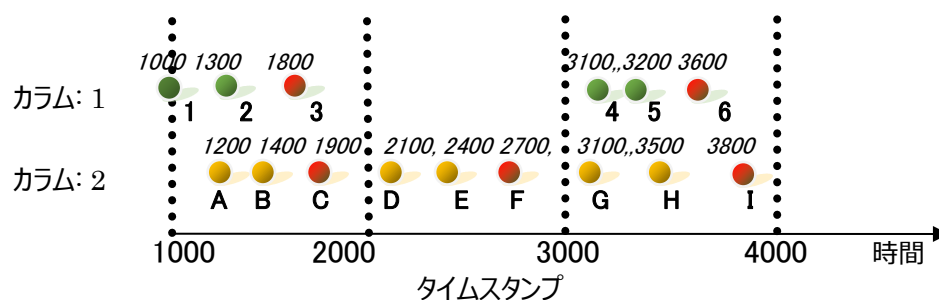
指定例は、以下を参照してください。

- 1) SDTS\_CUR\_OPT\_AGGR\_VAL\_TOP
- 2) SDTS\_CUR\_OPT\_AGGR\_VAL\_LAST

#### 5) SDTS\_CUR\_OPT\_AGGR\_VAL\_CONT

データが存在しない区間がある場合、その前にセットされた値を繰り返します。指定がない場合のデフォルトになります。以下の図のようにデータが登録された場合を例に説明します。

(データの上はタイムスタンプ, 下の 1, 2, ... および A, B, ... は値)



例として刻み値を 1000 として タイムスタンプを 1000 ~ 4000、  
集約オプション SDTS\_CUR\_OPT\_AGGR\_VAL\_CONT を指定し、検索します。  
sdts\_open\_cur の引数は、以下になります。

引数	値
1	db ディスクリプタ
2	検索対象カラム ID 配列 [カラム 1 ID, カラム 2 ID]
3	検索対象カラム I カラム数 2
4	検索開始タイムスタンプ 1000
5	検索終了タイムスタンプ 4000
6	刻み値 1000
7	集約オプション SDTS_CUR_OPT_AGGR_VAL_CONT SDTS_CUR_OPT_AGGR_VAL_LAST SDTS_CUR_OPT_AGGR_TS_NEXT

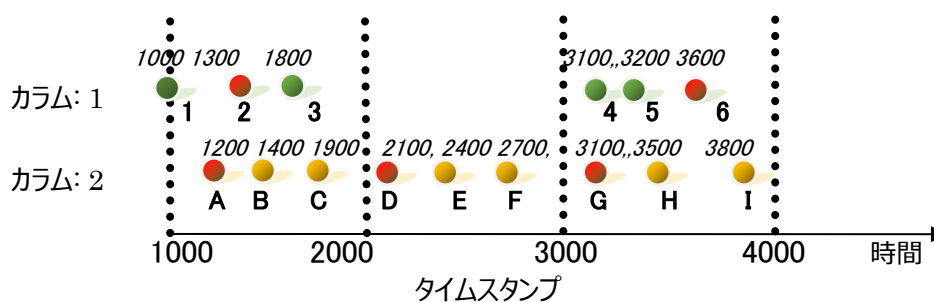
例では、カラム 1 の刻みの区間 2001~3000 までの間にデータが存在しません。

SDTS\_CUR\_OPT\_AGGR\_VAL\_CONT を指定することにより、直前の区間のカラムデータの値を繰り返し、  
NULL 値を埋めます。この例ではカラム 1:1800,3 のデータが区間 2001~3000 のデータとなります。

レコード	タイムスタンプ	カラム 1	カラム 2	タイムスタンプ区間
1	1000	1	NULL ※1	～1000
2	2000	3	C	1001～2000
3	3000	3	F	2001～3000
4	4000	6	I	3001～4000

※1 レコード1のカラム2は、1000以前の値がないため NULL になります。

この例において SDTS\_CUR\_OPT\_AGGR\_VAL\_CONT と SDTS\_CUR\_OPT\_AGGR\_VAL\_TOP を指定した結果を以下に示します。



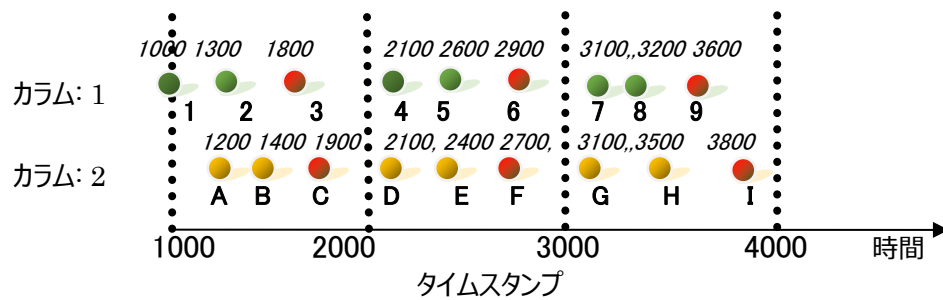
レコード	タイムスタンプ	カラム 1	カラム 2	タイムスタンプ区間
1	1000	1	NULL	～1000
2	2000	2	A	1001～2000
3	3000	2	D	2001～3000
4	4000	4	G	3001～4000

区間内にデータが存在しない場合の繰り返す値は、直前ではなく、直前の区間の代表値

（SDTS\_CUR\_OPT\_AGGR\_VAL\_TOP であれば区間の先頭データ、デフォルトであれば最終データ）になります。また、NULL 期間が連続する場合は、連続する前の区間の値がセットされます。

SDTS\_CUR\_OPT\_AGGR\_VAL\_CONT により、値がセットされた場合のカラム値のインディケータは、SD\_COL\_IND\_VAL\_CONT(2)を示します。

最初の例と同じオプション指定時に、区間内のデータが存在する例について、下記に示します。



刻み値を 1000 として タイムスタンプを 1000 ～ 4000、  
集約オプション SDTS\_CUR\_OPT\_AGGR\_VAL\_CONT を指定し、検索します。  
sdtc\_open\_cur の引数は、以下になります。

引数	値
1	db ディスクリプタ
2	検索対象カラム ID 配列 [カラム 1 ID, カラム 2 ID]
3	検索対象カラム I カラム数 2
4	検索開始タイムスタンプ 1000
5	検索終了タイムスタンプ 4000
6	刻み値 1000
7	集約オプション SDTS_CUR_OPT_AGGR_VAL_CONT SDTS_CUR_OPT_AGGR_VAL_LAST SDTS_CUR_OPT_AGGR_TS_NEXT

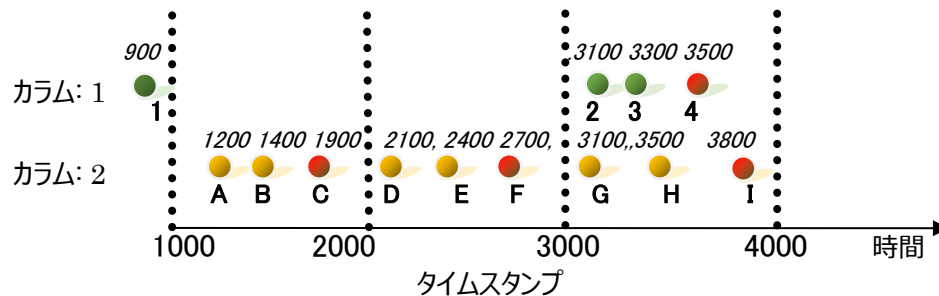
この例では、カラム 1 の刻みの区間 2001～3000 までの間にもデータが存在しています。

SDTS\_CUR\_OPT\_AGGR\_VAL\_CONT を指定することにより、直前の区間のカラムデータの値を繰り返し、  
NULL 値を埋めます。この例ではカラム 1:1000 のみカラム 2 が NULL となります。

レコード	タイムスタンプ	カラム 1	カラム 2	タイムスタンプ区間
1	1000	1	NULL ※1	～1000
2	2000	3	C	1001～2000
3	3000	6	F	2001～3000
4	4000	9	I	3001～4000

## 6) SDTS\_CUR\_OPT\_AGGR\_VAL\_CONT\_TOP

SDTS\_CUR\_OPT\_AGGR\_VAL\_CONT と一緒にこのオプションが指定された場合、結果の先頭にデータ存在しない際、検索条件外の 1 つ前のデータを先頭データとして引き継ぎます。



例として刻み値を 1000 として タイムスタンプを 1000 ~ 4000、集約オプション SDTS\_CUR\_OPT\_AGGR\_VAL\_CONT\_TOP を指定し、検索します。sdtc\_open\_cur の引数は、以下になります。

引数	値
1	db ディスクリプタ
2	検索対象カラム ID 配列 [カラム 1 ID, カラム 2 ID]
3	検索対象カラム I カラム数 2
4	検索開始タイムスタンプ 1000
5	検索終了タイムスタンプ 4000
6	刻み値 1000
7	集約オプション SDTS_CUR_OPT_AGGR_VAL_CONT SDTS_CUR_OPT_AGGR_VAL_LAST SDTS_CUR_OPT_AGGR_TS_PREV SDTS_CUR_OPT_AGGR_VAL_CONT_TOP

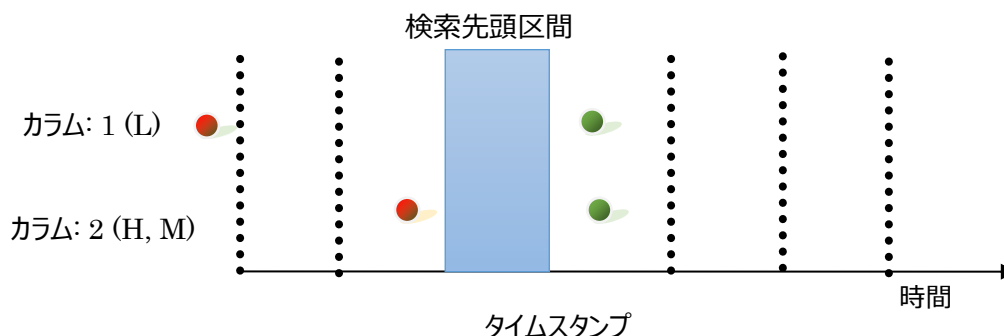
例では、カラム 1 のタイムスタンプを 1000 ~ 2000、2000 ~ 3000 の区間にデータが存在しません。SDTS\_CUR\_OPT\_AGGR\_VAL\_CONT オプションだけの指定だとこの区間の値は、NULL になりますが、SDTS\_CUR\_OPT\_AGGR\_VAL\_CONT\_TOP が指定された場合は、検索開始条件前の 1 つ前のデータを値として引き継ぎます。この例ではタイムスタンプ 900 の値 1 がセットされます。

また、先頭の結果として検索開始条件前の1つ前のデータを値として引き継いだ場合のデータ取得インディケータはSD\_COL\_IND\_VAL\_CONT\_TOP(3) を返します。

このオプションは、カラムタイプ H, M, L に対応し、1つ前のデータの定義は、どのタイプも同じで検索結果の先頭のデータ（下の図では緑の点）の1つ前のデータ（下の図では赤の点）を指します。Lタイプのような不定期

レコード	タイムスタンプ	カラム 1	カラム 1 取得インディケータ	カラム 2	カラム 2 取得インディケータ	タイムスタンプ区間
1	1000	1	3	C	1	1000～1999
2	2000	1	2	F	1	2000～2999
3	3000	4	1	I	1	3000～3999
4	4000	4	2	I	1	4000～4999

データの場合は、1つ前のデータのタイムスタンプが数時間前のものであっても先頭データの直前データということでその値を引き継ぎます。1つ前のデータ存在しない場合は、NULL をセットします。



## 7) SDTS\_CUR\_OPT\_AGGR\_VAL\_NULL

データが存在しない区間がある場合、値として NULL を返します

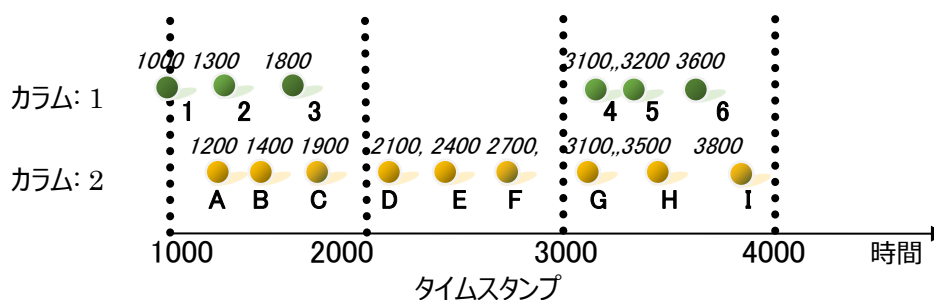
指定例は、以下を参照してください。

- 1) SDTS\_CUR\_OPT\_AGGR\_VAL\_TOP
- 2) SDTS\_CUR\_OPT\_AGGR\_VAL\_LAST
- 3) SDTS\_CUR\_OPT\_AGGR\_TS\_PREV

## 8) 指定例

以下の図のようにデータが登録された場合における複合指定の例をいくつか示します。

（データの上はタイムスタンプ、下の 1,2,... および A,B,...は値）



- ① SDTS\_CUR\_OPT\_AGGG\_TS\_PREV | SDTS\_CUR\_OPT\_AGGG\_VAL\_TOP |  
SDTS\_CUR\_OPT\_AGGG\_VAL\_NULL を指定した場合  
・sdts\_open\_cur の引数

引数	値
1	db ディスクリプタ
2	検索対象カラム ID 配列 [カラム 1 ID, カラム 2 ID]
3	検索対象カラム I カラム数 2
4	検索開始タイムスタンプ 1000
5	検索終了タイムスタンプ 4000
6	刻み値 1000
7	集約オプション SDTS_CUR_OPT_AGGG_TS_PREV SDTS_CUR_OPT_AGGG_VAL_TOP SDTS_CUR_OPT_AGGG_VAL_NULL

・結果

レコード	タイムスタンプ	カラム 1	カラム 2	タイムスタンプ区間
1	1000	1	A	1000～1999
2	2000	NULL	D	2000～2999
3	3000	4	G	3000～3999
4	4000	NULL	NULL	4000～4999

- ② SDTS\_CUR\_OPT\_AGGG\_TS\_PREV | SDTS\_CUR\_OPT\_AGGG\_VAL\_CONT |  
SDTS\_CUR\_OPT\_AGGG\_VAL\_LAST を指定した場合  
・sdts\_open\_cur の引数

引数	値
1	db ディスクリプタ
2	検索対象カラム ID 配列 [カラム 1 ID, カラム 2 ID]
3	検索対象カラム I カラム数 2
4	検索開始タイムスタンプ 1000
5	検索終了タイムスタンプ 4000
6	刻み値 1000
7	集約オプション SDTS_CUR_OPT_AGGG_TS_PREV SDTS_CUR_OPT_AGGG_VAL_CONT SDTS_CUR_OPT_AGGG_VAL_LAST

## ・結果

レコード	タイムスタンプ	カラム 1	カラム 2	タイムスタンプ区間
1	1000	3	C	1000～1999
2	2000	3	F	2000～2999
3	3000	6	I	3000～3999
4	4000	6	I	4000～4999

- ③ SDTS\_CUR\_OPT\_AGGG\_TS\_PREV | SDTS\_CUR\_OPT\_AGGG\_VAL\_TOP |  
SDTS\_CUR\_OPT\_AGGG\_VAL\_CONT を指定した場合

・sdtc\_open\_cur の引数

引数	値
1	db ディスクリプタ
2	検索対象カラム ID 配列 [カラム 1 ID, カラム 2 ID]
3	検索対象カラム I カラム数 2
4	検索開始タイムスタンプ 1000
5	検索終了タイムスタンプ 4000
6	刻み値 1000
7	集約オプション SDTS_CUR_OPT_AGGG_TS_PREV SDTS_CUR_OPT_AGGG_VAL_TOP SDTS_CUR_OPT_AGGG_VAL_CONT

## ・結果

レコード	タイムスタンプ	カラム 1	カラム 2	タイムスタンプ区間
1	1000	1	A	1000～1999
2	2000	1	D	2000～2999
3	3000	4	G	3000～3999
4	4000	4	G	4000～4999

## 【その他オプション】

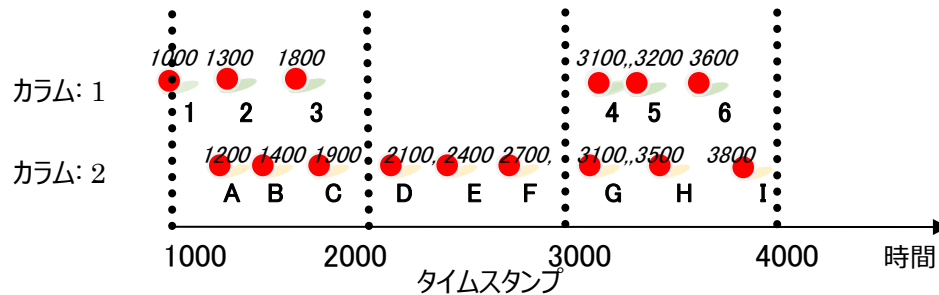
オプション種別	オプション名	※1	説明
ファイルパス結果キャッシュの有効化	SDTS_CAC_OPT_SEA_RES		ファイルパス検索結果キャッシュを利用する。詳しくは 6.8.1spcf_get_col_dat() 【ファイルパス検索結果キャッシュ】を参照

※1 デフォルト

## 【シリアル化検索】

刻み値に 0 を指定した場合、集約検索ではなく、データ 1 件単位にシリアル化され 1 列に取得することができます。以下の図のようにデータが登録された場合を例に説明します。

(データの上はタイムスタンプ, 下の 1, 2, ... および A, B, ... は値)



例として刻み値を 0 として タイムスタンプを 1000 ～ 4000 までを検索します。  
sdt\_s\_open\_cur の引数として、以下を指定します。

引数	値
1	db ディスクリプタ
2	検索対象カラム ID 配列 [カラム 1 ID, カラム 2 ID]
3	検索対象カラム I カラム数 2
4	検索開始タイムスタンプ 1000
5	検索終了タイムスタンプ 4000
6	刻み値 0
7	集約オプション なし 0

対象レコードは、図のそれぞれのデータで、結果は以下になります。

レコード	カラム	タイムスタンプ	データ
1	カラム 1	1000	1
2	カラム 1	1100	2
3	カラム 2	1200	A
4	カラム 2	1400	B
5	カラム 1	1800	3
6	カラム 2	1900	C
7	カラム 2	2100	D
8	カラム 2	2400	E
9	カラム 2	2700	F
10	カラム 1	3100※1	4
11	カラム 2	3100※1	G
12	カラム 1	3200	5
13	カラム 2	3500	H
14	カラム 1	3600	6
15	カラム 2	3800	I



※ 1 複数カラムでタイムスタンプが同じ場合、sdts\_open\_cur に指定されたカラム配列の順になります。

#### 【留意事項】

シリアルサイズ検索を実行し、データ取得する場合、集約検索用データ取得 API sdts\_get\_cur\_aggr ではなく、シリアルサイズ検索用データ取得 API sdts\_get\_cur\_dat を使用してください。

#### 【カーソル情報】

sdts\_open\_cur()によって取得したカーソルディスクリプタから、カーソル情報を取得することができます。取得可能な情報は以下になります。

##### 1) カレントカーソル情報

フェッチ時に毎回変更される情報です。

#	名前	説明
1	ref_fst	カーソルフェッチステータス フェッチ実行後に設定されるステータス情報で、以下の情報を確認することができます。 SD_CUR_NUL_REC(01) カレントレコードが NULL レコード。
2	ref_ctm	カレントレコードのタイムスタンプ

##### 2) sdts\_open\_cur()に指定した検索条件

sdts\_open\_cur()が成功した時点で取得が可能で、値が変わらない情報です。

#	名前	説明
1	ref_cst	開始タイムスタンプ
2	ref_cet	終了タイムスタンプ
3	ref_aiv	刻み値
4	ref_opt	集約オプション
5	ref_ccc	検索対象カラム数

##### 3) 検索結果件数

sdts\_open\_cur()が成功した時点で取得が可能で、値が変わらない情報です。

#	名前	説明
1	ref_acnt	集約検索結果のレコード件数
2	ref_tcnt	すべての指定カラムにおいて条件に一致したデータ件数 集約検索では、値が 0 である場合、該当データが見つからないことを示す。 (すべてのレコードは NUL レコードになる) シリアルサイズ検索ではレコード件数

カーソルディスクリプタの定義は以下になります。

・カーソルディスクリプタ

typedef struct {	名前	説明
sdntime_t	ref_cst;	開始タイムスタンプ
sdntime_t	ref_cet	終了タイムスタンプ
sdntime_t	ref_aiv	刻み値
ui32_t	ref_opt	集約オプション
int	ref_ccc	検索対象カラム数
ui32_t	ref_tcnt	条件に一致したデータ件数 ※2
	ref_acnt	集約検索結果のレコード件数
	ref_fst	カレントカーソルフェッチステータス
	ref_ctm	カレントレコードタイムスタンプ
} *sdtscur_t;		

※これらの変数は、参照を目的としたものですので更新はしないでください。

※2 ret\_tcnt はファイル検索を行った場合、常に 1 です。ファイル検索時は参照しないでください。

カーソル情報の利用として、以下のような使い方ができます。

- ① 集約検索において条件に一致するデータが見つかったかどうかを確認したい。

cur->ref\_tcnt が 1 以上であれば、条件に一致するデータがあります。

sdts\_open\_cur()実行直後に確認ができます。

- ② フェッチ実行前にレコード件数を知りたい。

sdts\_open\_cur() 実行直後に集約検索では、cur->ref\_acnt、シリアルライズ検索では cur->ref\_tcnt を参照することでレコード件数を確認することができます。

- ③ フェッチ実行後、カレントレコードが NUL レコードであるかどうかを確認したい。

sdts\_fetch\_cur()実行直後に、cur->ref\_fst カレントフェッチステータスを参照することで NUL レコードか確認できます。NUL レコードかどうかのコーディング例は、以下になります。

```
if ((cur->ref_fst & SD_CUR_NUL_REC) != 0)
```

### 6.6.2.sdts\_fetch\_cur ()…カーソルの移動

#### 【書式】

```
#include <speedbee.h>
int sdts_fetch_cur(sdtscur_t cur)
```

#### 【引数】

sdtscur_t cur	…	集約用検索カーソル
---------------	---	-----------

#### 【戻り値】

SD_FETCH_OK(1)	:	正常終了
SD_FETCH_END(0)	:	検索データ終了
SD_FETCH_ERR(-1)	:	異常終了

#### 【説明】

集約用検索カーソルを移動します。

### 6.6.3.sdts\_get\_cur\_aggr ()…集約検索カーソルから各カラムの集約値を取得

#### 【書式】

```
#include <speedbee.h>
int sdts_get_cur_aggr(sdtscur_t cur, sdntime_t *rts, sdtscurval_t *avarr, int acnt)
```

#### 【引数】

sdtscur_t cur	…	集約検索カーソル
sdntime_t *rts	…	タイムスタンプバッファ（出力）
sdtscurval_t *avarr	…	カーソルデータ出力（集約値）配列（出力） 配列数は、sdts_open_cur で指定したカラム数
int acnt	…	カーソルデータ出力（集約値）の配列数

#### 【戻り値】

0	:	正常終了
-1	:	異常終了

#### 【説明】

現在の集約検索カーソルから各カラムの集約値を取得します。

カレントタイムスタンプは、第 2 引数の rts に格納されます。また、データ値は、第 3 引数の avarr カーソルデータ出力（集約値）配列に格納されます。この配列は、コールする側で sdts\_open\_cur に指定したカラム数と同数を用意して下さい。

## ・カーソルデータ出力構造体 sdtscurval\_t

typedef struct {	名前	説明
sdtscid_t	cid	カラム ID
sdntime_t	ts	データタイムスタンプ※1
int	ind	データ取得インディケータ※2
char	*vp	データポインタ※3
ui32_t	vsz	データサイズ
} sdtscurval_t		

## ※1 データタイムスタンプ

データタイムスタンプは、集約したタイムスタンプ(sdts\_get\_cur\_aggr 引数 rts 値)ではなく、該当データのタイムスタンプがセットされています。

## ※2 データ取得インディケータ

データ取得インディケータは、以下を示します。

インディケータ	説明
SD_COL_IND_NOT_FOUND(-1)	該当カラムのデータが見つからない。(検索条件に一致するデータが存在しない) これは対象カラムのレコード全体で、同じステータスが返される。
SD_COL_IND_NUL(0)	カレントの集約のタイムスタンプ区間内において対象データが存在しない。
SD_COL_IND_GET(1)	カレントの集約のタイムスタンプ区間内において対象データが存在し、データをカーソルデータ出力構造体にセットした。
SD_COL_IND_VAL_CONT(2)	集約オプション SDTS_CUR_OPT_AGGR_VAL_CONT が設定された場合にカレントの集約のタイムスタンプ区間内において対象データが存在せず、その前の値を繰り返しセットしたことを示す。
SD_COL_IND_CONT_TOP(3)	集約オプション SDTS_CUR_OPT_AGGR_CONT_TOP が設定され、検索結果出力の先頭の区間内にデータが存在しない場合に、1 つ前のデータ値をセットしたことを示す。

## ※3 データポインタ

データポインタは、以下の点に留意が必要です。

- ・カーソル内部で確保している一時領域のため、次の FETCH 処理にて上書きされます。FETCH 中に値の保全が必要な場合は、アプリ側のバッファ領域にコピーして下さい。
- ・データ取得インディケータが、SD\_COL\_IND\_NOT\_FOUND(-1)、SD\_COL\_IND\_NUL(0)の場合、データポインタには NULL がセットされるため、アプリケーションでの出力する場合、確認が必要です。

## 【使用例】

sdts\_get\_cur\_aggr()を使ったコーディング例を示します。

取得したデータをプリントする例でカラム数は2、int 型の値とします。（※エラー処理は省略します）

```
#define COL_CNT 2
sdntime_t ts;
sdtscurval_t cval[COL_CNT];
sdts_get_cur_aggr(cur, &ts, cval, COL_CNT); // 値取得
printf("timestamp :%lld¥n", ts);          // 集約のタイムスタンプ出力
for (i = 0; i < COL_CNT; i++) {
    printf("  cid:%d real:%lld", cval[i].cid, cval[i].ts); // カラム ID 出力と実際のタイムスタンプ
    // インディケータが 1 以上であれば値がセットされています。
    if (cval[i].ind > 0) {
        printf(" value:%d", *(int *)cval[i].vp);
    } else {
        printf(" value:null");
    }
    printf("¥n");
}
```

## 6.6.4.sdts\_extend\_cur ()…検索カーソルの終了条件拡張

## 【書式】

```
#include <speedbee.h>
int sdts_extend_cur(sdtscur_t cur, sdntime_t et)
```

## 【引数】

sdtscur_t cur	…	集約・終了条件拡張用検索カーソル カーソルオープン時 sdts_open_cur()のオプションに SDTS_CUR_OPT_EXTEND を指定し、 フェッチステータスが SD_FETCH_END まで進んだカーソル
sdntime_t et	…	新しい終了条件：タイムスタンプ 指定した終了タイムタイムスタンプは、条件として含まない。次回、拡張した際の開始タイムスタンプとなる。

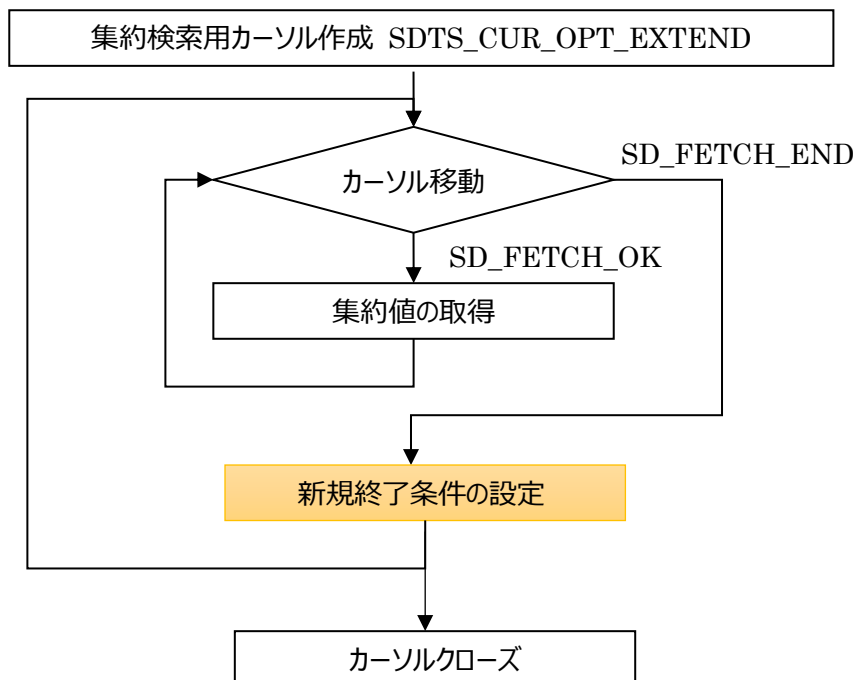
## 【戻り値】

0 :	正常終了
-1 :	異常終了

## 【説明】

カーソルフェッチ API(sdts\_fetch\_cur) により、終了ポイントの SD\_FETCH\_END まで進んだカーソルに対し、新規の終了条件を指定し、カーソルの終了ポイントを拡張します。

フローとしては、以下図のように SD\_FETCH\_END まで終了したカーソルに対し、新しい終了条件を設定することで、再度、検索カーソルを作成し直します。また、拡張した場合の検索開始は、前回の終了時間からになります。



## 【留意事項】

終了拡張 API を利用する場合、オープンカーソル時に SDTS\_CUR\_OPT\_EXTEND 検索オプションを指定してください。

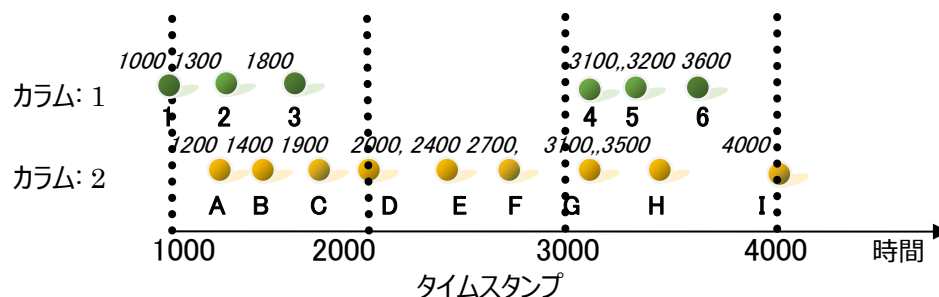
```
指定例)cur = sdts_open_cur(db, cids, 1, 1000000000ULL, 2000000000ULL,
      1000000, SDTS_CUR_OPT_EXTEND)
```

SDTS\_CUR\_OPT\_EXTEND を指定しない通常のオープンカーソルによる検索との違いですが、通常の検索では、結果として引数の終了時間までを検索の範囲としますが SDTS\_CUR\_OPT\_EXTEND を指定した検索では、終了時間は、次の開始時間となるため、**終了時間ちょうどの値が見つかった場合、検索結果に含みません**。この点が通常の検索と違いがあります。このことは、拡張 API でも同じで、指定した終了時間ちょうどの値は、カーソルの結果には含まず、拡張 API による次のカーソルの結果に含まれます。そのため、拡張 API を使用する場合、クローズカーソルの直前の最後の検索において、指定した終了時間を含める必要がある場合、終了条件にプラス 1 した値を設定します。

カーソル終了の拡張は、集約カーソルおよびシリアライズのどちらでも可能です。

この API の利用方法として、直前の SD\_FETCH\_END までのカーソル情報を保持しているため、カーソルオプション SDTS\_CUR\_OPT\_AGGR\_VAL\_CONT を指定したデータが存在しない区間の値を引き継がせたい場合に利用します。

以下のデータの場合の終了拡張 API を使った検索例を示します。



次の順序でコールする場合

- 1) オープンカーソル 開始 1000 ～終了 1999  
初回 sdts\_open\_cur の引数

引数	値
1	db ディスクリプタ
2	検索対象カラム ID 配列 [カラム 1 ID, カラム 2 ID]
3	検索対象カラム I カラム数 2
4	検索開始タイムスタンプ 1000
5	検索終了タイムスタンプ 2000 ※SDTS_CUR_OPT_EXTEND の指定で 2000 は含まない。
6	刻み値 1000
7	集約オプション

	SDTS_CUR_OPT_AGGR_TS_PREV SDTS_CUR_OPT_AGGR_VAL_TOP SDTS_CUR_OPT_AGGR_VAL_CONT SDTS_CUR_OPT_EXTEND
--	---

- 2) 終了拡張 API `sdts_extend_cur` 初回 開始 2000 ～終了 2999  
初回 `sdts_extend_cur` の引数

引数	値
1	カーソルディスクリプタ
2	検索終了タイムスタンプ 3000 ※SDTS_CUR_OPT_EXTEND の指定で 3000 は含まない。

- 3) 終了拡張 API 2 回目 開始 3000 ～終了 3999  
2 回目 `sdts_extend_cur` の引数

引数	値
1	カーソルディスクリプタ
2	検索終了タイムスタンプ 4000 ※SDTS_CUR_OPT_EXTEND の指定で 4000 は含まない。

結果は、以下になります。

- 1) 初回、オープンカーソル

レコード	タイムスタンプ	カラム 1	カラム 2	タイムスタンプ区間
1	1000	1	A	1000～1999

- 2) 終了拡張 API `sdts_extend_cur` 初回

レコード	タイムスタンプ	カラム 1	カラム 2	
1	2000	1	D	2000～2999

- 3) 終了拡張 API `sdts_extend_cur` 2 回目

レコード	タイムスタンプ	カラム 1	カラム 2	
1	3000	4	G	3000～3999



### 6.6.5.sdts\_get\_cur\_dat ()…シリアルイズ検索カーソルからカラムの値を取得

**【書式】**

```
#include <speedbee.h>
int sdts_get_cur_dat (sdtscur_t cur, sdtscurval_t *vp)
```

**【引数】**

sdtscur_t cur	…	シリアルイズ検索カーソル
sdtscurval_t *vp	…	カーソルデータ出力ポインタ（出力）

**【戻り値】**

0	:	正常終了
-1	:	異常終了

**【説明】**

現在のシリアルイズ検索カーソルからカラム値を取得します。

第 2 引数の vp には、コールする側で値を格納するための sdtscurval\_t 構造体のバッファを用意してください。シリアルイズ検索では、常に 1 レコードにつきカラム 1 つであるため、sdts\_get\_cur\_aggr()のように配列を指定する必要はありません。

カーソルデータ出力ポインタについては、5.5.3 sdts\_get\_cur\_aggr()を参照してください。

### 6.6.6.sdts\_close\_cur ()…カーソルのクローズ

**【書式】**

```
#include <speedbee.h>
int sdts_close_cur(sdtscur_t cur)
```

**【引数】**

sdtscur_t cur	…	集約用検索カーソル
---------------	---	-----------

**【戻り値】**

0	:	正常終了
-1	:	異常終了

**【説明】**

集約検索用カーソルをクローズします。

**【留意事項】**

使用し終えたカーソルは必ずクローズしてください。(クローズし忘れるとメモリリークになります。)

## 6.7.リアルタイム処理

### 6.7.1.sdts\_set\_win ()…リアルタイム処理用のウィンドウ定義の設定

#### 【書式】

```
#include <speedbee.h>
int sdts_set_win(sdtsdb_t db, sdtscid_t cid, char *dname,
                int wt, char *wap, sdudfn_t db, void *userdata)
```

#### 【引数】

sdtsdb_t db	…	DB 構造体ポインタ
sdtscid_t cid	…	カラム ID
char *dname	…	ウィンドウ定義名
int wt	…	ウィンドウタイプ
char *wapar	…	ウィンドウパラメータ
sdudfn_t cb	…	通知コールバック関数
void *userdata	…	ユーザー定義データ

#### 【戻り値】

0	:	正常終了
-1	:	異常終了

#### 【説明】

リアルタイム処理用のウィンドウ定義を設定します。指定したウィンドウタイプに応じて、ウィンドウパラメータに設定された処理が実行されます。カラムタイプ LO 可変長のカラムに対してはウィンドウ定義できません。

#### 【ウィンドウタイプ】

ウィンドウ種別	ウィンドウタイプ
カウントウィンドウ	SD_WT_COUNT
FFT ウィンドウ	SD_WT_FFT

#### 【ウィンドウパラメータ】

ウィンドウパラメータは、“パラメータ名=値”のフォーマットで指定します。複数指定する場合は、;(セミicolon)に続けて指定してください。

パラメータ名	ウィンドウ種別	必須	説明
WIN_STAT	カウント FFT		true : 統計処理を行う (デフォルト) false : 統計処理を行わない
WIN_DTIN	カウント FFT	○	登録データのデータタイプを指定します。 "i8","i16","i32","i64""ui8","ui16","ui32","ui64" ","float" "double"のいずれかを指定します。
WIN_COUNT	カウント	○	カウントウィンドウの件数を指定します。指定登録件数単位でウィンドウ処理を行います。

FFT_COUNT	FFT	○	FFT ウィンドウの件数を指定します。指定登録件数単位で FFT ウィンドウ処理を行います。 256,512,1024,2048,4096,8192,16384 のいずれかを指定します。
FFT_SMPL	FFT	○	FFT サンプリングレート (Hz)

## 【通知コールバック関数】

ウィンドウ処理が完了した時点で呼び出される関数です。

## 【書式】

```
#include <speedbee.h>
typedef int (*sdudfn_t)(sdntime_t ts, void *da, int dc,
                        sdstat_t *st, int ret, void *userdata);
```

## 【引数】

sdntime_t st	...	ウィンドウ開始時刻(UTC 1970-01-01 00:00:00 からの通算ナノ秒)
void *da	...	カウントウィンドウの場合 ウィンドウ内のデータ配列の先頭ポインタ (※ 1) FFT ウィンドウの場合 FFT 結果データ配列の先頭ポインタ (※ 2)
int dc	...	ウィンドウ内のデータ件数 (da の配列サイズ)
sdstat_t *st	...	基本統計量データ (※ 3) sdts_set_win のウィンドウパラメータで WIN_STAT=falseと指定した場合は NULL
int ret	...	0 固定
void *userdata	...	ユーザー定義データ。sdts_set_win に指定したデータ。

## 【戻り値】

0 を返します。

※ 1 データ配列はカラムタイプごとに下記ようになります。

H	データ配列書式 <table><tr><td>値 1</td><td>値 2</td><td>値 3</td><td>値 4</td><td>……</td><td>値 N</td></tr></table> ※値のデータサイズはカラムに設定したデータサイズ（固定長）	値 1	値 2	値 3	値 4	……	値 N	
値 1	値 2	値 3	値 4	……	値 N			
M、L（固定長）	データ配列書式 <table><tr><td>ts1</td><td>値 1</td><td>ts2</td><td>値 2</td><td>……</td><td>tsN</td><td>値 N</td></tr></table> ※タイムスタンプ（8byte）と値を一組で格納 ※タイムスタンプは UTC 1970-01-01 00:00:00 からの通算ナノ秒 ※値のデータサイズはカラムに設定したデータサイズ（固定長）	ts1	値 1	ts2	値 2	……	tsN	値 N
ts1	値 1	ts2	値 2	……	tsN	値 N		

※ 2 FFT 結果データは以下の構造体の配列となります

```
#include <tsflib.h>
```

```
tsf_fft_res_t
```

typedef struct	名前	説明
{		
double	freq	周波数
double	magni	スペクトル強度
} tsf_fft_res_t		

※ 3 基本統計量データは以下の構造体となります

```
sdstat_t
```

typedef struct	名前	説明
{		
double	sum	総和
double	sumsq	2 乗和
double	min	最小値
double	max	最大値
double	mean	平均
double	var	分散
double	stdev	標準偏差
double	uvar	不偏分散
double	ustdev	標本不偏標準偏差
double	stder	標準誤差
sdntime_t	stime	ウィンドウ開始時刻(UTC 1970-01-01 00:00:00 からの通算ナノ秒)
sdntime_t	etime	ウィンドウ終了時刻(UTC 1970-01-01 00:00:00 からの通算ナノ秒)
int	cnt	ウィンドウ内データ件数
} sdstat_t		

### 6.7.2.sdts\_unset\_win ()・・・リアルタイム処理用の指定ウィンドウ定義の削除

**【書式】**

```
#include <speedbee.h>
int sdts_unset_win(sdtsdb_t db, sdtscid_t cid, char *dname)
```

**【引数】**

sdtsdb_t db	...	DB 構造体ポインタ
sdtscid_t cid	...	カラム ID
char *dname	...	ウィンドウ定義名

**【戻り値】**

0	:	正常終了
-1	:	異常終了

**【説明】**

カラムに設定された指定の名前のウィンドウ定義を削除します。

### 6.7.3.sdts\_drop\_win ()・・・リアルタイム処理用のウィンドウ定義の全削除

**【書式】**

```
#include <speedbee.h>
int sdts_drop_win(sdtsdb_t db, sdtscid_t cid)
```

**【引数】**

sdtsdb_t db	...	DB 構造体ポインタ
sdtscid_t cid	...	カラム ID

**【戻り値】**

0	:	正常終了
-1	:	異常終了

**【説明】**

指定カラムに設定されたすべてのウィンドウ定義をすべて削除します。

## 6.8.カラムデータ取得 API

### 6.8.1.spcf\_get\_col\_dat () …カラムデータ取得

#### 【書式】

```
#include <sdfs_spcf.h>
int spcf_get_col_dat(sdfsdb_t db, sdfs_cid_t cid, sdfs_cdopt_t opt, uint64_t ts, dat_nxt_t
*nxt, dat_hdr_t *hdr, uint8_t *buf, int32_t bsz, int32_t *ocnt)
```

#### 【引数】

sdfsdb_t db	…	DB 構造体ポインタ
sdfs_cid_t cid	…	カラム ID
sdfs_cdopt_t opt	…	検索オプション。以下、指定可能。 SDTS_STO_OPT_MEM   メモリのみを検索 SDTS_STO_OPT_FILE   ファイルのみを検索 SDTS_STO_OPT_MEM   SDTS_STO_OPT_FILE ファイル・メモリの両方の検索※詳しくは【SDTS_STO_OPT_MEM   SDTS_STO_OPT_FILE 指定】を参照 SDTS_CAC_OPT_SEA_RES   一度、サーチしたファイルパスをキ ャッシュする※詳しくは【ファイル検索キャッシュ】を参照 オプション指定がない場合 SDTS_STO_OPT_MEM 指定と同意
uint64_t ts	…	検索条件の通算ナノ秒 0 を指定した場合、2 回目以降の検索で次回データ情報として 次の引数 dat_nxt_t *nxt を使うことを意味する。
dat_nxt_t *nxt	…	次の検索情報。 API 実行時に次の実行で取得するデータ位置に関する内部情報を 以下の構造体に格納し返す。連続して実行する場合、この情報をも とに次回以降のデータを取得することで高速にデータを取得する。 そのため、API 利用者はこの構造体に対して、値をセットはしてはいけ ない。 typedef struct { uint64_t    t; uint64_t    m; uint16_t    n; uint16_t    p; uint32_t    s; } dat_nxt_t;
dat_hdr_t *hdr	…	取得したデータ情報。 取得したデータの情報を以下の構造体に格納し返す。 typedef struct { int            smpl;    サンプリングレート uint64_t       ts;     先頭データのタイムスタンプ } dat_hdr_t;
uint8_t *buf	…	データバッファのポインタ データバッファは API 利用側で領域を用意する。
int32_t bsz	…	データバッファのサイズ
int32_t *ocnt	…	取得データ件数の指定

## 【戻り値】

SPCF_SUCCESS(1)	:	正常に取得。次回もデータ取得可能。
SPCF_ERROR(-1)	:	異常終了
SPCF_ERR_OUT_RANGE(0)	:	検索条件に指定された時間のデータは DB に存在しない
SPCF_MISS_DATA(2)	:	正常に取得。次回データは、今回の連続したデータではない。(取得した最終データの後に時間が空いている。)
SPCF_END_MEM(3)	:	メモリ DB 上の最新値に到達。
SPCF_END_RATE(4)	:	正常に取得。同一サンプルレートの末尾に到達。
SPCF_END_FILE(5)	:	永続化(ファイル)DB 上の最新値に到達

## 【説明】

永続化 DB およびメモリ DB に格納された指定した **Mi タイプ** のカラムデータを取得します。

連続して次データを取得する場合、引数 ts に 0 を指定し、初回に dat\_nxt\_t に設定された次データ情報を使用し、高速にデータを取得します。

・データの一部が存在しない場合の出力

データの一部が存在しない区間が含まれる場合、専用の戻り値を返却します。

以下は、出力バッファを 10×8 の 80Byte にした場合の配列イメージです。

1 回目)

戻り値： SPCF\_MISS\_DATA

出力：hdr:ヘッダの情報 + 配列先頭の時間、ocnt：6, next: 次のデータ位置情報（次にデータが存在する場合）

値 1	値 2	値 3	値 4	値 5	値 6	-	-	-	-
-----	-----	-----	-----	-----	-----	---	---	---	---

2 回目)

戻り値： SPCF\_SUCCESS

出力：hdr:ヘッダの情報 + 配列先頭の時間、ocnt：10, next: 次のデータ位置情報

値1	値2	値3	値4	値5	値6	値7	値8	値9	値10
----	----	----	----	----	----	----	----	----	-----

・サンプルレートが変更されたデータの場合

途中でサンプリングレートが変更された場合、戻り値として SPCF\_END\_RATE を返します。

以下は、出力バッファを 10×8 の 80Byte にした場合の配列イメージです。

1 回目)

戻り値： SPCF\_END\_RATE

出力：hdr:ヘッダ A の情報と配列先頭の時間、ocnt：6, next: 次のデータ位置情報（次にデータが存在する場合）

値 1	値 2	値 3	値 4	値 5	値 6	-	-	-	-
-----	-----	-----	-----	-----	-----	---	---	---	---

2 回目)

戻り値： SPCF\_SUCCESS

値1	値2	値3	値4	値5	値6	値7	値8	値9	値10
----	----	----	----	----	----	----	----	----	-----

出力：hdr:ヘッダ B の情報と配列先頭の時間、ocnt：10, next: 次のバッファ先頭時間

## 【使用例】連続検索 (抜粋)

```

db = sdts_open_db("DB_PATH=tsdb2");
ts = 1000000000ULL;
opt = SDTS_STO_OPT_FILE;
cid = 1;
while (1) {
    ret = spcf_get_col_dat(db, cid, opt, ts, &nxt, &hdr, buf, BUF_SIZ, &outc);
    if (ret == SPCF_ERROR) {
        printf("SPCF_ERROR [%d]¥n", sd_get_err()); break;
    } else if (ret == SPCF_ERR_OUT_RANGE) {
        printf("SPCF_ERR_OUT_RANGE¥n"); break;
    } else {
        // SPCF_SUCCESS SPCF_MISS_DATA SPCF_END_MEM SPCF_END_RATE
        int *p = (int *)&buf[0];
        for (i = 1; i <= outc; i++)
            printf("[%d][%d]¥n", ++cnt, *p++); // data 4 byte integer
    }
    if (ret == SPCF_END_MEM) break;
    ts = 0;
}
(void)sdts_close_db(db);

```

## 【SDTS\_STO\_OPT\_MEM | SDTS\_STO\_OPT\_FILE 指定】

検索オプションにファイルとメモリを両方検索する SDTS\_STO\_OPT\_MEM | SDTS\_STO\_OPT\_FILE を指定し、メモリとファイルの両方に取得対象データが存在する場合は、メモリ優先でメモリから取得します。

## 【ファイルキャッシュ】

ファイル検索指定(SDTS\_STO\_OPT\_FILE)を有効にした場合、1 度読み込んだファイルは、キャッシュされます。保持可能な最大キャッシュ数は、DB パラメータ READ\_MAX\_CACHE に指定した数です。(デフォルト 2) それを超えた場合、古いキャッシュから解放されます。

**spcf\_get\_col\_dat()でファイルキャッシュを使用する場合、1 カラムのみの取得であるため、カレントのファイルと次のファイルの 2 つのキャッシュがあれば足りませんが、sdts\_open\_cur()で使用する場合は、参照カラム数やカラムタイプにより、2 つでは不足する可能性があります。この場合は、DB パラメータ READ\_MAX\_CACHE のキャッシュ数を増やします。ただし、ファイルキャッシュ数を増やした場合、それだけのメモリリソースを消費することに留意する必要があります。**

ファイルキャッシュは、使用スレッド専用です。これは性能優先しているためスレッド間での共有はされません。また、使用したキャッシュは、同一スレッドにおいて、DB クローズするまでは永続的に使用することができます。キャッシュの解放は、DB クローズ時に自動解放されますが、spcf\_clean\_col\_dat() API を実行することで、実行したスレッドのキャッシュを明示的に解放することができます。

ファイルキャッシュ数を増やした場合、それだけのメモリリソースを消費することに留意する必要があります。



**【ファイルパス検索結果キャッシュ】**

ファイルパス検索結果キャッシュ(SDTS\_CAC\_OPT\_SEA\_RES)指定が有効で、かつファイル読み込み設定(SDTS\_STO\_OPT\_FILE)が有効である場合、1度、検索したファイルパスをキャッシュし、次回、同じ条件の時にキャッシュしたパスを使用します。これはストレージIOが遅いマシンの対応で、条件に一致するファイルの検索処理の性能改善効果があります。ストレージIOが速いマシンでは、基本的には不要です。

**【留意事項】**

spcf\_get\_col\_dat は、カラムタイプが M タイプのみを扱うことができます。(H,L タイプは、対応中)

**6.8.2.spcf\_clean\_col\_dat () …検索リソースの解放****【書式】**

```
#include <sdts_spcf.h>
void spcf_clean_col_dat(sdtsdb_t db)
```

**【引数】**

sdtsdb_t db	...	DB 構造体ポインタ
-------------	-----	------------

**【戻り値】**

なし

**【説明】**

spcf\_get\_col\_dat()を使用し、ファイル検索した場合、DB ライブラリ内部においてファイルキャッシュやファイルパス検索結果キャッシュを保持することがあります。これらの内部で管理されている検索リソースは、DB クローズ時に解放しますが、明示的に即時に解放したい場合にこの API を使用します。

## 6.9.情報取得 API

### 6.9.1.sdts\_get\_db\_info () ... DB 情報取得

#### 【書式】

```
#include <speedbee.h>
sdtsdbinfo_t *sdts_get_db_info(sdtsdb_t db);
```

#### 【引数】

sdtsdb_t db	...	DB 構造体ポインタ
-------------	-----	------------

#### 【戻り値】

NULL 以外	:	正常終了 - DB 情報構造体ポインタ
NULL	:	異常終了

#### 【説明】

DB 情報を取得します。

DB 情報は、以下の DB 情報構造体 sdtsdbinfo\_t に格納され、戻り値として取得します。

取得した DB 情報は、sdts\_get\_db\_info()内でメモリアロケートした領域であるため、参照後、sdts\_free\_db\_info() API で確保された領域を解放する必要があります。

#### ・DB 情報構造体 sdtsdbinfo\_t

typedef struct	名前	説明
{		
char	*dbpar	DB パラメータ (システム辞書に保存した DB パラメータで作成時に設定したものとは異なる)
char	*path;	DB パス DB をストレージ保存した場合のみ
int	idsz;	バイナリ ID サイズ(バイト)
int	ccnt	カラム数
int	sbsz	ファイル同期書き込みバッファサイズ
int	rmaxccnt	最大ファイルキャッシュ数
int	rmaxccnt2	最大サーチ結果キャッシュ数
int	scnt_h	Hi ファイル書き込むカラム数
int	scnt_m	Mi ファイル書き込むカラム数
int	scnt_l	Lo ファイル書き込むカラム数
int	sper_h	Hi ファイル同期データ期間(秒)
int	sper_m	Mi ファイル同期データ期間(秒)
int	sper_l	Lo ファイル同期データ期間(秒)

int	lag_h	Hi ファイル同期ラグ時間(秒)
int	lag_m	Mi ファイル同期ラグ時間(秒)
int	lag_l	Lo ファイル同期ラグ時間(秒)
int	maxf_h	Hi ファイル同期最大書き込み数
int	maxf_m	Mi ファイル同期最大書き込み数
int	maxf_l	Lo ファイル同期最大書き込み数
ui64_t	stcnt	ファイル同期実行回数
sdtscolinfo_t	**colinfo	DB 内のすべてのカラムのカラム情報ポインタ配列 カラム情報の構造体(sdtscolinfo_t)は、 6.9.3.sdts_get_col_info()を参照
} sdtsdbinfo_t;		

### 6.9.2.sdts\_free\_db\_info () ... DB 情報領域の解放

#### 【書式】

```
#include <speedbee.h>
void sdts_free_db_info(sdtsdbinfo_t *di)
```

#### 【引数】

sdtsdbinfo_t *di	...	DB 情報構造体ポインタ
------------------	-----	--------------

#### 【戻り値】

なし

#### 【説明】

sdts\_get\_db\_info()で取得した DB 情報領域を解放します。

sdts\_get\_db\_info()で取得した DB 情報は、メモリアロケートした領域であるため、参照後、この API により領域を解放する必要があります。

### 6.9.3.sdts\_get\_col\_info () … カラム情報の取得

#### 【書式】

```
#include <speedbee.h>
sdtscolinfo_t *sdts_get_col_info(sdtsdb_t db, sdtsid_t cid)
```

#### 【引数】

sdtsdb_t db	…	DB 構造体ポインタ
sdtsid_t cid	…	カラム ID

#### 【戻り値】

NULL 以外	:	正常終了 - カラム情報構造体
NULL	:	異常終了

#### 【説明】

指定のカラム情報を取得します。

カラム情報は、以下のカラム情報構造体 sdtscolinfo\_t に格納されます。

取得したカラム情報は、sdts\_get\_col\_info()内でメモリアロケートした領域であるため、参照後、sdts\_free\_col\_info() API で確保された領域を解放する必要があります。

・カラム情報構造体 sdtscolinfo\_t

typedef struct {	名前	タイプ※ 1	説明
sdid_t	cname;	共通	ユーザー指定のカラム識別子 ※sdts_create_col() cname に指定した文字列またはバイナリ
int	ctype;	共通	カラムタイプ SDTS_CT_HI_FIX(1)     Hi タイプ SDTS_CT_MI_FIX(11)   Middle タイプ SDTS_CT_LO_FIX(101)   Lo 固定長 タイプ SDTS_CT_LO_VAR(102)   Lo 可変長タイプ
int	dsz;	共通	固定長データサイズ(バイト) ※LO 可変長タイプは 0
int	rsz	共通	メモリ上での 1 レコードサイズ Hi, Mi は、データサイズ (dsz)と同じ Lo 固定長は、タイプスタンプ(8) + 固定長データサイズ Lo 可変長は、
int	scnt	共通	メモリ保存件数
ui64_t	icnt	共通	sdts_open_db()後からの登録件数
bool	act	共通	カラムアクティブフラグ true アクティブ

char	*dtype	共通	カラムデータタイプ文字列
char	*colpar	共通	カラムパラメータ文字列
sdntime_t	msst	共通	メモリ内に格納されている先頭データのタイムスタンプ
sdntime_t	mset	共通	メモリ内に格納されている最終データのタイムスタンプ
ui64_t	mscnt	共通	メモリ内に格納されているデータ件数
double	hmsmpl	H,M	サンプリングレート
sdntime_t	hmst	H,M	開始タイムスタンプ
int	mmmax	M	M タイプ管理領域の最大件数
int	mmspos	M	M タイプ管理領域の開始位置
int	mmucnt	M	M タイプ管理領域の使用件数
ui64_t	micnt	M	M タイプのカレント登録件数※2
sdtscolmminfo_t	*mminfo	M	M タイプ管理情報配列※3
int	wdcnt	共通	ウィンドウ定義数
sdtscolwinfo_t	*winfo	共通	ウィンドウ定義情報配列※4
} sdtscolinfo_t;			

※1 タイプは、取得可能なカラムタイプを示しています。「共通」は、すべてのカラムタイプで取得ができます。

※2 M タイプは、複数のデータの登録管理を持ち、micnt は、現在使用している管理情報を取得します。

※3 mminfo は、カラムが M タイプの場合にメモリ上のすべての M タイプの管理情報を M タイプ管理情報構造体の配列として取得します。配列件数は、M タイプ管理領域の使用件数 mmucnt になります。  
また、簡易版の情報取得 API sdt\_get\_col\_simp\_info()では取得しません。

※4 winfo は、カラムにウィンドウ定義が設定されている場合にウィンドウ定義情報をウィンドウ定義情報構造体の配列として取得します。配列件数は、ウィンドウ定義数 wdcnt になります。  
また、簡易版の情報取得 API sdt\_get\_col\_simp\_info()では取得しません。

#### ・M タイプ管理情報構造体 sdtscolmminfo\_t

typedef struct {	名前	タイプ	説明
double	smpl;	M	サンプリングレート
sdntime_t	st	M	開始タイムスタンプ
sdntime_t	et	M	終了タイムスタンプ
ui64_t	micnt	M	登録件数※1
ui64_t	mdcnt	M	削除件数
ui64_t	mscnt	M	メモリ内の件数※1
}			
sdtscolmminfo_t			

※1 micnt と mscnt の違い

micnt は、この管理定義での登録件数になります。mscnt は、現在、メモリ上に存在する件数で、上書きされた場合、件数は減らされます。

## ・ウィンドウ定義情報構造体 sdtscolwinfo\_t

typedef struct	名前	タイプ	説明
{			
char	name	共通	ウィンドウ定義名
int	wtype	共通	ウィンドウタイプ SD_WT_COUNT (2)      カウントウィンドウ SD_WT_FFT(4)      FFT ウィンドウ
uchar	ftype	共通	ウィンドウアクションタイプ※1 SD_WF_WAG(01)      集計処理 SD_WF_NCF(04)      通知処理
}			
sdtscolwinfo_t			

※1 ウィンドウファンクションタイプは、複数定義することができるため、例えば集計処理と通知処理の場合は、05 が取得されます。

## 【留意点】

取得したカラム情報は、sdts\_free\_col\_info() で領域を解放する必要があります。

## 6.9.4.sdts\_free\_col\_info () … カラム情報領域の解放

## 【書式】

```
#include <speedbee.h>
void sdts_free_col_info(sdtscolinfo_t *ci)
```

## 【引数】

sdtscolinfo_t *ci	…	カラム情報構造体ポインタ
-------------------	---	--------------

## 【戻り値】

なし

## 【説明】

sdts\_get\_col\_info()で取得したカラム情報領域を解放します。

sdts\_get\_col\_info()で取得したカラム情報は、メモリアロケートした領域であるため、参照後、この API により領域を解放する必要があります。

### 6.9.5.sdts\_get\_col\_simp\_info () …簡易版カラム情報の取得

**【書式】**

```
#include <speedbee.h>
int sdts_get_col_simp_info(sdtsdb_t db, sdtsid_t cid, sdtscolinfo_t *ci)
```

**【引数】**

sdtsdb_t db	...	DB 構造体ポインタ
sdtsid_t cid	...	カラム ID
sdtscolinfo_t *ci	...	カラム情報構造体ポインタ

**【戻り値】**

0	:	正常終了
-1	:	異常終了

**【説明】**

カラム情報を取得します。

カラム情報は、引数 ci に指定したカラム情報構造体 sdtscolinfo\_t (5.3.6 sdts\_get\_col\_info() カラム情報構造体を参照) に格納されます。カラム情報取得の API には、sdts\_get\_col\_info()とこの簡易版の sdts\_get\_col\_simpl\_info()があります。違いは、以下の2点になります。

- ・sdts\_get\_col\_info()は、M タイプ のすべての管理領域情報とウィンドウ定義情報を返しますが、簡易版では、これらの情報を取得しません。

- ・sdts\_get\_col\_info()は、取得したカラム情報構造体を sdts\_free\_col\_info()で解放しなければなりませんが、簡易版では、呼び出し側のカラム情報構造体の領域を渡すため、sdts\_free\_col\_info()の必要がありません。

M タイプ のすべての管理領域情報やウィンドウ定義情報が不要である場合は、こちらの簡易版の sdts\_get\_col\_simpl\_info()を利用することで情報収集の性能向上およびメモリ解放の煩わしさがなくなります。

## 6.10.エラー API

### 6.10.1.sd\_get\_err() … エラーコードの取得

---

**【書式】**

```
#include <speedbee.h>
int sd_get_err(void)
```

**【引数】**

なし

**【戻り値】**

エラーコード

**【説明】**

内部で設定されたエラーコードを返します。  
エラーコードに関しては「7.2 エラーコード」を参照してください。

### 6.10.2.sd\_clear\_err() … エラーコードのクリア

---

**【書式】**

```
#include <speedbee.h>
void sd_clear_err(void)
```

**【引数】**

なし

**【戻り値】**

なし

**【説明】**

内部のエラーコードをクリアします。

**【補足】**

内部においてエラー情報はスレッド単位に管理されており、これらの管理にローカルスレッドメモリを利用しています。Linux/Windows においては、ローカルスレッドメモリ機能は OS 標準でサポートされているため、各 API 内において自動でエラーコードをクリアする処理が含まれており、この API を呼び出す必要はありませんが、ローカルスレッドメモリをサポートしないプラットフォームでは、スレッドメモリ単位のメモリ領域へのアクセスはコストのかかる処理であり、性能に影響が出るため、デフォルトでは API ごとのエラーコードクリア処理を行うようにはしていません。そのため、このような環境では sd\_clear\_err() API を使用しエラーコードをクリアします。コールするタイミングは、DB 処理の開始やエラー時のエラーコード参照後にコールします。(ただし、エラークリア処理は負荷がかかるため、大量なコールは避けてください。)

負荷がかかっても API による自動クリアしたい場合は、環境変数 ECI\_ERROR\_CLEAR\_SELF を false に設定してください。



## 7.エラー

時系列 DB 用の SDTS-API のエラー発生時、include/eci\_error.h に定義されたエラーコードを返します。

### 7.1.エラー種別

include/eci\_error.h は、Speedbee のすべての API 共通のエラーコードヘッダーファイルで、大きく分けると以下の 4 つの種類になります。

#	種別	エラー接頭辞	コード範囲	説明
1	システムエラー	ERR_SYS_	1 - 9999	システム関係のエラー(システムコールエラー)
2	インターナルエラー	ERR_INT_	10000-99999	インターナルエラーは通常の利用では発生しないエラーで、原因はデータファイル破損、内部バグ、ユーザーの操作の誤りによるエラーになります。  eci_error.h 内では内部レイヤごとに定義されていますが、これらのエラーは内部ロジックや構造に関係し、エラーの説明が難しいため省略します。
3	ユーザーエラー	ERR_USR_	100000-199999	ユーザー操作の誤りによるエラー
4	アプリケーション API エラー	ユーザー定義	200000 - 299999	ユーザー定義によるアプリケーション API のエラーコード範囲

※上記以外のエラータイプも存在しますが、時系列 DB とは関係ないため省略します。

## 7.2.エラーコード

時系列 DB 用の SDTS-API のエラーコードの範囲は以下になります。

#	種別	エラー接頭辞	コード範囲
1	インターナルエラー	ERR_INT_SDTS_	82000 - 82999
2	ユーザーエラー	ERR_USR_SDTS_	182000 -182999

SDTS API エラーコード内容は以下になります。

### 1) インターナルエラー

エラーコード	番号	エラー内容
ERR_INT_SDTS_SPEC	82001	内部エラー
ERR_INT_SDTS_HISM_INVALID	82002	内部エラー
ERR_INT_SDTS_CTYPE_INVALID	82003	内部エラー
ERR_INT_SDTS_MM_INVALID	82004	内部エラー
ERR_INT_SDTS_PAR_BUF_NOT_ENOUGH	82005	内部エラー
ERR_INT_SDTS_SYNC_ADJUST_LAST_TIME	82006	内部エラー sdts_adjust_ts()コール時のエラー
ERR_INT_SDTS_INVALID_DATA_FILE	82007	データファイル破損
ERR_INT_SDTS_COL_PAR_BROKEN	82008	システムディクショナリ破損
ERR_INT_SDTS_REMOVE_DIR	82009	チェック済みのディレクトリ確認エラー

### 2) ユーザーエラー

エラーコード	番号	エラー内容
ERR_USR_SDTS_ARG_DB_PAR	182001	API 引数 DB パラメータが指定されていない 対象 API:sdts_drop_db
ERR_USR_SDTS_ARG_COL_PAR	182002	API 引数 COLUM パラメータが指定されていない 対象 API:sdts_create_col
ERR_USR_SDTS_ARG_DB_DESC	182003	API 引数 DB ディスクリプタが指定されていない 対象 API:DB ディスクリプタを引数とする API
ERR_USR_SDTS_ARG_DUP_KID	182004	API 引数 カラム名（またはカラム識別 ID）と同じカラムがすでに存在する。 対象 API: sdts_create_col
ERR_USR_SDTS_ARG_COL_ID	182005	API 引数 指定カラム ID は DB 内に存在しない。 対象 API: カラム ID を引数とする API
ERR_USR_SDTS_ARG_INS_DAT	182006	API 引数 データポインタ(NULL)、データ件数(0 以下)の不正 対象 API: sdts_insert
ERR_USR_SDTS_ARG_INS_DSZ	182007	API 引数 データサイズ(0 以下、LV_MAX 以上)の不正 対象 API: sdts_insert Low 可変長データ登録
ERR_USR_SDTS_ARG_KEY_ID	182008	API 引数 カラム名（またはカラム識別 ID）が不正 (NULL) 対象 API: カラム名（またはカラム識別 ID）を引数とする API
ERR_USR_SDTS_ARG_GET_BUF	182009	API 引数 カラムデータ格納バッファ(NULL)かバッファサイズ (0 以下)が不正 対象 API: sdts_get_col_dat, spcf_get_col_dat

ERR_USR_SDTS_ARG_CND_ST_ET	182010	API 引数 検索条件時間指定の不正 st または et が 0、st のほうが et より大きい 対象 API: sdts_open_cur、sdts_get_col_dat
ERR_USR_SDTS_ARG_CID_LST	182011	API 引数 カラム ID 配列(NULL)、カラム ID 件数(0 以下)が不正 対象 API: sdts_open_cur
ERR_USR_SDTS_ARG_CUR_DESC	182013	API 引数カーソルディスクリプタが不正 対象 API:カーソルディスクリプタを引数とする API
ERR_USR_SDTS_ARG_CUR_VP	182014	API 引数 カーソルカラムデータ取得バッファが不正(NULL) 対象 API: sdts_get_cur_dat
ERR_USR_SDTS_ARG_CUR_AGGR_RTS	182015	API 引数 集約カーソルタイムスタンプ取得バッファが不正(NULL) 対象 API: sdts_get_cur_aggr
ERR_USR_SDTS_ARG_CUR_AGGR_RSVP	182016	API 引数 集約カーソルカラムデータ取得バッファ配列(NULL)または、バッファ配列数(0)が不正 対象 API: sdts_get_cur_aggr
ERR_USR_SDTS_ARG_SMPL	182017	API 引数サンプリングレートが不正 (0.0) 対象 API: sdts_set_smpl_rate
ERR_USR_SDTS_ARG_GET_TS	182018	API 引数 タイムスタンプ取得バッファポインタが不正(NULL) 対象 API: sdts_get_col_lval
ERR_USR_SDTS_ARG_MI_NXT	182019	API 引数 検索情報バッファポインタが不正(NULL) 対象 API: spcf_get_col_dat
ERR_USR_SDTS_ARG_MI_HDR	182020	API 引数 データ情報バッファポインタが不正(NULL) 対象 API: spcf_get_col_dat
ERR_USR_SDTS_ARG_MI_OCNT	182021	API 引数 取得データ件数バッファポインタが不正(NULL) 対象 API: spcf_get_col_dat
ERR_USR_SDTS_ARG_MIN	182022	API 引数 データ保存時間が不正 (0 以下) 対象 API: sdts_delete_db
ERR_USR_SDTS_ARG_COL_CNT	182023	API 引数 カラム数の指定が不正 (0 以下) 対象 API: sdts_create_col_bat
ERR_USR_SDTS_PAR_COL_TYPE	182101	API 引数 カラムパラメータにカラムタイプ(COL_TYPE)が指定されていない 対象 API: sdts_create_col
ERR_USR_SDTS_PAR_SMPL	182102	API 引数 カラムパラメータにサンプリングレート(SML_RATE)が指定されていない(Hi 限定) 対象 API: sdts_create_col
ERR_USR_SDTS_PAR_DSZ	182103	API 引数 カラムパラメータにデータサイズ (DATA_SIZE)が指定されていない(固定長データ) 対象 API: sdts_create_col
ERR_USR_SDTS_PAR_MCNT	182105	API 引数 カラムパラメータに保存件数(SAVE_COUT)が指定されていないか 1 以下 対象 API: sdts_create_col
ERR_USR_SDTS_PAR_LV_MAX	182106	API 引数 カラムパラメータ LV_MAX が指定された場合の値が不正 (65536 以上) 対象 API: sdts_create_col
ERR_USR_SDTS_PAR_SYNC_PERIOD	182107	API 引数 DB パラメータ SYNC_PERIOD_H, SYNC_PERIOD_M, SYNC_PERIOD_L の値不正 (0 以下か 60 を割ったときにあまりがある場合) 対象 API: sdts_open_db
ERR_USR_SDTS_DB_NOT_DIR	182200	API 引数 DB パラメータ DB_PATH に指定された値のディレクトリが存在しない 対象 API: sdts_open_db, sdts_drop_db
ERR_USR_SDTS_ID_NOT_FOUND	182202	API 引数 カラム名に指定したカラムが DB に見つからない。

		(ERR_USR_SDTS_ARG_COL_ID とは異なる。カラム名とカラム ID) 対象 API: カラム ID を引数とする API
ERR_USR_SDTS_BUF_NOT_ENOUGH	182204	カラムデータ取得 API においてカラムデータバッファが足りない 対象 API: sdts_get_col_dat
ERR_USR_SDTS_CUR_NOT_FETCH	182205	シリアルカーソルカラムデータ取得において、引数に指定されたカーソルはフェッチされていない 対象 API: sdts_get_cur_dat
ERR_USR_SDTS_CUR_NOT_COLUMN	182206	シリアルカーソルカラムデータ取得において、引数に指定されたカーソルは集約カーソルが渡された 対象 API: sdts_get_cur_dat
ERR_USR_SDTS_CUR_NOT_AGGR	182207	集約カーソルカラムデータ取得において、引数に指定されたカーソルはシリアルカーソルが渡された 対象 API: sdts_get_cur_aggr
ERR_USR_SDTS_MM_LIMIT_COUNT	182208	タイプ M カラムで、管理領域不足 対象 API: sdts_insert
ERR_USR_SDTS_NO_SET_TS	182209	タイプ M カラムで管理アイテムが存在しない タイムスタンプの指定がない 対象 API: sdts_insert
ERR_USR_SDTS_NO_SET_SAMPL	182210	タイプ M カラムでサンプリングレート指定がされていない。 対象 API: sdts_insert
ERR_USR_SDTS_INVALID_TIMESTAMP	182211	タイプ M カラムで指定されたタイムスタンプが不正。 対象 API: sdts_insert
ERR_USR_SDTS_ALREADY_USE_CID	182212	カラム作成でカラム ID を直接指定したが、すでに同じカラム ID が存在し、使われている。 対象 API: sdts_create_col
ERR_USR_SDTS_SPEC_COL_ID	182213	カラム作成でカラム ID を直接指定したが、指定したカラム ID の値が不正 (0 以下か最大カラム作成可能数を越える) 対象 API: sdts_create_col
ERR_USR_SDTS_DAT_OVERWRITE	182214	参照データが登録により上書きされた 対象 API: sdts_fetch_cur, spcf_get_col_dat
ERR_USR_SDTS_CTYPE_INVALID	182215	データ参照対象カラムタイプが不正 対象 API: sdts_get_col_dat, sdts_fetch_cur
ERR_USR_SDTS_COL_INACTIVE	182217	登録および参照、管理系 API で対象カラムがアクティブではない 対象 API: 登録および参照、管理系 API
ERR_USR_SDTS_SYNC_WBUF_NOT_ENOUGH	182221	同期 API で、書き込みバッファが不足している。 対象 API: sdts_sync_db
ERR_USR_SDTS_SYNC_ALREADY_CLOSE	182222	同期設定されているカラムにおいて、データ登録したが同期 API の実行していないか、または追いついていない。 対象 API: sdts_insert
ERR_USR_SDTS_END_OF_DATA	182223	spcf_get_col_dat API でデータの最後まで参照しているが、それ以上に参照しようとした 対象 API: spcf_get_col_dat
ERR_USR_SDTS_NOT_WRITE_COL	182224	spcf_get_col_dat API でデータファイルを参照しようとしたがデータ書き込み設定がされているカラムが存在しない 対象 API: spcf_get_col_dat
ERR_USR_SDTS_INVALID_DATA_FILE	182225	データファイルの参照で次のファイルを探したが、ファイルに問題があり探すことができなかった。 対象 API: spcf_get_col_dat
ERR_USR_SDTS_CUR_NO_END_POS	182227	sdts_extend_cur API の引数カーソルが SD_FETCH_END 状態のカーソルではない。 対象 API: sdts_extend_cur

ERR_USR_SDTS_CUR_NO_EXTEND	182228	sdts_extend_cur API の引数カーソルが、カーソルオープン時に SDTS_CUR_OPT_EXTEND オプションが指定されていない。 対象 API: sdts_extend_cur
ERR_USR_SDTS_READ_CACHE_ALL_USE	182229	データファイル参照でリードキャッシュがすべて使用済みで空かない 対象 API: spcf_get_col_dat
ERR_USR_SDTS_SYNC_OVERWRITE_DATA	182230	同期 API で、メモリ上の書き込むべきデータが登録により上書きされた 対象 API: sdts_sync_db
ERR_USR_SDTS_NOT_SUPPORT_COMP	182231	データ圧縮が使用されている DB ファイルの参照で、使用している DB ディストリビューションにおいて圧縮機能をサポートしていない。 対象 API: spcf_get_col_dat
ERR_USR_SDTS_NOT_PERSIST_DB	182233	永続化 DB ではない。
ERR_USR_SDTS_INVALID_SYSDb	182234	システム辞書が不正

SDTS DB API 6.2 リファレンスマニュアル 1.0 版

Copyright (C) 2020 SALTYSTER Inc. All rights reserved.

本書の記載内容の著作権は、株式会社ソルティスターに帰属します。

内容の全部か一部を問わず、著作権の許可なく転載、複製することを禁じます。