SQLite を用いた
組み込み機器向け M2M 複合イベント処理機能
外部仕様書

2014 年 7 月 23 日（水）

第 1.0 版

安田晃久

改定履歴

日付	内容
2014/03/14	・草稿作成
2014/04/10	・2 章ユースケースを追加 ・5 章インタフェースを修正
2014/07/23	・5 章インタフェースを実装内容に合わせて修正

目次

1. はじめに

本仕様書は，ネットワークを通じ，人手を介さず機器同士が相互通信を行うことでシステムが構築される「M2M」環境での動作を対象とする．本書では，この M2M 環境において，「複合イベント処理」（Complex Event Processing：刻々と発生し続ける機器データを素早く分析，判断し，処理するための技術．以下 CEP と表記する）を行うためのデータ処理ライブラリ機能の実装について述べる．

本仕様書記載のソフトウェアの対象ユーザ，及び対象環境を表 1-1 のように定義する．

表 1-1　本ソフトウェアの対象ユーザ，対象環境

項目			内容
対象ユーザ			CEP 初心者（但し，データベース／SQL に関する知識を有している方が望ましい）
対象環境	H/W		CPU ボード（Raspberry Pi[1]や BeagleBone Black[2]等）
		CPU	ARM v6[3]以上
		メモリ	256 [MB]以上
		ディスク	microSDHC I/F
	OS		GNU／Linux

なお，上記以外の環境，例えば CPU に AMD64 を用いた一般 PC に対しても適用可能であるが，CEP における有効性は，データストアとして SQLite を利用している関係上，NoSQL や RDBMS 等のソフトウェアを利用した場合と比較して低減する事を予め明記しておく．

[1] URL：http://www.raspberrypi.org/
[2] URL：http://beagleboard.org/Products/BeagleBone+Black
[3] URL：http://www.arm.com/ja/products/processors/classic/arm11/index.php

2. ユースケース

ここでは，本書で仕様を提示する M2M CEP 機能について述べる前に，そのユースケースとして，CEP 機能を組み込んだアプリケーションの処理を示す．

2.1. ストリームデータ

まず，CEP で取り扱うデータの特徴を説明する．

CEP で取り扱うデータは一般的に「ストリームデータ」と呼ばれており，その特徴は以下となる．

- 分類可能な複数種別のデータが連続的に生成（計測）される（毎回異なる種別のデータがバラバラに生成するのではなく，同じ種別のデータが連続生成）
- データ生成頻度はデータ種別に依存し，高頻度／低頻度のどちらもストリームデータに含まれる（高頻度だからストリームデータである，という訳ではない）
- データの時系列性に特徴があり，直近の過去データと比較すると，そのデータの持つ意味性を見出す事が出来る

ストリームデータはこれまでにも存在しており，新たにカテゴライズされ得る種別のデータではない．しかし，M2M 環境の進展によって，例えば計測データを生成するセンサ機器種別の増加，或いは機器台数の増加に伴い，“データサイズ”によってストリームデータの特徴が変化してきている．そのため，ストリームデータの取り扱い方を既存の方法から変更しないと，データの持つ意味を正確に把握する事が難しくなってしまう，という問題が新たに発生している．

こうした問題に対処するための方法の一つが CEP である．

2.2. CEP 機能を使ったアプリケーション例

では，M2M CEP 機能を使ったアプリケーションの一例として，「温湿度センサで計測した温湿度の変化に伴う空調システムの制御」を示す（図 2-1 参照）．

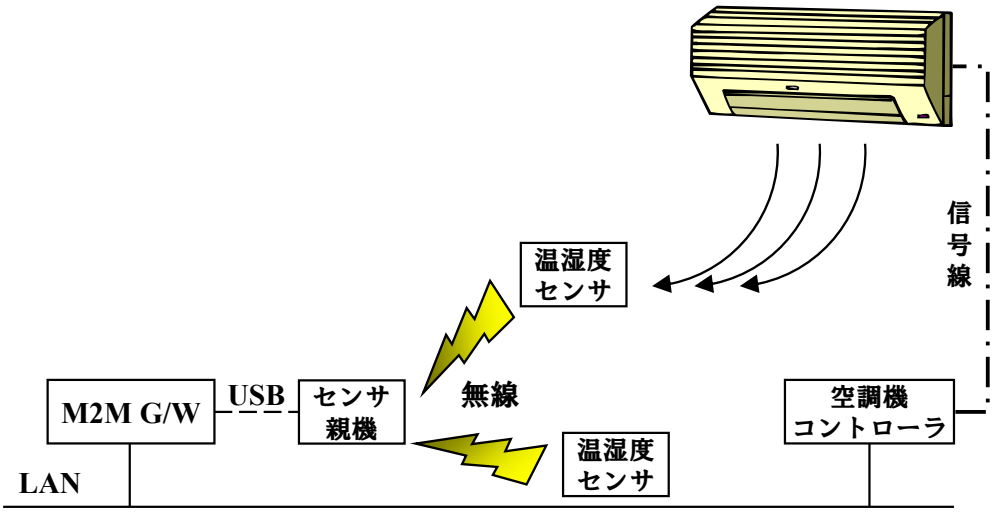

図 2-1　温湿度センサで計測した温湿度の変化に伴う空調システムの制御構成図

図 2-1 のような空調制御システムは既存の空調システムと同様の構成であるが（温湿度センサに該当するセンサが既存の空調システムの室内機の吸い込み口に設置されている），図 2-1 では，よりきめ細やかな制御を行うため，外部のセンサを利用する構成としている．

このシステムを用いたアプリケーション（M2M G/W 内で稼動するソフトウェア）の内部処理を図 2-2 に示す．

図 2-2 では，入力データとして，温湿度センサから取得した時系列の温湿度データを CSV 形式のファイルとして既定のディレクトリに設置しておくと，アプリケーションは当該データを読み取り，解析を行い，解析結果を CSV 形式のファイルとして出力する，というフローを示している（**3.1** に示すように，ストック型データ分析のため，解析サーバへセンサデータを送信する場合も存在する）．

解析処理では，例えば 10 分間毎の温度の平均値を計算し，その結果が，設定温度からの閾値である 0.5℃を超えた場合のみ，CSV ファイルに「この ID の温湿度センサの計測結果が閾値を超えた」という内容の情報を出力する．解析結果ファイルが出力された場合，室温を設定温度に近付けるための制御を実行すれば，この空間の室温を常に一定に保つ事が出来る．

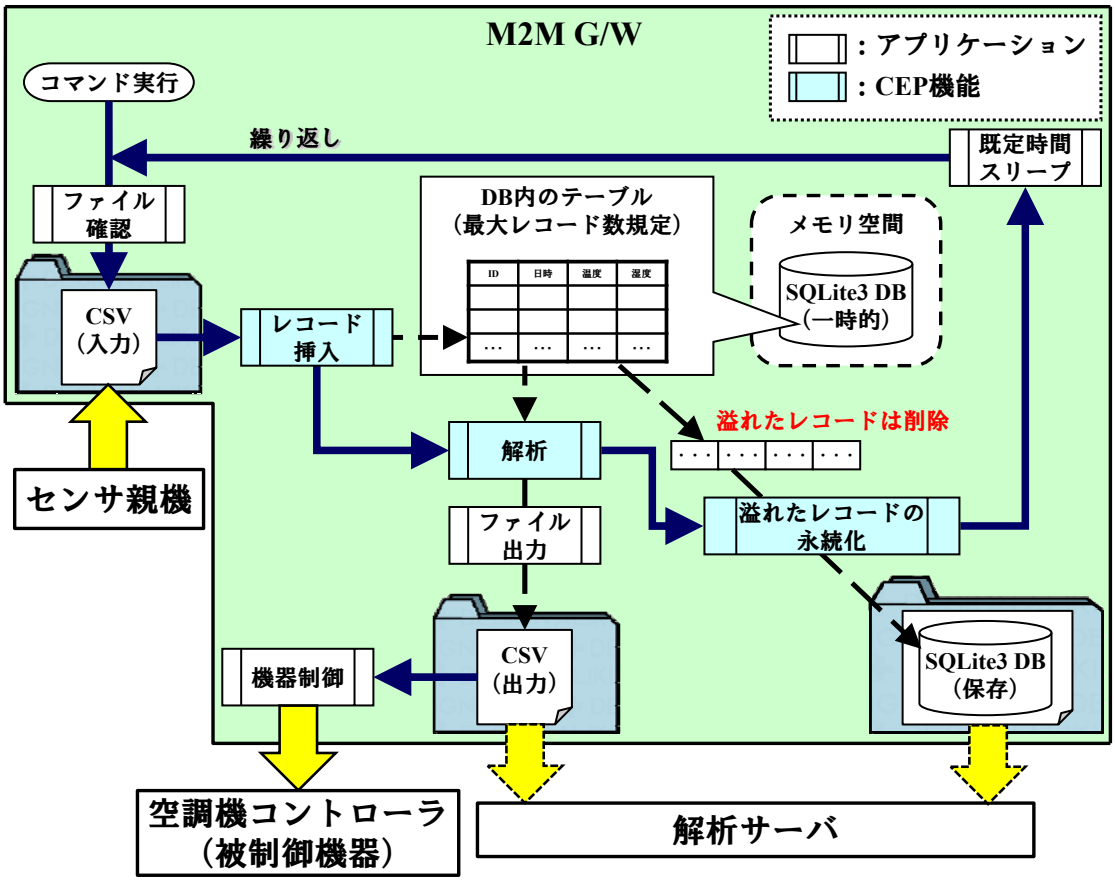

図 2-2　CEP 機能を組み込んだアプリケーション例

3. 構成

3.1. システム構成

本仕様を含む，M2M CEP システムの構成図を図 3-1 に，構成要素の説明を表 3-1 に示す．

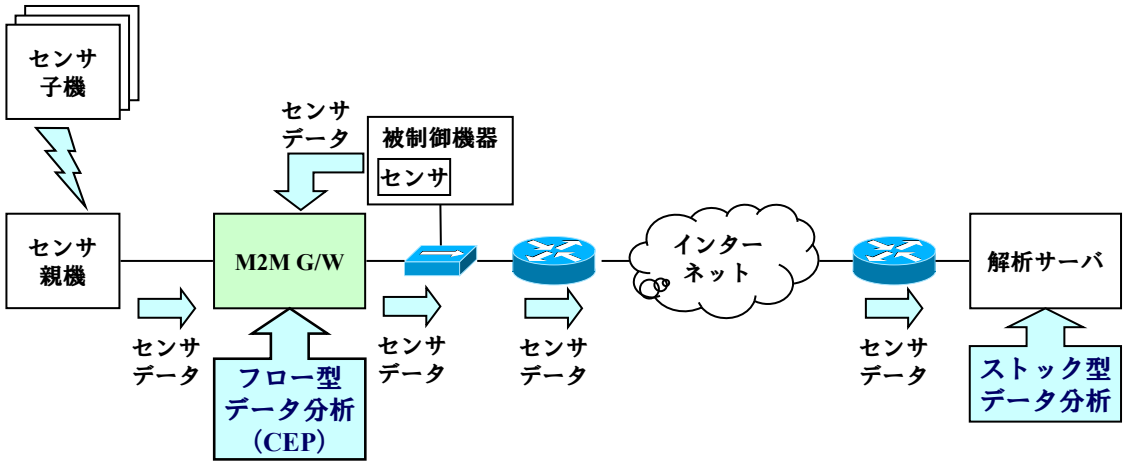

図 3-1　M2M CEP システム構成図

表 3-1　M2M CEP システム構成要素

要素名	内容
センサ子機	内臓されている各種センサデバイスを利用して環境データを計測する機器．計測データは無線通信を利用してセンサ親機へ定期的に送信する．
センサ親機	無線通信を通じてセンサ子機から送信されてくる計測データを収集する機器．USB 端子等を通じて接続された機器へ収集した計測データを渡す．
センサ	被制御機器が稼動するために必要な環境データを計測する機器．
被制御機器	内臓されているセンサデータをネットワーク上に接続された機器へ送信する．また，解析結果に基づいて稼動内容を変更するための制御命令を受信し，稼動内容を変更する．
M2M G/W	本仕様書記載の CEP 機能が稼動する組み込み機器．フロー型データ分析処理を実行し，指定された場所に記録する．また，解析サーバにおけるストック型データ分析実行のため，収集したセンサデータをインターネット経由で解析サーバへ送信する．
インターネット	TCP/IP で接続された広域網．
解析サーバ	M2M G/W から送信されてきたセンサデータを利用してストック型データ分析を実行するための機器．

3.2. ハードウェア／ソフトウェア構成

M2M G/W 内において，CEP を実行するために必要なハードウェア，及びソフトウェア構成を図 3-2，表 3-2，表 3-3 に示す.

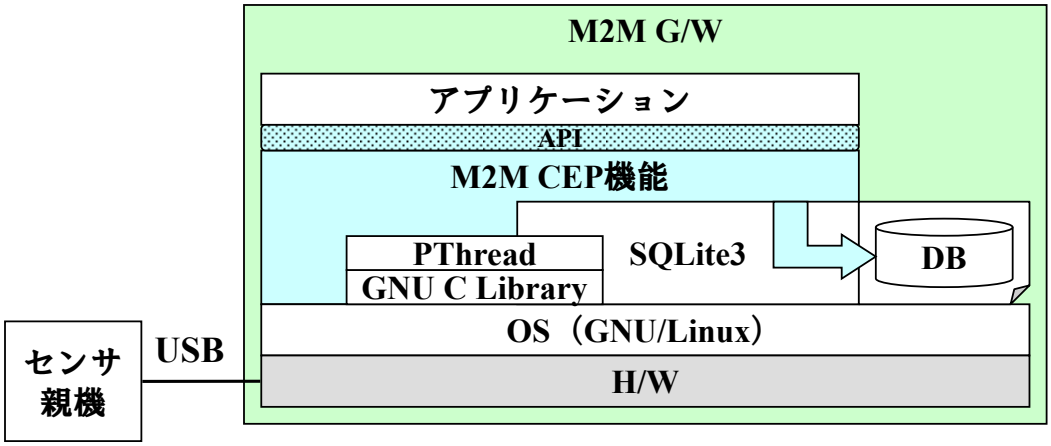

図 3-2　M2M CEP 機能ハードウェア／ソフトウェア構成図

表 3-2　M2M 複合イベント処理機能ハードウェア仕様

項目	内容
CPU	ARM v6 以上
メモリ	256 [MB]以上
ディスク	microSDHC I/F
インタフェース	USB 2.0/3.0

表 3-3　M2M 複合イベント処理機能ソフトウェア仕様

名称	バージョン	内容
GNU/Linux	3.10	GNU GPL v2.0 で利用可能な Linux カーネルと，GNU GPL v2.0/3.0 で利用可能なユーザランドソフトウェアで構成されるオープンソース OS.
libc.so	2.13	GNU LGPL で利用可能な標準 C ライブラリ.
libpthread.so	2.13	GNU LGPL で利用可能なスレッド処理用ライブラリ（libsqlite3.so が依存する）.
libsqlite3.so	0.8.6	パブリックドメインで提供される SQLite3 データベース処理用ライブラリ.
libcep.so	0.2.7	2 条項 BSD ライセンスで提供する，本仕様書記載内容で実装した M2M CEP 機能ライブラリ.
アプリケーション	－	M2M CEP 機能ライブラリ（libcep.so）を利用して解析処理を実行する，任意のアプリケーションソフトウェア.

4. 機能

4.1. M2M CEP 機能（libcep.so）

M2M CEP 機能は下記ディレクトリ下に設置される共有ライブラリとして実装する.

- /usr/local/lib/libcep.so

M2M CEP 機能が提供する API を利用して任意のプログラムをコンパイルする場合，下記の通り“-lcep”をオプションとして指定する（ここではコンパイラに gcc を利用しているが，clang でも同様である）.

```
$ gcc -lcep (ソースファイルパス).c
```

M2M CEP 機能に含まれる主な機能要素を表 4-1 に示す．また，M2M CEP 機能を任意のアプリケーションに組み込んで利用する際の，M2M CEP の処理シーケンスを図 4-1 に示す. なお，図 4-1 について，M2M CEP 機能は共有ライブラリなので，アプリケーションにおけるシーケンスとは意味合いが異なり，図 4-1 は単なる一例である事に留意する（API の呼び出し方によっては図 4-1 とはシーケンスが異なる）.

表 4-1　M2M CEP 機能の主な機能要素

機能要素	内容
M2M CEP 機能の開始	CEP 構造体オブジェクト（7.1 参照）を作成し，蓄積用 SQLite3 データベースへのレコード挿入準備を行う（ディレクトリの確認，データベースの存在確認等）.
M2M CEP 機能の終了	未だ蓄積用 SQLite3 データベースに挿入されていない CEP 用 SQLite3 データベースのレコード情報を全て蓄積用 SQLite3 データベースに挿入し，CEP 構造体オブジェクトのメモリ領域を解放する.
CEP 用 SQLite3 データベースにデータを挿入	API を通じて指定された CSV（Comma-Separated Values）を解析し，テーブルオブジェクト（7.2 参照）を作成して，CEP 用 SQLite3 データベースに挿入する. また，挿入レコード数が規定値を超過した場合，超過分の過去レコードを蓄積用 SQLite3 データベースに挿入し，CEP 用 SQLite3 データベースから削除する.
CEP 用 SQLite3 データベースに対し検索用 SQL 実行	API を通じて指定された SQL を実行する. 実行結果は CSV 形式の文字列（先頭行にフィールド名のヘッダ，2 行目以降に検索結果レコード）で返す.

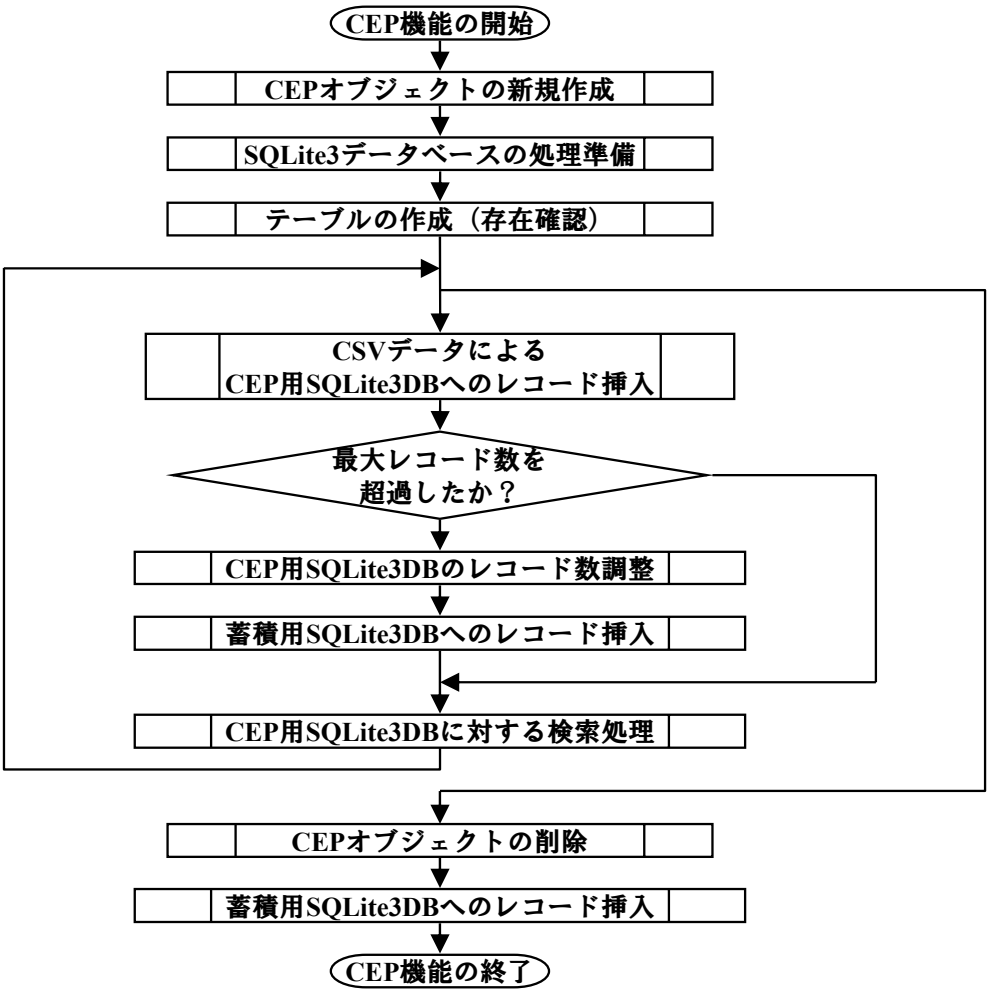

図 4-1　M2M CEP 機能の利用シーケンス例

5. インタフェース

ここでは，CEP に利用する API に絞ってインタフェース仕様を説明する．内部で利用している他の API が必要となる場合，各機能のヘッダファイルを直接参照すること（ヘッダファイルには，ここで説明しているものと同レベルの詳細な説明が記述してある）．

5.1. M2M CEP 機能 API

M2M CEP 機能は C 言語で実装する共有ライブラリ（libcep.so）に内包される機能であり，当該機能が提供するインタフェースは以下に示す C 言語 API（Application Programming Interface）となる（関数自体は M2MCEP.h ヘッダファイルで宣言）．なお，API の各関数で取り扱う文字列は以下を既定とする．

- 形式：unsigned char 型の配列
- 文字コード：UTF-8
- 改行コード：¥r¥n（キャリッジリターン＋ラインフィード）
- 終端：¥0 を付加する（例えば“dog”という ASCII 3 文字の場合，d o g ¥0 の 4 バイトとなり，ヌル文字 1 バイト分だけ文字列のサイズが増える）

5.1.1. M2MCEP_delete（）

項目	内容		
形式	void M2MCEP_delete (M2MCEP **self)		
関数名	CEP_delete		
引数	型	名前	内容
	M2MCEP **	self	M2MCEP_new()関数で作成した M2MCEP 構造体のポインタのポインタ（引数ではポインタのコピーが渡されるため，構造体そのもののメモリ領域を解放するためには，ポインタのポインタを渡す必要がある）．
戻り値	無し．		
処理内容	M2MCEP 構造体に残っている，未だ蓄積用 SQLite3 データベースに未挿入なレコードを一括挿入した後，引数で指定された M2MCEP 構造体の（ヒープ）メモリ領域を解放する． 【注意】 この関数では，M2MTableBuilder_new () で作成したテーブル構築オブジェクトと，M2MColumnList_new () で作成したカラム構造体オブジェクトもまとめて削除するため，これらオブジェクトのヒープメモリ解放用の関数を個別に呼ぶ必要は無く，この関数を呼ぶだけでよい．		

5.1.2. M2MCEP_insertCSV（）

項目	内容

形式	int M2MCEP_insertCSV (M2MCEP *self, const unsigned char *tableName, const unsigned char *csv)		
関数名	M2MCEP_insertCSV		
引数	型	名前	内容
	M2MCEP *	self	CEP 構造体.
	unsigned char *	tableName	CSV データ挿入対象のテーブル名を示す文字列.
	unsigned char *	csv	CSV 形式の文字列. 1 行目にはヘッダとしてテーブルのフィールド名を必ず指定し，2 行目以降に挿入レコードを指定する. レコードには複数行の指定が可能. なお, テーブル名とフィールド名の整合性が取れなかった場合はエラーとなる.
戻り値	成功した場合：挿入に成功したレコード数		
	失敗した場合：-1		
処理内容	CEP を行うためのレコードをメモリ上の SQLite3 データベーステーブルに挿入する．この時，テーブルのレコード数が規定の最大レコード数を超えた場合，過去のレコードをメモリ上の SQLite3 データベースから削除し，尚且つ，当該レコードを蓄積用 SQLite3 データベースに挿入する. なお，引数で指定する CSV の挿入レコード数が最大レコード数を超えると，CEP の対象レコードから除外されてしまうため，最大レコード数を超過しない事に留意する事.		

5.1.3. M2MCEP_new（）

項目	内容		
形式	M2MCEP *M2MCEP_new (const unsigned char *databaseName, const M2MTableBuilder *tableBuilder)		
関数名	M2MCEP_new		
引数	型	名前	内容
	unsigned char *	databaseName	蓄積用 SQLite3 データベース名（ファイル名）を示す文字列. ファイル設置ディレクトリパスは含めない（詳細は 6.2 参照）.
	M2MTableBuilder *	tableBuilder	SQLite3 データベースにテーブルを構築するためのオブジェクト. 蓄積用 SQLite3 データベース（ファイル）については，既にファイルが存在していれば当該オブジェクトは不要であるが，CEP 用 SQLite3 データベースは毎回テーブルを構築する必要があるため，当該コンストラクタにて設定する必要がある.

戻り値	成功した場合：CEP 構造体のポインタ
	失敗した場合：NULL
処理内容	（ヒープ）メモリ領域を新規に獲得し，M2MCEP 構造体を作成する．また，CEP 用 SQLite3 データベースと蓄積用 SQLite3 データベース構築のための準備を行う．

5.1.4. M2MCEP_select（）

項目	内容		
形式	unsigned char *M2MCEP_select (const M2MCEP *self, const unsigned char *sql, unsigned char **result)		
関数名	M2MCEP_select		
引数	型	名前	内容
	M2MCEP *	self	CEP 構造体.
	unsigned char *	sql	CEP 用 SQLite3 データベースのテーブルから CEP に必要なデータを取得するための SELECT 文を示す SQL 文字列.
	unsigned char **	result	SQL 処理結果をコピーするための CSV 形式の文字列ポインタ. バッファリングは関数内部で行うため, 呼び出し側はポインタを指定するだけでよい. 1 行目にフィールド名を示すヘッダ，2 行目以降に結果データを格納する.
戻り値	成功した場合：引数の〝result”変数にコピーした，CSV 形式の文字列ポインタ		
	失敗した場合：NULL		
処理内容	CEP 用 SQLite3 データベースに対し, 引数で指定した SELECT 文の SQL を実行し, その結果として CSV 文字列を引数で指定されたポインタにコピーする. SQL 処理結果として返す CSV について，1 行目にはフィールド名が挿入されるため，処理結果のデータは 2 行目以降に挿入される事となる. 【注意】 “result”変数は関数内部でヒープメモリ領域を獲得しており，メモリリークを防止するため，当該変数の使用後は必ず free()を呼んでメモリ領域を解放する事.		

5.1.5. M2MCEP_setMaxRecord（）

項目	内容		
形式	M2MCEP *M2MCEP_setMaxRecord (M2MCEP *self, const unsigned int maxRecord)		
関数名	M2MCEP_setMaxRecord		
引数	型	名前	内容
	M2MCEP *	self	CEP 構造体.
	unsigned int	maxRecord	CEP 用 SQLite3 データベースのテーブルに関し, 蓄積可能な最大レコード数を設定するための自然数.

戻り値	成功した場合：引数で指定した maxRecord を格納した CEP 構造体
	失敗した場合：NULL
処理内容	CEP 用 SQLite3 データベースに蓄積可能な，各テーブル毎の最大レコード数を更新する（この各テーブル毎の最大レコード数はデータベース単位で一意であり，テーブル毎に指定可能ではない事に留意する事）. また，更新された最大レコード数に合わせ，CEP 用 SQLite3 データベースから蓄積用 SQLite3 データベースへレコードを移動させる必要が生じた場合，レコードの移行処理を実行する.

5.1.6. M2MCEP_setPersistence（）

項目	内容		
形式	M2MCEP *M2MCEP_setPersistence (M2MCEP *self, const bool persistence)		
関数名	M2MCEP_setPersistence		
引数	型	名前	内容
	M2MCEP *	self	CEP 構造体
	bool	persistence	永続性（ファイル上の SQLite3 データベースへの記録）の可否を示すフラグ
戻り値	成功した場合：引数で指定した永続性可否を示すフラグを格納した CEP 構造体		
	失敗した場合：NULL		
処理内容	引数で指定された CEP 構造体オブジェクトに対し，永続性（ファイル上の SQLite3 データベースへの記録）の可否を示すフラグをセットする. true を設定した場合，M2MCEP_insertCSV() で挿入されたレコードは全て，ファイル上の SQLite3 データベースへ記録する. false を設定した場合，ファイル上の SQLite3 データベースへの記録は実行しない. なお，デフォルトの設定値は true である.		

5.1.7. M2MCEP_setPersistence（）

項目	内容		
形式	M2MCEP * M2MCEP_setVacuumRecord (M2MCEP *self, const unsigned int vacuumRecord)		
関数名	M2MCEP_setVacuumRecord		
引数	型	名前	内容
	M2MCEP *	self	CEP 構造体
	unsigned int	vacuumRecord	バキューム処理のレコード数を示す整数値（0 を指定した場合は自動バキューム設定）
戻り値	成功した場合：引数で指定した maxRecord を格納した CEP 構造体		
	失敗した場合：NULL		
処理内容	SQLite3 データベースのバキューム処理を行うレコード数を設定する. このバキューム処理設定については，メモリ上の SQLite3 データベースとファイル上の SQLite3 データベース共用の設定となる. なお，デフォルトの設定値は 0 である（デフォルト設定におけるパフォーマンス		

	は極めて良好であるため，特別な用途でない限り値を指定しない方が良い)．

5.2. テーブル構築機能 API

テーブル構築機能は C 言語で実装する共有ライブラリ（libcep.so）に内包される機能であり，M2M CEP 機能が CEP 用／蓄積用 SQLite3 データベースにテーブルを構築する際に利用する．

5.2.1. M2MTableBuilder_new（）

項目	内容
形式	M2MTableBuilder *M2MTableBuilder_new ()
関数名	M2MTableBuilder_new
引数	無し．
戻り値	成功した場合：テーブル構築オブジェクト
	失敗した場合：NULL
処理内容	ヒープメモリ領域を獲得し，テーブル構築オブジェクトを新規作成する．

5.2.2. M2MTableBuilder_setConfig（）

項目	内容		
形式	M2MTableBuilder *M2MTableBuilder_setConfig (M2MTableBuilder *self, const unsigned char *tableName, M2MColumnList *columnList)		
関数名	M2MTableBuilder_getSQL		
引数	型	名前	内容
	M2MTableBuilder *	self	テーブル構築オブジェクト
	unsigned char *	tableName	テーブル名を示す文字列
	M2MColumnList *	columnList	カラム設定情報を格納したカラム構造体オブジェクト
戻り値	成功した場合：カラム設定情報を含むテーブル構築オブジェクト		
	失敗した場合：NULL		
処理内容	テーブルを構築するために必要なカラム構造体オブジェクトを，テーブル名毎に設定する． 即ち，テーブル名を変更するだけで，1 つのテーブル構築オブジェクトには複数のテーブルを構築するための情報を設定する事が出来る．		

5.3. カラム設定機能 API

カラム設定機能は，M2M CEP で利用する SQLite3 データベースのテーブルを構築するのに必要となる，各カラム情報を設定する際に利用する．M2M CEP ライブラリの内部処理としては，当該 API を利用して構築したカラム構造体オブジェクトから，テーブルを新規に構築するための SQL 文を示す文字列を作成し，当該 SQL を実行する事で，SQLite3 データベースを構築する．

なお，**5.2.2** の API 仕様を見れば分かる通り，複数のテーブルを構築するためには複数のカラム構造体オブジェクトが必要である．

5.3.1. M2MColumnList_add（）

項目	内容		
形式	M2MColumnList *M2MColumnList_add (M2MColumnList *self, const unsigned char *columnName, const M2MDataType dataType, const bool primaryKey, const bool autoIncrement, const bool allowNULL, const bool unique)		
関数名	M2MColumnList_add		
引数	型	名前	内容
	M2MColumnList *	self	カラム構造体オブジェクト
	unsigned char *	columnName	カラム名を示す文字列
	M2MDataType	dataType	カラムのデータ型を示す列挙子（**7.6** 参照）
	bool	primaryKey	主キーか否かを示すフラグ
	bool	autoIncrement	自動インクリメントを行うか否かを示すフラグ
	bool	allowNULL	NULL を許容するか否かを示すフラグ
	bool	unique	ユニーク性を加味するか否かを示すフラグ
戻り値	成功した場合：カラム設定情報を蓄積したカラム構造体オブジェクト		
	失敗した場合：NULL		
処理内容	テーブルを構築するために必要なカラム情報をカラム構造体オブジェクトに追加する.		

5.3.2. M2MColumnList_new（）

項目	内容
形式	M2MColumnList *M2MColumnList_new ()
関数名	M2MColumnList_new
引数	無し
戻り値	成功した場合：カラム構造体オブジェクト
	失敗した場合：NULL
処理内容	（ヒープ）メモリ領域を新規に獲得し，M2MColumnList 構造体を新規作成する.

6. データベース仕様

本ソフトウェアで利用する SQLite3 データベースは下記 2 種類であり，各々の利用目的が異なっている．

- CEP 用：CEP のため，一時的にデータを蓄えるために利用する
- 蓄積用：蓄積用のため，永続的にデータを蓄えるために利用する

なお，各データベースに挿入されるデータが重複する事は無く，CEP 用 SQLite3 データベースに挿入した後，既定の最大レコード数を超過した場合に CEP 用 SQLite3 データベースファイルから当該レコードを除外し，蓄積用 SQLite3 データベースに挿入する．

6.1. CEP 用 SQLite3 データベース（メモリ）

CEPのために構築するSQLite3データベースは処理高速化のため,メモリ上にCEP用 SQLite3 データベースを設置する．SQLite3 は標準機能として，メモリ上にデータベースを構築する事が可能であり，この機能を利用する（下記参照）．

```
struct sqlite3* database;                    // データベースオブジェクトの宣言
sqlite3_open(":memory:", & database);    // メモリ上にデータベース構築
```

即ち，CEP 用 SQLite3 データベースではファイルを利用しないため，高速処理が可能であると同時に，情報が永続的に記録される事は無いため，電源を落とすと当該情報は消失する．

6.2. 蓄積用 SQLite3 データベース（ファイル）

蓄積用に構築する SQLite3 データベース（ファイル）は下記ディレクトリを利用する．ここで,「m2m」ディレクトリはユーザのホームディレクトリ配下に本ソフトウェアが自動的に作成するディレクトリである．

- ファイル作成ディレクトリ：/home/(ユーザ名)/m2m/

作成するファイル名（＝データベース名.sqlite）は，API で指定されたデータベース名に拡張子「.sqlite」を添付したものとなる．

7. データ仕様

7.1. M2MCEP 構造体

ライブラリ内部で利用する M2MCEP 構造体を図 7-1 に示す.

```
typedef struct
    {
    unsigned char *databaseName;        // 蓄積用 SQLite3 データベース名文字列
    sqlite3 *fileDatabase;              // 蓄積用 SQLite3 データベース
    sqlite3 *memoryDatabase;            // CEP 用 SQLite3 データベース
    M2MCEPRecord *record;               // レコード情報
    unsigned int maxRecord;             // テーブルに格納出来る最大レコード数
    } M2MCEP;
```

図 7-1　CEP 構造体

7.2. M2MCEPRecord 構造体

ライブラリ内部で利用する M2MCEP レコード構造体を図 7-2 に示す.

```
typedef struct M2MCEPRecord
    {
    struct M2MCEPRecord *previousRecord;    // 前の M2MCEPRecord 構造体
    struct M2MCEPRecord *nextRecord;        // 後ろの M2MCEPRecord 構造体
    unsigned char *tableName;               // テーブル名文字列
    M2MList *columnNameList;                // カラム名を示すリスト配列
    M2MList *newRecordList;                 // 新規レコードのリスト配列
    M2MList *oldRecordList;                 // 挿入済みレコードのリスト配列
    } M2MCEPRecord;
```

図 7-2　M2MCEP レコード構造体

7.3. M2MList 構造体

レコード情報を蓄える M2MList 構造体を図 7-3 に示す.

```
typedef struct M2MList
    {
    struct M2MList *previous;   // 直前の List 構造体ポインタ（先頭は自分と同じ）
    struct M2MList *next;       // 直後の List 構造体ポインタ（末尾は NULL）
    void *value;                // M2MList 構造体が保有する値
    size_t length;              // 値のサイズを示す整数[Byte]
    } M2MList;
```

図 7-3　M2MList 構造体

M2MList 構造体では, 各 M2MList 構造体がポインタで接続された構造となってお

り, 先頭の M2MList 構造体の previous 変数は自分を指し示す. 即ち, 先頭の M2MList 構造体のインスタンスを node とすると,

```
node->previous==node
```

となる. また, 末尾の M2MList 構造体の next 変数は NULL を指し示す. 即ち, 末尾の M2MList 構造体のインスタンスを node とすると,

```
node->next==NULL
```

となる.

7.4. M2MTableBuilder 構造体

SQLite3 データベースのテーブル構築用 SQL 文を作成するためのカラム設定情報を蓄える M2MTableBuilder 構造体を図 7-4 に示す.

```
typedef struct M2MTableBuilder
    {
    struct M2MTableBuilder previous;    // 直前のテーブル構築オブジェクト
    struct M2MTableBuilder next;        // 直後のテーブル構築オブジェクト
    unsigned char *tableName;           // テーブル名を示す文字列
    M2MColumnList *columnList;          // 各カラムの設定情報を格納した配列
    } M2MTableBuilder;
```

図 7-4 M2MTableBuilder 構造体

7.5. M2MColumnList 構造体

SQLite3 データベースのテーブルを構築する際の, 各カラム設定情報を格納した M2MColumnList 構造体を図 7-5 に示す.

```
typedef struct
    {
    unsigned char *columnName;          // カラム名を示す文字列
    M2MDataType dataType;               // カラムのデータ型
    bool primaryKey;                    // 主キー有効化フラグ
    bool autoIncrement;                 // 自動インクリメント有効化フラグ
    bool allowNULL;                     // NULL 有効化フラグ
    bool unique;                        // ユニーク性有効化フラグ
    } M2MColumnList;
```

図 7-5 M2MColumnList 構造体

7.6. M2MDataType 列挙型（テーブルのデータ型定義）

M2M CEP 機能において利用するデータ型を下記に示す. これらは, M2M CEP 機能の内部処理的に SQLite3 で利用可能な 5 種類のデータ型(NULL, INTEGER, REAL, TEXT, BLOB）に自動的に分類し, テーブルを構築するのに利用する.

```
enum M2MDataType
    {
```

```
    M2M_CEP_BLOB;          // バイナリ型
    M2M_CEP_BOOL;          // 真理値型
    M2M_CEP_CHAR;          // 文字型
    M2M_CEP_DATETIME;      // 時刻型（1970/01/01 00:00:00 からのロング値）
    M2M_CEP_DOUBLE;        // 実数型
    M2M_CEP_FLOAT;         // 実数型
    M2M_CEP_INTEGER;       // 整数型
    M2M_CEP_NUMERIC;       // 整数型
    M2M_CEP_REAL;          // 実数型
    M2M_CEP_TEXT;          // 文字列型
    M2M_CEP_VARCHAR;       // 文字列型
    };
```

図 7-6　テーブルのデータ型

8. 性能諸元

TBD