abs_agent　Advanced User Manual

abs_agent Advanced User Manual Rev A.1.2b
abs_agent version: 2.33
2024/11/8

オールブルーシステム (All Blue System)
ウェブページ: www.allbluesystem.com
コンタクト:contact@allbluesystem.com

1 はじめに

このマニュアルでは abs_agent で使用可能な幾つかのオプション機能と補足情報を説明します。

ここで紹介している機能を使用するためには、OS（Debian GNU/Linux, Raspberry Pi OS等）に幾つかのパッケージのインストールが必要になる場合があります。パッケージをインストールする時に、abs_agent が動作するコンピュータを一時的にインターネットに接続して最新のパッケージデータベースの更新と依存パッケージの導入作業を行います。

詳しいセットアップ方法については 各機能の章を参照してください。また、インストールするパッケージの動作環境やライセンスについては各パッケージのドキュメントを参照してください。

オールブルーシステムは、abs_agent から利用する OS やライブラリで提供される機能のなかで、ユーザーによって別途セットアップされるパッケージのライセンスについては関知しません。インストールする各パッケージのライセンス条項を確認した後、セットアップ作業を行って下さい。

2 表記について

2.1 著作権および登録商標

Linux は、Linus Torvalds 氏の日本およびその他の国における登録商標または商標です。
Raspberry Piは英国Raspberry Pi財団の登録商標です。Raspberry Pi is a trademark of the Raspberry Pi Foundation.
Debian は Software in the Public Interest, Inc. の登録商標です。
WindowsXP, Windows2000, Windows 2003 Server, Windows7 はマイクロソフト社の登録商標です。
XBee, XBee Pro は Digi, Inc. の登録商標です。

2.2 連絡先

オールブルーシステム（All Blue System）
ウェブページ http://www.allbluesystem.com
メール contact@allbluesystem.com

3 パーマネントデータ保存機能 (STORAGE)

3.1 機能概要

ストレージ機能は abs_agent にパーマネントストレージ(電源断でも情報が失われない)と格納データの検索機能を提供します。ストレージ機能を利用するとチャンネルとキー値(UID)を指定して任意の文字列データを保存できます。ストレージに格納するデータ毎に登録・更新時のタイムスタンプが同時に保存され、データ検索時のソートや排他制御に利用できます。

ストレージ機能のチャンネルは、登録するデータのキー値(UID)をグループ分けするために使用できます。
ストレージ機能のライブラリ関数ではチャンネル毎に保存領域が分かれますので、データ登録や検索時にはチャンネル内に限定したデータを扱うことができます。

ストレージに格納する複数のデータが RDBMS のレコードの様に同じデータ構造を持っている場合には、チャンネルは RDBMS のテーブルと同様に扱えます。この時、登録時に指定するキー名(UID)に空文字列を指定すると、重複しないユニークな文字列を自動的に付与させることができます。

ストレージ機能で保存可能なデータ値は文字列のみで、数値やテーブル等を保存したい場合には数値<->文字列変換やJSONシリアライズ処理を併用してください。データ登録時に、後でデータ検索するときに利用可能な検索キーとその検索データ値のペアを任意の個数登録できます。検索データ値には文字列型、整数型、日付型、日付時刻型のデータを使用できます。

ストーレジ機能ではキー文字列(UID)の最大サイズは 64バイトで、キー(UID)に対応するデータの最大文字列サイズは 4096バイトです。

データの取得と更新時に RDBMS の持つプリミティブなロック機構を利用していますので、マルチスレッドで動作するスクリプトでの同時アクセスを安全に行えます。

3.2 データベース構成

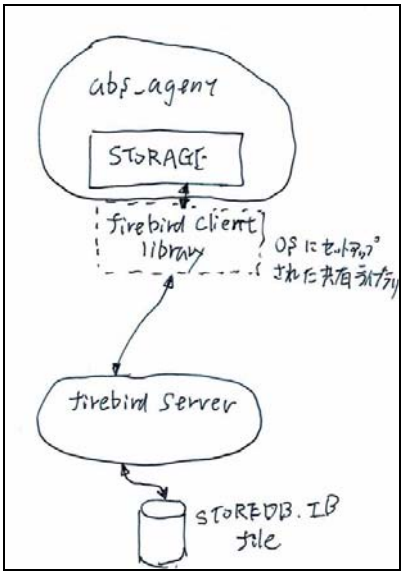
abs_agent のストレージ機能で利用する RDBMS は、オープンソースで開発されている Firebird RDBMS[1](以降 RDBMS)を使用します。RDBMS は abs_agent が動作するコンピュータ上に配置したり、ネットワーク上の外部コンピュータで動作する様に設定できます。

RDBMS を設置したコンピュータを、複数の abs_agent からリモートアクセスして同時に共有できます。この構成で運用すると、複数の abs_agent サーバーのデータ共有と永続データ保存を同時に実現できます。
RDBMS の動作するコンピュータを電源供給の安定した PC に配置することで、リモート側に配置した複数の abs_agentのデータ保全と安定性を向上できます。

[1] https://firebirdsql.org/

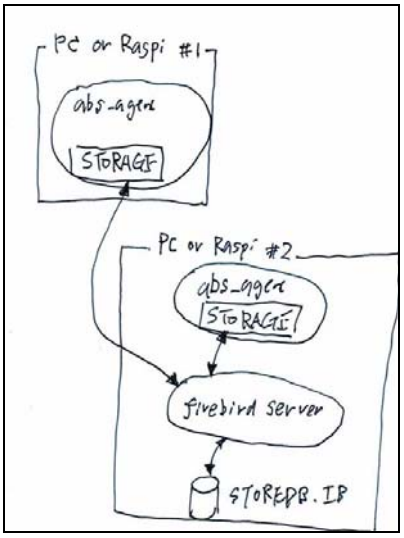
RDBMS を設置するコンピュータではファイルシステムへの頻繁なアクセスを行うため、無停電電源や耐久性のあるデータ記憶媒体(HDD,SSD etc..)、十分なCPU性能と物理メモリが必要です。これらの要求スペックを満たすために、RDMSの運用環境は PC/AT 互換機上で動作する Debian GNU Linux に構築して、abs_agent の動作コンピュータとは分離することをお勧めします。要求スペックを満たすことができれば Raspberry Pi 上にRDMS をインストールして、abs_agent と RDMS を１つのハードウエア上に構築することも可能です。

(1) abs_agent のストレージ機能は、OS にインストールされた Firebird RDBMS クライアントライブラリ(以降 Firebird Client)経由で Firebird RDMS データベースサーバー(以降 Firebird Server)にアクセスします。abs_agent が動作するコンピュータで Firebird Server を起動する場合には下記の図のような構成になります。



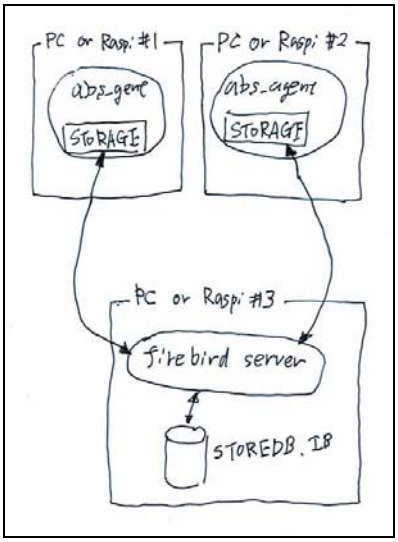
(abs_agent の ストレージ機能は Firebird Clientライブラリ経由で Firebird Server へアクセスする)

(2) 複数のコンピュータで動作している abs_agent から、同一のリモート Firebird Serverを共有することもできます。それぞれの abs_agent からは、共有されたデータの取得、更新、削除、ロック等の操作が可能で、abs_agent 間の共有データ領域として利用できます。リモート側に設置した Firebird Server を使用する場合でも、abs_agent が動作するコンピュータにはFirebird Client機能を利用するため Firebird RDBMS(firebird-server3.0) のインストールが必要です。



(ストレージ機能で参照するデータベースを２つの abs_agent 間で共有する)

(3) abs_agent を動作させるコンピュータとは別に、Firebird RDBMS のみをインストールしたコンピュータを設置して、abs_agentからリモート接続する方法で運用することもできます。リモートで動作させる Firebird のハードウエアタイプ("x86","Raspi"等)は、abs_agent が動作するものと異なっていても構いません。



(abs_agent が動作するコンピュータとは別のハードウエアに専用のデータベースサーバーを設置)

3.3 アプリケーションデータ保護機能

ストレージ機能では RDBMS 自身が持つトランザクション機能を利用した排他・並列アクセス機能を提供する他に、アプリケーションレベルでのデータ保護機能も提供します。

複数のアプリケーションやスクリプトからアプリケーションレベルでのデータの同時更新機能を提供するために、"楽観的ロック"のためのタイムスタンプ(LastUpdate)パラメータと、"悲観的ロック"のためのユーザーロック(LockInfo)パラメータを指定できます。これらの機能は片方のみ使用することや両方同時に使用することもできます。

登録しているデータに対してユーザーが任意のロック状態を設定できます(悲観的ロック)。
ロック状態はアプリケーション側でデータの取得後にそのデータへの更新、削除を行う期間に、他のアプリケーションからの意図しないデータ更新を防ぐために設定します。

ロック状態はユーザーが任意のロック文字列をデータに指定することで設定でき、データの更新操作を行う場合にはロック時に指定したロック文字列を併せて指定します。ロック中のデータに対して正しいロック文字列を指定しないで更新を試みるとエラーになります。

また、ストレージデータ取得時に同時にそのデータの最終更新時刻を示すタイムスタンプ値を取得できます(楽観的ロック)。データ更新時にこのタイムスタンプを指定することで、更新前のタイムスタンプとは異なっている時にエラーを検出します。このようにデータ更新前のタイムスタンプを指定することで、他のクライアントによって更新されたデータに対する意図しない上書きを防止できます。

これらのロック機能を同時に使用することで、"悲観的ロック"に使用するロック文字列を複数のアプリケーションやセッション間で共有してアプリケーションの利便性やメンテナンス性の向上を図り、且つ "楽観的ロック"機能を併用することでアプリケーションで使用するデータの整合性を保つことが可能になります。

複数のアプリケーションやクライアントからの同一データ更新を保護する必要がない場合や、他の方法でアプリケーションのデーター貫性を実現できる場合には、"LockInfo" や "LastUpdate" パラメータを省略してこれらのアプリケーションレベルでのロック機能を使用しないこともできます。

3.4 セットアップ

3.4.1 Firebird RDBMS インストール

ストレージ機能を動作させるためには、abs_agent が動作するコンピュータに "firebird3.0-server" パッケージとそれに付属する firebird-client 共有ライブラリのインストールが必要です。また、単独で Firebird RDBMS を動作させるコンピュータにも同様のインストールしてください。

(1) 最初に、以下のコマンドで現在 OS に "firebird3.0-server" パッケージがインストール済みかどうかを確認します。

```
sudo apt-get update
sudo apt list "*firebird*"
```

下記の様な表示になった場合には既に "firebird3.0-server" パッケージが導入済みなのでインストールは不要です。

```
firebird-dev/oldstable 3.0.1.32609.ds4-14 armhf
firebird3.0-common/oldstable,now 3.0.1.32609.ds4-14 all [インストール済み]
firebird3.0-common-doc/oldstable,now 3.0.1.32609.ds4-14 all [インストール済み、自動]
firebird3.0-doc/oldstable 3.0.1.32609.ds4-14 all
```

```
firebird3.0-examples/oldstable 3.0.1.32609.ds4-14 all
firebird3.0-server/oldstable,now 3.0.1.32609.ds4-14 armhf [インストール済み]
firebird3.0-server-core/oldstable,now 3.0.1.32609.ds4-14 armhf [インストール済み、自動]
firebird3.0-utils/oldstable,now 3.0.1.32609.ds4-14 armhf [インストール済み、自動]
libdbd-firebird-perl/oldstable 1.24-1+deb9u1 armhf
libopendbx1-firebird/oldstable 1.4.6-11 armhf
libsoci-firebird3.2/oldstable 3.2.3-2 armhf
libwtdbofirebird-dev/oldstable 3.3.6+dfsg-1.1 armhf
libwtdbofirebird40/oldstable 3.3.6+dfsg-1.1 armhf
```

("firebird3.0 server" パッケージがインストール済みの表示例)

(2) 未インストールの場合には、下記のコマンドでバイナリパッケージをインストールします。

```
sudo apt-get install firebird3.0-server
```

インストールプロセス内でデータベース管理者ユーザーとパスワードの入力を促された場合には、ユーザー名 "sysdba"、パスワードを "masterkey" に設定します。任意の文字列を設定することもできますが、その場合は以降のセットアップ作業時に SQL によるデータベース操作やデータベースファイル作成等を行う場合に、ここで指定したデータベース管理者ユーザー名とパスワードを指定します。

(3) 次に firebird.conf 設定ファイル中の "RemoteBindAddress = localhost" 指定部分をコメントアウトします。

エディタで設定ファイルを開きます。

Raspberry Pi OS の場合は、

```
sudo vi /usr/lib/arm-linux-gnueabihf/firebird/3.0/firebird.conf
```

x86 Debian の場合は、

```
sudo vi /usr/lib/i386-linux-gnu/firebird/3.0/firebird.conf
```

になります。(バージョンによってファイルパスが異なる場合があります)

エディタで、ファイル内の以下の部分をコメントアウト(先頭に '#' 文字を挿入)してください。

```
#RemoteBindAddress =
#RemoteBindAddress = localhost
```

ファイルの変更が完了したら、firebird3.0-serverサービスを再起動させます。

最初に以下のコマンドでサービスの動作状態を確認します。

```
sudo service --status-all
          ..
          ..
 [ + ]  firebird3.0
          ..
          ..
```

次のコマンドでサービスを再起動させます。

```
sudo service firebird3.0 stop
sudo service firebird3.0 start
```

(4) Firebird RDBMS のクライアントライブラリのインストール状態確認と必要に応じてシンボリックリンクを作成します。ホスト環境と firebird3.0-server インストール環境によってクライアントライブラリのシンボリックリンクが作成されていない場合があります。OS(Debian や Raspi OS) の共有ライブラリが格納されているディレクトリ内に libfbclient.so または libfbclient.so.2.5.1 が存在するかどうかを以下のコマンドで確認します。
以下は Raspberry Pi OS の例です。x86 Debian の場合は "cd /usr/lib/i386-linux-gnu/" の様にします。

```
pi@raspi3:~/fptest/fb_select$ cd /usr/lib/arm-linux-gnueabihf/
pi@raspi3:/usr/lib/arm-linux-gnueabihf$ ls -l libfb*
lrwxrwxrwx 1 root root      16  3月 15  2020 libfbclient.so -> libfbclient.so.2
lrwxrwxrwx 1 root root      20 11月 20  2021 libfbclient.so.2 -> libfbclient.so.3.0.1
-rw-r--r-- 1 root root 1337400 11月 20  2021 libfbclient.so.3.0.1
```

上記の実行例の様に libfbclient.so または libfbclient.so.2.5.1 が存在する場合は以降の作業は必要ありません。存在しない場合は既存の最新版ライブラリファイルへのシンボリックリンクを作成します。以下のRaspberry Pi OS での実行例では libfbclient.so.2.5.1 を新規に作成して、リンクの実体はインストールされている既存ライブラリファイル libfbclient.so.3.0.7 を指定しています。

```
satoshi@uranus:~$ cd /usr/lib/arm-linux-gnueabihf/
satoshi@uranus:/usr/lib/arm-linux-gnueabihf$ ls -l libfb*
lrwxrwxrwx 1 root root      20 12月 26  2020 libfbclient.so.2 -> libfbclient.so.3.0.7
-rw-r--r-- 1 root root 1399888 12月 26  2020 libfbclient.so.3.0.7
satoshi@uranus:/usr/lib/arm-linux-gnueabihf$ sudo ln -s libfbclient.so.3.0.7 libfbclient.so.2.5.1
satoshi@uranus:/usr/lib/arm-linux-gnueabihf$ ls -l libfb*
lrwxrwxrwx 1 root root      20 12月 26  2020 libfbclient.so.2 -> libfbclient.so.3.0.7
lrwxrwxrwx 1 root root      20 12月 11 22:20 libfbclient.so.2.5.1 -> libfbclient.so.3.0.7
-rw-r--r-- 1 root root 1399888 12月 26  2020 libfbclient.so.3.0.7
```

x86 Debian でシンボリックリンクを作成する例は下記になります。この例では既存ライブラリファイル libfbclient.so.3.0.11 からシンボリックリンク(libfbclient.so.2.5.1)を作成しています。

```
satoshi@jupiter:~$ cd /usr/lib/i386-linux-gnu
satoshi@jupiter:/usr/lib/i386-linux-gnu$ ls -l libfb*
lrwxrwxrwx 1 root root      21 11月  9  2022 libfbclient.so.2 -> libfbclient.so.3.0.11
-rw-r--r-- 1 root root 1977080 11月  9  2022 libfbclient.so.3.0.11
```

```
satoshi@jupiter:/usr/lib/i386-linux-gnu$ sudo ln -s libfbclient.so.3.0.11 libfbclient.so.2.5.1
satoshi@jupiter:/usr/lib/i386-linux-gnu$ ls -l libfb*
lrwxrwxrwx 1 root root      21 11月  9  2022 libfbclient.so.2 -> libfbclient.so.3.0.11
lrwxrwxrwx 1 root root      21  3月 29 17:19 libfbclient.so.2.5.1 -> libfbclient.so.3.0.11
-rw-r--r-- 1 root root 1977080 11月  9  2022 libfbclient.so.3.0.11
```

3.4.2 データベースファイル設置

abs_agent のストレージ機能からアクセスする Firebird RDBMS のデータベースファイルを設置します。データベースファイルは abs_agent のインストールキットに格納されていますので、abs_agent のコンフィギュレーションファイルを設置したディレクトリに STOREDB.IBファイルをコピーしてください。abs_agent インストールディレクトリでそのまま運用している場合はコピーは不要です。

```
cd ~/my_config_raspi3
cp ~/abs_agent/STOREDB.IB .
```

データベースファイル(STOREDB.IB) のパーミッションを変更して一般ユーザー(firebirdプロセス)で読み書き可能にします。

```
sudo chmod 666 STOREDB.IB
```

また、データベースファイルを格納しているディレクトリが一般ユーザーからアクセス可能になっていることも確認してください。これは STOREDB.IB を格納しているディレクトリのパーミッションに実行権(ディレクトリ内アクセス権)が付与されているかを確認して、もし付与されていない場合には "sudo chmod +x <デイレクトリパス名>" または "sudo chmod 777 <デイレクトリパス名>"コマンドで設定します。

```
cd ..
sudo chmod 777 my_config_raspi3   (またはディレクトリ名 "abs_agent" を指定する)
```

動作させる Firebird RDBMS のバージョンによっては、インストールキット付属の STOREDB.IB ファイルが使用できない場合があります。このときは、Firebird RDBMS をインストールしたコンピュータ上でデータベースファイルを新規に作成します。データベースファイルを作成するための SQL スクリプトは abs_agent のインストールキット内の contrib/DB/db_def.sql ファイルです。このファイルをデータベースファイルを配置するディレクトリにコピーして使用します。データベースファイル作成 SQLスクリプト内では、データベースファイルに指定するユーザー名とパスワードはデータベース管理者と同じ "sysdba", "masterkey" になっています。これらを変更したい場合は SQL スクリプトファイルを修正してから実行します。

```
pi@raspi3:~/my_config_raspi3$ cp ~/abs_agent/contrib/DB/db_def.sql .
pi@raspi3:~/my_config_raspi3$ sudo isql-fb
Use CONNECT or CREATE DATABASE to specify a database
```

```
SQL> input db_def.sql;
SQL> commit;
SQL> quit;


pi@raspi3:~/my_config_raspi3$ sudo chmod 666 STOREDB.IB
```

（データベース作成のコマンド実行例）

3.4.3 Firebird RDBMS 動作確認

前項でデータベースファイル(STOREDB.IB) を設置したディレクトリに移動した後、SQL を使用してデータベース動作を確認します。ここではテーブル一覧表示表示コマンド "show tables;" と、テーブル定義表示コマンド "show table storage_data;" が実行できるかを確認します。

```
pi@raspi3:~$ cd my_config_raspi3              (STOREDB.IB ファイルを配置したディレクトリに移動)
pi@raspi3:~$ sudo isql-fb
Use CONNECT or CREATE DATABASE to specify a database
SQL> connect 'STOREDB.IB' user 'sysdba' password 'masterkey';
Database: 'STOREDB.IB', User: SYSDBA
SQL> show database;
Database: STOREDB.IB
        Owner: SYSDBA
PAGE_SIZE 8192
Number of DB pages allocated = 224
Number of DB pages used = 216
Number of DB pages free = 8
Sweep interval = 20000
Forced Writes are ON
Transaction - oldest = 19
Transaction - oldest active = 20
Transaction - oldest snapshot = 20
Transaction - Next = 24
ODS = 12.0
Database not encrypted
Default Character set: UTF8
SQL>
SQL> show tables;
       DATETIME_INDEX                  INT_INDEX
       STAT_DATA                       STORAGE_DATA
       STR_INDEX                       VER
```

```
SQL> show table storage_data;
CHANNEL                         VARCHAR(64) Not Null
STORAGE_UID                     VARCHAR(64) Not Null
STORAGE_STR                     VARCHAR(4096) Nullable
LOCKINFO                        VARCHAR(128) Nullable
LASTUPDATE                      TIMESTAMP Not Null
CONSTRAINT INTEG_4:
  Primary key (CHANNEL, STORAGE_UID)
SQL>
SQL> quit;
pi@raspi3:~$
```

上記の様な表示になれば、"firebird3.0-server" の設置とデータベースファイルの配置は正常に完了しています。Firebird RDBMS が動作するコンピュータで abs_agent を動作させずに運用する場合は設置完了です。これ以降のセットアップ作業は abs_agent を動作させるコンピュータで必要となります。

3.4.4 abs_agent 設定と動作確認

abs_agent のコンフィギューレションファイルを編集して ストレージ機能を有効にします。STOREGE 機能で使用する設定項目は以下の部分になります。abs_agent インストール直後は全て空白ですが、最初の起動時に下記のデフォルト値が設定されます。

```
<STORAGE>
  <AutoOnline type="boolean">False</AutoOnline>
  <DBSessionPool type="integer">3</DBSessionPool>
  <DBHostName type="string">localhost</DBHostName>
  <DBPath type="string">$CONFIGDIR$/STOREDB.IB</DBPath>
  <DBUser type="string">sysdba</DBUser>
  <DBPassword type="string">masterkey</DBPassword>
</STORAGE>
```

(abs_agent コンフィギュレーションファイル abs_agent.xml ファイル中の ストレージ機能設定部分)

エディタ等でコンフィギューレションファイルを編集して、<STORAGE> タグ内にある各チャイルドタグの設定を行います。

<AutoOnline> を True に設定して abs_agent で ストレージサービスモジュールを使用可能にします。

<DBHostName> を "localhost" に指定するとローカルコンピュータで動作中の Firebird RDBMS に接続します。リモートコンピュータで動作する Firebird RDBMSに接続する場合にはリモートコンピュータのホスト名または IPアドレスを指定します。デフォルト値は "localhost" です。

<DBUser> と <DBPassword> は Firebird RDBMS サーバー側に設置したデータベース(STOREDB.IB) に設定したユーザー名とパスワードを指定します。デフォルト値はデータベース管理ユーザーと同じ "sysdba" と "masterkey" です。

<DBPath> には接続先データベースファイル名を指定します。abs_agent が動作するコンピュータ上で Firebird RDBMS を動作させる場合には、abs_agent をインストールしたコンピュータ内のデータベースファイルへのパス名を指定します。パス名の文字列中に "$CONFIGDIR$" を記述すると、その部分が abs_agent 起動時に参照されるコンフィギュレーションファイルを設置したディレクトリパス名に置き換えられて解釈されます。デフォルト設定は "$CONFIGDIR$/STOREDB.IB" です。リモートコンピュータに設置した Firebird RDBMS に接続する場合は、リモートコンピュータ上にに存在するデータベースファイルのフルパス名を指定します。このとき "$CONFIGDIR$" の記述は無効になりますので、必ずリモートコンピュータのファイルシステム上でのデータベースファイル(絶対パス)名を指定します。

<DBSessionPool> はデータベース接続を行う同時セッション数を指定します。通常はデフォルト値 3 から変更する必要はありません。

● 設定例１: abs_agent が動作しているコンピュータの Firebird RDBMS に接続する場合

```
<STORAGE>
  <AutoOnline type="boolean">True</AutoOnline>
  <DBSessionPool type="integer">3</DBSessionPool>
  <DBHostName type="string">localhost</DBHostName>
  <DBPath type="string">$CONFIGDIR$/STOREDB.IB</DBPath>
  <DBUser type="string">sysdba</DBUser>
  <DBPassword type="string">masterkey</DBPassword>
</STORAGE>
```

● 設定例２ : リモートに設置したコンピュータ(192.168.100.12) の Firebird RDBMS に接続する場合

```
<STORAGE>
  <AutoOnline type="boolean">True</AutoOnline>
  <DBSessionPool type="integer">3</DBSessionPool>
  <DBHostName type="string">192.168.100.12</DBHostName>
  <DBPath type="string">/home/user1/STOREDB.IB</DBPath>
  <DBUser type="string">sysdba</DBUser>
  <DBPassword type="string">masterkey</DBPassword>
</STORAGE>
```

設定が終了したら、abs_agent を再起動させることで ストレージ機能が使用可能になります。

```
cd ~/abs_agent
./agent_task -K
```

```
./agent_shutdown
(abs_agent がシャットダウンするまで 10秒程度待つかログを確認する)
sudo /home/pi/abs_agent/abs_agent -l 192.168.100.45 -c /home/pi/my_config_raspi3/abs_agent.xml
```

3.4.5 ストレージ機能動作確認 (ログ確認とエラー対応)

abs_agent で ストレージ機能を有効にしたときは、ログサーバーでログメッセージを確認してください。このとき、正常に起動した場合には下記の様なメッセージが表示されます。

```
2024/07/10 10:47:54 raspi3      FBSessionCtrl         0 InitPool success, entries = 3
2024/07/10 10:47:54 raspi3      FBSessionCtrl         0 FTimeOutTimer start..
2024/07/10 10:47:54 raspi3      STORAGE               0 STORAGE startup...
2024/07/10 10:47:54 raspi3      SERVER_START          0 start..
```

何らかの原因でエラーが発生した場合はログサーバーにエラーメッセージが記録されます。対応方法については、下記のエラーメッセージ出力例を参考に、前項までのセットアップ手順を確認してください。

(1) Firebird3.0 RDBMS 未インストールまたはクライアントライブラリが見つからない

```
2024/03/27 09:17:45 mercury     STORAGE               0 Startup:*ERROR* initialization failed, Can not load default Firebird clients ("libfbclient.so.2.5.1" or "libgds.so" or "libfbembed.so.2.5"). Check your installation.
2024/03/27 09:17:45 mercury     FBSessionCtrl         0 TFBSessionPool destroy, entries = 0
2024/03/27 09:17:45 mercury     STORAGE               0 Startup: cleanup FDBSessions.
2024/03/27 09:17:45 mercury     ServiceMain           0 SvcMonitor:*ERROR* service class startup failed. msg = InitPool() failed.
```

```
2024/03/29 12:39:04 jupiter     STORAGE               0 Startup:*ERROR* initialization failed,  : DoInternalConnect :
 -Unable to complete network request to host "localhost".
 -Failed to establish a connection.
2024/03/29 12:39:04 jupiter     FBSessionCtrl         0 TFBSessionPool destroy, entries = 0
2024/03/29 12:39:04 jupiter     STORAGE               0 Startup: cleanup FDBSessions.
2024/03/29 12:39:04 jupiter     ServiceMain           0 SvcMonitor:*ERROR* service class startup failed. msg = InitPool() failed.
```

(2) データベースファイルのパス名が間違っている

```
2024/01/16 09:15:13 Startup:*ERROR* initialization failed, TIBConnection : DoInternalConnect :
 -I/O error during "open" operation for file "/home/aaa.db"
 -Error while trying to open file
 -No such file or directory
2024/01/16 09:15:13 TFBSessionPool destroy, entries = 0
2024/01/16 09:15:13 Startup: cleanup FDBSessions.
Main:*ERROR* service class startup failed. msg = InitPool() failed.
```

(3) データベースファイルへのアクセス失敗（データベースファイル自身のパーミッション未設定や、データベース

ファイルを格納したディレクトリアクセスが未許可の場合など）

```
2024/03/27 09:35:38 uranus      STORAGE             0 Startup:*ERROR* initialization failed, TIBConnection :
DoInternalConnect :
 -no permission for read-write access to database /home/satoshi/my_config_uranus/STOREDB.IB
 -IProvider::attachDatabase f...
2024/03/27 09:35:38 uranus      FBSessionCtrl       0 TFBSessionPool destroy, entries = 0
2024/03/27 09:35:38 uranus      STORAGE             0 Startup: cleanup FDBSessions.
2024/03/27 09:35:38 uranus      ServiceMain         0 SvcMonitor:*ERROR* service class startup failed. msg = InitPool() failed.
```

(4) データベースバージョンの不一致

```
  2024/07/01 04:24:41 raspi3      STORAGE             0 Startup:*ERROR* initialization failed, TIBConnection :
  DoInternalConnect :
   -unsupported on-disk structure for file /home/pi/my_config_raspi3/STOREDB.IB; found 12.2, support 12.0
   -IProvider::atta...
  2024/07/01 04:24:41 raspi3      FBSessionCtrl       0 TFBSessionPool destroy, entries = 0
  2024/07/01 04:24:41 raspi3      STORAGE             0 Startup: cleanup FDBSessions.
2024/07/01 04:24:41 raspi3      ServiceMain         0 SvcMonitor:*ERROR* service class startup failed. msg = InitPool() failed.
```

この場合はインストールキット同梱の STOREDB.IB を使用しないで、データベースサーバー上で新たにデータベースファイルを作成します。データベースファイル作成方法は "データベースファイル設置"の項を参照してください。

3.4.6 運用時の注意事項

ストレージサービスはデータベースファイルへの読み書きが発生するので、コンピュータの電源を不用意に切断するとデータベースファイルに不具合が発生する可能性があります。

コンピュータの電源を切断するときは、事前に abs_agent が動作しているコンピュータで agent_shutdown コマンドを使用してプログラムを正常終了させてください。その後データベースサーバーを含む OS のシャットダウンを行います。

データベースサーバーを複数の abs_agent から共有アクセスしている場合は、全てのコンピュータ上で同様に agent_shutdown コマンドを使用して abs_agent を正常終了させてから OS とデータベースサーバーのシャットダウンを行って下さい。

データベースサーバーを別コンピュータで動作させている場合は、そのコンピュータは最後にシャットダウンしてください。起動時はデータベースサーバーが動作するコンピュータを最初に起動させてから、abs_agent が動作するコンピュータを起動させます。

abs_agent のストレージ機能使用中に、データベースサーバーとして動作しているコンピュータが何らかの原因で再起動された場合には、データベースサーバーに接続している abs_agent のストレージ機能でエラーが発生します。

この場合にはリモートアクセスしている全ての abs_agent を再起動させてください。

また、abs_agent を自動起動させるときにウェイトを入れることで、ローカルもしくはリモートに設置したデータベースサーバーが起動するまでの時間を確保してから ストレージサービスを起動させることができます。

abs_agent 起動時に 30秒ウェイトを /etc/rc.local スクリプトに設定する例は以下になります。

```
#!/bin/sh -e


(sleep 30
LANG=ja_JP.UTF-8
export LANG
/home/user/abs_agent/abs_agent -c /home/user/my_config_raspi3/abs_agent.xml -l 192.168.100.45 )&


exit 0
```

3.5 storage_stat()

- **機能概要**

  ストレージ機能が有効になっているかを調べる

- **関数定義**

  stat, dbhost, dbpath = storage_stat()

- **パラメータとリターン値**

stat:Boolean	コンフィギュレーションで有効になっていて、且つストレージモジュールインスタンスが存在してかつステータスが ONLINE になっているときのみ true を返す。それ以外は falseが返る
dbhost:String	ストレージがアクセスしているデータベースサーバーのホスト名
dbdpath:String	データベースサーバー内に配置されたデータベースファイルのパス名

- **備考**

  abs_agent で現在 ストレージ機能が動作している場合は true が返り、それ以外は false が返る。

  dbhost, dbpath にはデータベースサーバーのホスト名、データベースファイルのパス名が返る。

- **使用例**

```
local dbhost, dbpath = stat_check(storage_stat())
log_msg('dbhost=' .. dbhost)
log_msg('dbpath=' .. dbpath)
```

上記スクリプトの実行結果ログ

```
dbhost=192.168.100.12
dbpath=/home/satoshi/STOREDB.IB
```

3.6 storage_ver()

- **機能概要**

  ストレージ機能でアクセスしているデータベースのバージョンレコードを返す

- **関数定義**

  stat, major, minor = storage_ver()

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
major:Number	VERテーブル中の MAJORフィールドに格納されている数値
minor:Number	VERテーブル中の MINORフィールドに格納されている数値

- **備考**

  VER テーブルに格納されているレコード値をそのまま返す。実行結果をチェックすることで、Firebird RDBMS サーバーへの接続やデータベースファイルのアクセス時に問題が発生していないかどうかを判断できる。

- **使用例**

```
local major,minor = stat_check(storage_ver())
log_msg('DB ver: ' .. tostring(major) .. "." .. tostring(minor))
```

  上記スクリプトの実行結果ログ

```
DB ver: 1.0
```

3.7 storage_get()

- **機能概要**

  ストレージに保存されているデータを取得する

- **関数定義**

  stat, data, lockinfo, lastupdate = storage_get(channel, uid)

  stat, data_arr, lockinfo_arr, lastupdate_arr = storage_get(channel, uid_arr) -- 複数データ取得

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
data:String	uid で指定した UID(キー)に対応するデータ
lockinfo:String	データがロック中の場合はそのロック情報が入る。未ロック時は "" 空文字列
lastupdate:String	データを最後に更新した日時(YYYY/MM/DD HH:MM:SS.mmm)
channel:String	チャンネル文字列。"" 空文字列を指定すると "_<g_hostname>" を使用する

uid:String　　データ保存時に指定したキー文字列

uid_arr:Table [1..#n] of String　データ保存時に指定した UID(キー)文字列の配列

data_arr:Table [1..#n] of String

　　uid_arr で指定した複数の UID(キー)に対応するデータ配列

lockinfo_arr:Table [1..#n] of String

　　uid_arr で指定した複数の UID(キー)に対応するロック情報の配列

lastupdate_arr:Table [1..#n] of String

　　uid_arr で指定した複数の UID(キー)に対応する最終更新日時の配列

- **備考**

ストレージに保存されているデータを取得する。キー(UID)に指定したデータが存在しない場合は stat に false を返す。対象データがロック中の場合でもデータ取得は常に可能で、この場合にはリターン値 lockinfo にロック文字列が返る。ロック中ではない場合にはリターン値 lockinfo には "" 空文字列が返る。
リターン値 lastupdate にはデータの最終更新時のタイプスタンプ値が "YYYY/MM/DD HH:MM:SS.mmm" の日付時刻フォーマットで返る。

データ登録時に指定したインデックスを取得したい場合は storage_get_index() を使用する。

uid_arr には複数のキー(UID)文字列を配列に格納したものを指定できる。この場合には対応するデータ値が配列に格納されて data_arr に返る。複数のデータ取得中に１つでもエラーが発生した場合には stat に false が返り、data_arr は nil になる。

channel に "" 空文字列を指定すると "_<g_hostname>" をチャンネル名に使用する。(g_hostname はライブラリ関数を実行したコンピュータのホスト名)

- **使用例(1)**

```
local channel = "試験チャンネル"
local uid = "テストキー"
stat_check(storage_set(channel,uid,"<<試験データ>>"))
local data,lockinfo,lastupdate = stat_check(storage_get(channel,uid))
log_msg(channel .. ","..uid .. "," .. data .. "," .. lockinfo .. "," .. lastupdate)
```

上記スクリプトの実行結果ログ

```
試験チャンネル,テストキー,<<試験データ>>,,2024/09/14 15:12:35.447
```

- **使用例(2)**

```
local channel = "試験チャンネル"
stat_check(storage_purge(channel))
for i = 1,10 do
  stat_check(storage_set(channel,'',"<<試験データ#" .. tostring(i) .. ">>"))
```

```
end
local uid_arr = stat_check(storage_select(channel))
local data_arr = stat_check(storage_get(channel,uid_arr))
for i,v in ipairs(data_arr) do
  log_msg(v)
  wait_time(1)
end
```

上記スクリプトの実行結果ログ

```
<<試験データ#10>>
<<試験データ#9>>
<<試験データ#8>>
<<試験データ#7>>
<<試験データ#6>>
<<試験データ#5>>
<<試験データ#4>>
<<試験データ#3>>
<<試験データ#2>>
<<試験データ#1>>
```

3.8 storage_set(),storage_set_json(),storage_set2(),storage_set_json2()

- **機能概要**

  ストレージにデータを保存する。キー(UID)に指定したデータが存在する場合は上書きする。

- **関数定義**

  ■関数コール後自動アンロック

  stat, r_uid, r_lastupdate = storage_set(channel, uid, data *[,*index_tbl *[,*lockinfo *[,*lastupdate*]]])*

  stat, r_uid, r_lastupdate = storage_set_json(channel, uid, data *[,*lockinfo *[,*lastupdate*])*

  ■関数コール前のロックまたはアンロック状態を保つ

  stat, r_uid, r_lastupdate = storage_set2(channel, uid, data *[,*index_tbl *[,*lockinfo *[,*lastupdate*]]])*

  stat, r_uid, r_lastupdate = storage_set_json2(channel, uid, data *[,*lockinfo *[,*lastupdate*])*

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
r_uid:String	データ保存時に使用したキー文字列が返る。
r_lastupdate:String	データを最後に更新した日時(YYYY/MM/DD HH:MM:SS.mmm)が返る
channel:String	チャンネル文字列。"" 空文字列を指定すると "_<g_hostname>" を使用する
uid:String	データ保存時のキー文字列を指定する。""空文字列をリクエストパラメータに指定するとランダムな文字列をキーに使用する。

data:String　　UID(キー)に対応する保存データ

index_tbl:Table of (String or Number)

　　検索時に利用可能な任意のインデックス(キーと値のペア)を格納したテーブル

lockinfo:String　　データがロック中の場合にロック情報を指定する

lastupdate:String　　データを最後に更新した日時(YYYY/MM/DD HH:MM:SS.mmm)を指定する

- **備考**

ストレージにデータを保存する。キー(UID)に指定したデータが存在する場合は上書きする。

データ保存と同時にデータ値に付随するタイムスタンプ値は更新され、リターン値 r_lastupdate に更新後のタイプスタンプ値が返る。

uid に "" 空文字列を指定すると自動生成したユニークな UID 文字列を保存時のキー値に使用する。データ更新時に使用したキー値(UID)はリターン値 r_uid に返る。

lastupdate を指定した場合は、更新前のデータ値に付随するタイムスタンプ値と一致しない場合はデータ更新は行われないで stat に false を返す。lastupdate を指定しない場合はパラメータを省略するか "" 空文字列を指定する。

対象データがロック中の場合は、lockinfo パラメータにロック時に指定したロック文字列を指定する。
lockinfo を指定しない場合はパラメータを省略するか "" 空文字列を指定する。

対象データがロック中の場合に、ロック操作時に指定していたロック文字列が lockinfo パラメータと一致しない場合や lockinfo パラメータを省略した場合にはデータ更新は行われないで stat に false を返す。

対象データがロック中の場合に、ロック操作時に指定していたロック文字列が lockinfo パラメータと一致した場合は、データ更新が行われてその後 storage_set(), storage_set_json() 関数の場合にはレコードはアンロック状態になる。storage_set2(), storage_set_json2() 関数の場合には、データ更新後のロック状態は、コール前の状態を維持したままになる。

対象データがロックされていない時のデータ更新時には lockinfo パラメータに渡された内容は無視され、storage_set()とstorage_set2()、storage_set_json() と storage_set_json2() は全く同じ動作になる。

index_tbl パラメータには検索時に利用可能な任意のインデックスを指定できる。index_tbl を指定しない場合はパラメータを省略するか {} 空テーブルを指定する。

storage_set() コール時には uid で指定したデータに関連付けられた既存のインデックスは全て削除されて、その後 index パラメータで指定したインデックスが登録される。登録時に作成する検索キー(i_key#)と検索値(i_val#)を指定して、index パラメータ(Lua table)を作成する方法は以下の様なスクリプトになる。

```
index = {}
index[i_key#1] = i_val#1
index[i_key#2] = i_val#2
```

index[i_key#n] テーブルに格納するインデックス・キー値 i_key|#n は常に文字列です。インデックス値は文字列、整数、日付時刻の何れかを指定する。

文字列のインデックス値の場合には i_val#n に Lua 文字列をそのまま格納する。
整数のインデックス値には Lua 数値を格納するが、数値が浮動少数値の場合には少数部を切り捨て整数部のみにしてからインデックスに登録される。

数値のインデックス値は整数のみ
ストレージ機能では数値型のインデックス値に浮動少数値を使用しないで整数値のみを使用します。これは RDBMS 上で一致検索を行う時の、誤差発生に起因する検索漏れを防ぐためです

日付時刻の場合には i_val#n に Lua 文字列として "YYYY/MM/DD HH:MM:SS" または "YYYY/MM/DD HH:MM:SS.mmm" または "YYYY/MM/DD" の日付時刻フォーマットで格納することで、ライブラリ関数内部で日付時刻型に変換されてからインデックスに登録される。時刻のミリ秒部分を省略した場合は .000 秒として登録される。時刻部分を省略して "YYYY/MM/DD" のフォーマットで格納すると、日付時刻型の時刻部分は 0:0:0.000 の値としてインデックスに登録される。

channel に "" 空文字列を指定すると "_<g_hostname>" をチャンネル名に使用する。(g_hostname はライブラリ関数を実行したコンピュータのホスト名)

storage_set_json() 関数は data パラメータに JSON 文字列が指定されることを前提に、index データを自動設定する。この関数はプリロードライブラリ(scripts/preload/018_STORAGE/STORAGE_LIB.lua) によって下記の様に定義される。

```
function storage_set_json(channel,uid,data,...)
  return storage_set(channel,uid,data,g_json.decode(data),...)
end
```

storage_set_json() 関数では data パラメータに渡される JSON 表現されたフィールド値全てをインデックスに登録する。この時インデックスレコード数は(データ数 * 各データ内のフィールド数)になる。必要最低限のインデックスレコード数に抑えるよりも、簡単にデータ登録時に自動でインデックス検索を利用したい場合に適しています。

- **使用例(このスクリプトは scripts/TEST/STORAGE_LOAD_SCOTT.lua に格納されています)**

```
function register(channel,records)
  for i,v in ipairs(records) do
    local new_uid,lastupdate = stat_check(storage_set_json(channel,'',v))
    log_msg('channel,uid,data,lastupdate=' .. channel .. "," .. new_uid .. "," .. lastupdate)
  end
end
local channel = "EMP"
local records = {
'{"EMPNO":7369,"ENAME":"SMITH","JOB":"CLERK"    ,"MGR":7902,"HIREDATE":"1980/12/17","SAL":800 ,"COMM":null,"DEPTNO":20}',
'{"EMPNO":7499,"ENAME":"ALLEN","JOB":"SALESMAN","MGR":7698,"HIREDATE":"1981/2/20" ,"SAL":1600,"COMM":300 ,"DEPTNO":30}',
'{"EMPNO":7521,"ENAME":"WARD", "JOB":"SALESMAN","MGR":7698,"HIREDATE":"1981/2/22" ,"SAL":1250,"COMM":500 ,"DEPTNO":30}',
'{"EMPNO":7566,"ENAME":"JONES","JOB":"MANAGER" ,"MGR":7839,"HIREDATE":"1981/4/2"  ,"SAL":2975,"COMM":null,"DEPTNO":20}',
'{"EMPNO":7654,"ENAME":"MARTIN","JOB":"SALESMAN" ,"MGR":7698,"HIREDATE":"1981/9/28"  ,"SAL":1250,"COMM":1400,"DEPTNO":30}',
'{"EMPNO":7698,"ENAME":"BLAKE","JOB":"MANAGER" ,"MGR":7839,"HIREDATE":"1981/5/1"  ,"SAL":2850,"COMM":null,"DEPTNO":30}',
'{"EMPNO":7782,"ENAME":"CLARK","JOB":"MANAGER" ,"MGR":7839,"HIREDATE":"1981/6/9"  ,"SAL":2450,"COMM":null,"DEPTNO":10}',
'{"EMPNO":7788,"ENAME":"SCOTT","JOB":"ANALYST" ,"MGR":7566,"HIREDATE":"1987/7/13"  ,"SAL":3000,"COMM":null,"DEPTNO":20}',
'{"EMPNO":7839,"ENAME":"KING","JOB":"PRESIDENT" ,"MGR":null,"HIREDATE":"1981/11/17"  ,"SAL":5000,"COMM":null,"DEPTNO":10}',
'{"EMPNO":7844,"ENAME":"TURNER","JOB":"SALESMAN" ,"MGR":7698,"HIREDATE":"1981/9/8"  ,"SAL":1500,"COMM":null,"DEPTNO":30}',
'{"EMPNO":7876,"ENAME":"ADAMS","JOB":"CLERK" ,"MGR":7788,"HIREDATE":"1987/7/13"  ,"SAL":1100,"COMM":null,"DEPTNO":20}',
'{"EMPNO":7900,"ENAME":"JAMES","JOB":"CLERK" ,"MGR":7698,"HIREDATE":"1981/12/3"  ,"SAL":950,"COMM":null,"DEPTNO":30}',
'{"EMPNO":7902,"ENAME":"FORD","JOB":"ANALYST" ,"MGR":7566,"HIREDATE":"1981/12/3"  ,"SAL":3000,"COMM":null,"DEPTNO":20}',
'{"EMPNO":7934,"ENAME":"MILLER","JOB":"CLERK" ,"MGR":7782,"HIREDATE":"1982/1/23"  ,"SAL":1300,"COMM":null,"DEPTNO":10}',
}
stat_check(storage_purge(channel))
register(channel,records)
channel = "DEPT"
records = {
'{"DEPTNO":10,"DNAME":"ACCOUNTING","LOC":"NEW YORK"}',
'{"DEPTNO":20,"DNAME":"RESEARCH"  ,"LOC":"DALLAS"  }',
'{"DEPTNO":30,"DNAME":"SALES"     ,"LOC":"CHICAGO" }',
'{"DEPTNO":40,"DNAME":"OPERATIONS","LOC":"BOSTON"  }',
}
stat_check(storage_purge(channel))
register(channel,records)
channel = "SALGRADE"
records = {
'{"GRADE":1,"LOSAL":700 ,"HISAL":1200}',
'{"GRADE":2,"LOSAL":1201,"HISAL":1400}',
'{"GRADE":3,"LOSAL":1401,"HISAL":2000}',
```

```
'{"GRADE":4,"LOSAL":2001,"HISAL":3000}',
'{"GRADE":5,"LOSAL":3001,"HISAL":9999}',
}
stat_check(storage_purge(channel))
register(channel,records)
```

上記スクリプトの実行結果ログ

```
channel,uid,data,lastupdate=EMP,STO20240914151751754974989O9,2024/09/14 15:17:55.009
channel,uid,data,lastupdate=EMP,STO20240914151751755220060O25,2024/09/14 15:17:55.263
channel,uid,data,lastupdate=EMP,STO20240914151751755477508O76,2024/09/14 15:17:55.521
channel,uid,data,lastupdate=EMP,STO20240914151751755733866O85,2024/09/14 15:17:55.777
channel,uid,data,lastupdate=EMP,STO20240914151751755976168l2,2024/09/14 15:17:56.005
channel,uid,data,lastupdate=EMP,STO20240914151751756204768l9,2024/09/14 15:17:56.249
channel,uid,data,lastupdate=EMP,STO20240914151751756447960l2,2024/09/14 15:17:56.491
channel,uid,data,lastupdate=EMP,STO20240914151751756689761l5,2024/09/14 15:17:56.733
channel,uid,data,lastupdate=EMP,STO20240914151751756946029l2,2024/09/14 15:17:56.990
channel,uid,data,lastupdate=EMP,STO20240914151751757217921l3,2024/09/14 15:17:57.261
channel,uid,data,lastupdate=EMP,STO20240914151751757474292l2,2024/09/14 15:17:57.518
channel,uid,data,lastupdate=EMP,STO20240914151751757731882O4,2024/09/14 15:17:57.777
channel,uid,data,lastupdate=EMP,STO20240914151751757988640l8,2024/09/14 15:17:58.032
channel,uid,data,lastupdate=EMP,STO20240914151751758244031l2,2024/09/14 15:17:58.289
channel,uid,data,lastupdate=DEPT,STO20240914151751758669506360,2024/09/14 15:17:58.706
channel,uid,data,lastupdate=DEPT,STO20240914151751758914040O6,2024/09/14 15:17:58.945
channel,uid,data,lastupdate=DEPT,STO20240914151751759100370l9,2024/09/14 15:17:59.145
channel,uid,data,lastupdate=DEPT,STO20240914151751759299386l0,2024/09/14 15:17:59.330
channel,uid,data,lastupdate=SALGRADE,STO20240914151751759739040l5,2024/09/14 15:17:59.777
channel,uid,data,lastupdate=SALGRADE,STO20240914151751759884195l7,2024/09/14 15:17:59.915
channel,uid,data,lastupdate=SALGRADE,STO20240914151801800027631l4,2024/09/14 15:18:00.058
channel,uid,data,lastupdate=SALGRADE,STO20240914151801800170710l9,2024/09/14 15:18:00.201
channel,uid,data,lastupdate=SALGRADE,STO20240914151801800340229l2,2024/09/14 15:18:00.372
```

3.9 storage_delete()

- **機能概要**

  ストレージに保存されているデータを削除する。

- **関数定義**

  stat = storage_delete(channel, uid *[,*lockinfo *[,*lastupdate*]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	チャンネル文字列。"" 空文字列を指定すると "_<g_hostname>" を使用する
uid:String	データ保存時に指定したキー文字列
lockinfo:String	データがロック中の場合にロック情報を指定する
lastupdate:String	データを最後に更新した日時(YYYY/MM/DD HH:MM:SS.mmm)を指定する

- **備考**

ストレージに保存されているデータを削除する。キー(UID)に指定したデータが存在しない場合は false を返す。削除対象となったデータと関連付けられたインデックスも同時に削除する。

storage_delete() ライブラリ関数は１つのデータのみを削除対象としますが、複数のデータを削除したい場合には storage_purge() ライブラリ関数を使用します。

lastupdate を指定した場合は、更新前のデータ値に付随するタイムスタンプ値と一致しない場合は削除に失敗して stat に false を返す。lastupdate を指定しない場合はパラメータを省略するか "" 空文字列を指定する。

lockinfo を指定しない場合はパラメータを省略するか "" 空文字列を指定する。
対象データがロック中の場合は、lockinfo パラメータにロック時に指定したロック文字列を指定する。
ロック中のデータに指定していたロック文字列が lockinfo パラメータと一致しない場合や lockinfo パラメータを省略した場合には削除に失敗して stat に false を返す。

channel に "" 空文字列を指定すると "_<g_hostname>" をチャンネル名に使用する。(g_hostname はライブラリ関数を実行したコンピュータのホスト名)

- **使用例**

```
local channel,uid,data,index
channel = "Channel1"
uid = "ABC"
data = "{これは<データ3>}です"
index = {}
index["i1"] = 111
index["s1"] = '文字列インデックス111'
index["d1"] = '2011/11/11 11:22:33.456'
index["i2"] = 222.222
index["s2"] = '<<[文字列インデックス222]>>'
index["d2"] = '2011/11/22'
stat_check(storage_purge(channel))
stat_check(storage_set(channel,uid,data,index))
```

```
local uid_arr = stat_check(storage_select(channel))
log_msg("data count=" .. #uid_arr)
stat_check(storage_delete(channel,uid))
local uid_arr = stat_check(storage_select(channel))
log_msg("data count=" .. #uid_arr)
```

上記スクリプトの実行結果ログ

```
data count=1
data count=0
```

3.10 storage_inc()

- **機能概要**

  現在値を数値とみなして１インクリメントする

- **関数定義**

  stat, data = storage_inc(channel, uid *[,*lockinfo *[,*lastupdate*]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
data:String	uid で指定した UID(キー)に対応するデータ
channel:String	チャンネル文字列。"" 空文字列を指定すると "_<g_hostname>" を使用する
uid:String	データ保存時に指定したキー文字列
lockinfo:String	データがロック中の場合にロック情報を指定する
lastupdate:String	データを最後に更新した日時(YYYY/MM/DD HH:MM:SS.mmm)を指定する

- **備考**

  ストレージの uid に対応するデータに対して、その現在値を数値とみなして１をインクリメントした値を文字列形式にして設定した後、リターン値 data に返す。uid パラメータに "" 空文字列を指定することはできません。

  データが見つからないか既存のデータが数値に変換できない場合は、"1" がデータに設定されてリターンパラメータの data に "1" が返り、stat には常に true が返ります。

  lastupdate を指定した場合は、更新前のデータ値に付随するタイムスタンプ値と一致しない場合は削除に失敗して stat に false を返す。lastupdate を指定しない場合はパラメータを省略するか "" 空文字列を指定する。

  lockinfo を指定しない場合はパラメータを省略するか "" 空文字列を指定する。
  対象データがロック中の場合は、lockinfo パラメータにロック時に指定したロック文字列を指定する。

storage_set() とは違って、ロック中のデータ更新を行った場合でもロック状態は維持したままになる。

ロック中のデータに指定していたロック文字列が lockinfo パラメータと一致しない場合や lockinfo パラメータを省略した場合には削除に失敗して stat に false を返す。

- **使用例**

```
channel = 'TestCh'
uid = 'MyUID'
stat_check(storage_delete(channel,uid))
local last
for i = 1,100 do
  last = stat_check(storage_inc(channel,uid))
end
local data,lockinfo,lastupdate = stat_check(storage_get(channel,uid))
log_msg('channel,uid,data,lockinfo,lastupdate=' .. channel .. ",".. uid .. "," .. data .. "," ..
lockinfo .. "," .. lastupdate)
```

上記スクリプトの実行結果ログ

```
channel,uid,data,lockinfo,lastupdate=TestCh,MyUID,100,,2024/09/16 09:12:43.034
```

3.11 storage_dec()

- **機能概要**

現在値を数値とみなして１デクリメントする

- **関数定義**

stat, data = storage_dec(channel, uid *[,lockinfo [,lastupdate]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
data:String	uid で指定した UID(キー)に対応するデータ
channel:String	チャンネル文字列。"" 空文字列を指定すると "_<g_hostname>" を使用する
uid:String	データ保存時に指定したキー文字列
lockinfo:String	データがロック中の場合にロック情報を指定する
lastupdate:String	データを最後に更新した日時(YYYY/MM/DD HH:MM:SS.mmm)を指定する

- **備考**

ストレージの uid に対応するデータに対して、その現在値を数値とみなして１をデクリメントした値を文字列形式に設定した後、リターン値 data に返す。uid パラメータに "" 空文字列を指定することはできません。

uid に指定したデータが見つからない場合、リターン値 data には "" 空文字列が返り、stat には常に true

が返る。

デクリメント後の値が "0" 以下("0"を含む)になった場合にはデータ自身を削除する。ライブラリ関数をコールする前のデータが "1" の場合には、デクリメント後の値が "0" になるためデータが削除される。
デクリメント後の値が負の値を示す場合や、既存のデータが数値に変換できない場合にも同様にデータは削除される。この様にライブラリ関数内でデータ削除が行われた場合には、data に "" 空文字列が返り、stat には true が返る。

> **channel,uid に指定したデータが存在しない場合でもエラーになりません**
> storage_get() ライブラリ関数でのデータ取得時は channel, uid パラメータで指定したデータが存在しない場合には stat にエラーを返しますが、storage_dec() ではデータが存在しない場合でもエラーになりません。リターンパラメータ data が "" 空文字列かどうかでデータが存在するかどうかを判断します。

lastupdate を指定した場合は、更新前のデータ値に付随するタイムスタンプ値と一致しない場合は削除に失敗して stat に false を返す。lastupdate を指定しない場合はパラメータを省略するか "" 空文字列を指定する。

lockinfo を指定しない場合はパラメータを省略するか "" 空文字列を指定する。
対象データがロック中の場合は、lockinfo パラメータにロック時に指定したロック文字列を指定する。
storage_set() とは違って、ロック中のデータ更新を行った場合でもロック状態は維持したままになる。
ロック中のデータに指定していたロック文字列が lockinfo パラメータと一致しない場合や lockinfo パラメータを省略した場合には削除に失敗して stat に false を返す。

- **使用例**

```
channel = 'TestCh'
uid = 'MyUID'
stat_check(storage_set(channel,uid,"5"))
local last
for i = 1,10 do
  last = stat_check(storage_dec(channel,uid))
  log_msg('last=' .. last)
end
```

上記スクリプトの実行結果ログ

```
last=4
last=3
last=2
last=1
last=
last=
```

```
last=
last=
last=
last=
```

3.12 storage_lock()

- **機能概要**

  ストレージに保存されているデータをロック状態にする。

- **関数定義**

  stat = storage_lock(channel, uid, lockinfo [,lastupdate])

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	チャンネル文字列。"" 空文字列を指定すると "_<g_hostname>" を使用する
uid:String	データ保存時に指定したキー文字列
lockinfo:String	ロック情報を指定する。データがロック中の場合には現在のロック情報を指定する
lastupdate:String	データを最後に更新した日時(YYYY/MM/DD HH:MM:SS.mmm)を指定する

- **備考**

  ストレージに保存されているデータをロック状態にする。lockinfo にはアプリケーション側で任意の文字列を指定できる。

  キー(UID)に指定したデータが存在しない場合は false を返す。キー(UID)に指定したデータが現在ロック中の場合には、ロック文字列が lockinfo パラメータで指定した文字列と一致している場合には true を返し、一致していない場合には false が返りロック状態は以前の状態が維持される。

  lastupdate を指定した場合は、更新前のデータ値に付随するタイムスタンプ値と一致しない場合は削除に失敗して stat に false を返す。lastupdate を指定しない場合はパラメータを省略するか "" 空文字列を指定する。

  このライブラリ関数によって現在のロック状態を更新した場合でも、データのタイムスタンプは変化しない。

  channel に "" 空文字列を指定すると "_<g_hostname>" をチャンネル名に使用する。(g_hostname はライブラリ関数を実行したコンピュータのホスト名)

- **使用例**

```
channel = 'TestCh'
```

```
uid = 'MyUID'
lockinfo = 'ロックキー'
stat_check(storage_purge(channel,'',true))
stat_check(storage_set(channel,uid,'試験データ'))
stat_check(storage_lock(channel,uid,lockinfo))
local data,tmp_lockinfo,lastupdate = stat_check(storage_get(channel,uid))
log_msg('data,lockinfo=' .. data .. ',' .. tmp_lockinfo)
if not storage_set(channel,uid,'更新データ',{},lockinfo) then
  log_msg('*ERROR* storage_set() failed')
end
data,tmp_lockinfo,lastupdate = stat_check(storage_get(channel,uid))
log_msg('data,lockinfo=' .. data .. ',' .. tmp_lockinfo)
```

上記スクリプトの実行結果ログ

```
data,lockinfo=試験データ,ロックキー
data,lockinfo=更新データ,
```

3.13 storage_unlock(), storage_force_unlock()

- **機能概要**

  ストレージに保存されているデータをアンロック状態にする。

- **関数定義**

  stat = storage_unlock(channel, uid, lockinfo *[,lastupdate]*)

  stat = storage_force_unlock(channel_or_all)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	チャンネル文字列。"" 空文字列を指定すると "_<g_hostname>" を使用する
channel_or_all:String	チャンネル文字列。"" 空文字列を指定すると "_<g_hostname>" を使用する アスタリスク '*' 文字を指定すると全チャンネルが対象。
uid:String	データ保存時に指定したキー文字列
lockinfo:String	データがロック中の場合には現在のロック情報を指定する
lastupdate:String	データを最後に更新した日時(YYYY/MM/DD HH:MM:SS.mmm)を指定する

- **備考**

  ストレージに保存されているデータをアンロック状態にする。通常は storage_lock() -> storage_set() を順にコールするとアンロック状態に戻るため　storage_unlock() のコールは必要ない。

  ただし、ロック中のデータに対する storage_set() コールがエラーになった場合は、データはロック状態のままのため、後のレコード操作に支障がないように storage_unlock() を必要に応じてコールすること。

lockinfo パラメータにロック時に指定したのと同じ lockinfo 文字列を指定する。現在ロック中のデータのロック文字列が lockinfo と一致しない場合には falseが返りロック状態はそのまま維持される。

キー(UID)に指定したデータが存在しない場合は false を返す。

キー(UID)に指定したデータがロックされていなかった場合はなにもしないで stat に true を返す。

lastupdate を指定した場合は、更新前のデータ値に付随するタイムスタンプ値と一致しない場合は削除に失敗して stat に false を返す。lastupdate を指定しない場合はパラメータを省略するか "" 空文字列を指定する。

このライブラリ関数によって現在のロック状態を更新した場合でも、データのタイムスタンプは変化しない。

channel や channel_or_all に "" 空文字列を指定すると "_<g_hostname>" をチャンネル名に使用する。(g_hostname はライブラリ関数を実行したコンピュータのホスト名)

storage_force_unlock() 関数は現在のロック状態を確認しないで、ストレージ内の複数データを強制的にアンロック状態にする。channel_or_all にチャンネル名を指定するとそのチャンネル内の全データをアンロックする。channel_or_all に "*" アスタリスク文字を指定するとストレージの全データが強制的にアンロックされる。

- **使用例**

```
channel = 'TestCh'
uid = 'MyUID'
lockinfo = 'ロックキー'
stat_check(storage_purge(channel,'',true))
stat_check(storage_set(channel,uid,'試験データ'))
stat_check(storage_lock(channel,uid,lockinfo))
if not storage_set(channel,uid,'更新データ#1') then
  log_msg('*ERROR* storage_set() failed, #1')
end
stat_check(storage_unlock(channel,uid,lockinfo))
if not storage_set(channel,uid,'更新データ#2') then
  log_msg('*ERROR* storage_set() failed, #2')
end
data,tmp_lockinfo,lastupdate = stat_check(storage_get(channel,uid))
log_msg('data,lockinfo=' .. data .. ',' .. tmp_lockinfo)
```

上記スクリプトの実行結果ログ

```
*EXCEPTION* lockinfo mismatch error, recordUID = MyUID
GetDocument:*EXCEPTION* lockinfo mismatch error, recordUID = MyUID
storage_set:*EXCEPTION* SetData failed, lockinfo mismatch error, recordUID = MyUID
*ERROR* storage_set() failed, #1
data,lockinfo=更新データ#2,
```

3.14 storage_select()

- **機能概要**

  タイムスタンプ更新期間内に更新されたキー(UID)一覧を取得する。

- **関数定義**

  stat, uid_arr, break = storage_select(channel *[*,datetime_from *[*,datetime_until *[*,descend *[*,maxlist *[*,rows *[*,rows_to*]]]]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
uid_arr:Table [1..#n] of String	キー(文字列)が格納された配列
break:Boolean	検索結果が maxlist に指定した値(デフォルトは 5000) を超えた場合には、true がセットされる。この場合には uid_arrに maxlist 分のデータが格納される。それ以外の場合には false が入る。
channel:String	チャンネル文字列。"" 空文字列を指定すると "_<g_hostname>" を使用する
datetime_from:String	指定した日時以降のデータ更新日付を持つデータを検索する
datetime_until:String	指定した日時よりも前のデータ更新日付を持つデータを検索する
descend:Boolean	true 指定時はデータを新しい更新日付順に並べる。false は古い順になる。
maxlist:Number	検索結果とするレコード数の最大値を指定する。
rows:Number	検索結果とするレコードの数または最初のレコード番号を指定する。
rows_to:Number	検索結果とするレコードの最後のレコード番号を指定する。

- **備考**

  登録されたデータのタイムスタンプ値を元に、パラメータで指定したタイムスタンプ更新期間内に更新されたキー(UID)一覧を取得する。

  channel に "" 空文字列を指定すると "_<g_hostname>" をチャンネル名に使用する。(g_hostname はライブラリ関数を実行したコンピュータのホスト名)

  パラメータに指定する日付時刻のフォーマットは "YYYY/MM/DD HH:MM:SS" または "YYYY/MM/DD HH:MM:SS.mmm" にする。日付のみを指定する場合は "YYYY/MM/DD" のフォーマットで指定することで、日付時刻型の時刻部分が 0:0:0 に自動指定されて検索される。

datetime_from パラメータは検索対象となるデータのタイムスタンプの開始日付時刻を指定する。指定した日時を含むそれ以降のタイムスタンプを持つデータを検索対象となる。パラメータを使用しない場合には省略もしくは "" 空文字列を指定する。この場合にはストレージに登録されている最も古いデータから検索する。

datetime_until パラメータは検索対象となるデータのタイムスタンプの終了日付時刻を指定する。指定した日付時刻を含まない直前までのデータが検索対象となる。パラメータを使用しない場合には省略もしくは "" 空文字列を指定する。この場合にはストレージに登録されている最も新しいデータまでを検索する。

descend パラメータはデータのタイムスタンプを降順(タイムスタンプ値が新しいものから)に検索してリターン値に格納する場合には true を指定する。パラメータ省略時のデフォルトは false でタイムスタンプが古い順に格納する。

maxlist パラメータは検索結果の最大レコード数を指定する。このパラメータを省略した場合には 5000 がデフォルトで指定される。検索条件に合致するデータレコード数が maxlist に指定した数よりも多かった場合には break には true が返る。

rows パラメータのみを指定して rows_to パラメータを省略した場合は、rows に指定したレコード数分を検索対象とする。rows パラメータと rows_to を両方指定したときは、検索結果のデータレコード中からリターン値に返す最初のレコード番号が rows で rows_to に指定した値が最後のレコード番号になる。

- **使用例 「事前に storage_set() 使用例に記述したスクリプトによってデータ登録済み」**

```
local channel = "EMP"
local uid_arr = stat_check(storage_select(channel))
local data_arr = stat_check(storage_get(channel,uid_arr))
for i,v in ipairs(data_arr) do
  log_msg(v)
  wait_time(1)
end
```

上記スクリプトの実行結果ログ

```
{"EMPNO":7934,"ENAME":"MILLER","JOB":"CLERK" ,"MGR":7782,"HIREDATE":"1982/1/23"  ,"SAL":1300,"COMM":null,"DEPTNO":10}
{"EMPNO":7902,"ENAME":"FORD","JOB":"ANALYST" ,"MGR":7566,"HIREDATE":"1981/12/3"  ,"SAL":3000,"COMM":null,"DEPTNO":20}
{"EMPNO":7900,"ENAME":"JAMES","JOB":"CLERK" ,"MGR":7698,"HIREDATE":"1981/12/3"  ,"SAL":950,"COMM":null,"DEPTNO":30}
{"EMPNO":7876,"ENAME":"ADAMS","JOB":"CLERK" ,"MGR":7788,"HIREDATE":"1987/7/13"  ,"SAL":1100,"COMM":null,"DEPTNO":20}
{"EMPNO":7844,"ENAME":"TURNER","JOB":"SALESMAN" ,"MGR":7698,"HIREDATE":"1981/9/8"  ,"SAL":1500,"COMM":null,"DEPTNO":30}
{"EMPNO":7839,"ENAME":"KING","JOB":"PRESIDENT" ,"MGR":null,"HIREDATE":"1981/11/17"  ,"SAL":5000,"COMM":null,"DEPTNO":10}
{"EMPNO":7788,"ENAME":"SCOTT","JOB":"ANALYST" ,"MGR":7566,"HIREDATE":"1987/7/13"  ,"SAL":3000,"COMM":null,"DEPTNO":20}
{"EMPNO":7782,"ENAME":"CLARK","JOB":"MANAGER" ,"MGR":7839,"HIREDATE":"1981/6/9"  ,"SAL":2450,"COMM":null,"DEPTNO":10}
{"EMPNO":7698,"ENAME":"BLAKE","JOB":"MANAGER" ,"MGR":7839,"HIREDATE":"1981/5/1"  ,"SAL":2850,"COMM":null,"DEPTNO":30}
```

```
{"EMPNO":7654,"ENAME":"MARTIN","JOB":"SALESMAN" ,"MGR":7698,"HIREDATE":"1981/9/28"  ,"SAL":1250,"COMM":1400,"DEPTNO":30}
{"EMPNO":7566,"ENAME":"JONES","JOB":"MANAGER" ,"MGR":7839,"HIREDATE":"1981/4/2"  ,"SAL":2975,"COMM":null,"DEPTNO":20}
{"EMPNO":7521,"ENAME":"WARD", "JOB":"SALESMAN","MGR":7698,"HIREDATE":"1981/2/22" ,"SAL":1250,"COMM":500 ,"DEPTNO":30}
{"EMPNO":7499,"ENAME":"ALLEN","JOB":"SALESMAN","MGR":7698,"HIREDATE":"1981/2/20" ,"SAL":1600,"COMM":300 ,"DEPTNO":30}
{"EMPNO":7369,"ENAME":"SMITH","JOB":"CLERK"   ,"MGR":7902,"HIREDATE":"1980/12/17","SAL":800 ,"COMM":null,"DEPTNO":20}
```

3.15 storage_search_str()

- **機能概要**

  データ登録時に文字列で指定したインデックス値(文字列)を検索して条件に一致するキー(UID)一覧を取得する。

- **関数定義**

  stat, uid_arr, break = storage_search_str(channel ,i_key ,i_val_ptn *[,maxlist [,rows] [,rows_to]]*)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  uid_arr:Table [1..#n] of String　キー(文字列)が格納された配列

  break:Boolean　　検索結果が maxlist に指定した値(デフォルトは 5000) を超えた場合には、true がセットされる。この場合には uid_arrに maxlist 分のデータが格納される。それ以外の場合には false が入る。

  channel:String　　チャンネル文字列。"" 空文字列を指定すると "_<g_hostname>" を使用する

  i_key:String　　インデックス登録時に指定したインデックスキー名

  i_val_ptn:String　　検索対象となるインデックスキーに対応する文字列パターン

  maxlist:Number　　検索結果とするレコード数の最大値を指定する。

  rows:Number　　検索結果とするレコードの数または最初のレコード番号を指定する。

  rows_to:Number　　検索結果とするレコードの最後のレコード番号を指定する。

- **備考**

  データ登録時に文字列で指定したインデックス値を検索して条件に一致するキー(UID)一覧を取得する。

  channel に "" 空文字列を指定すると "_<g_hostname>" をチャンネル名に使用する。(g_hostname はライブラリ関数を実行したコンピュータのホスト名)

  i_key は storage_set() 関数で index パラメータに指定したインデックスのキー名で、このキー名に設定したインデックス値は文字列型である必要がある。

  i_val_ptn には検索する文字列パターンを指定する。i_val_ptn の文字列内に "%" 文字を指定して部分一致検索を行うことができる。"<検索文字列>%" は前方一致検索で "%<検索文字列>"は後方一致検索、"%<検索文字列

>%"は部分一致検索になる。"%" 文字を含めない場合は、"<検索文字列>" と完全に一致するものだけを検索する。

maxlist パラメータは検索結果の最大レコード数を指定する。このパラメータを省略した場合には 5000 がデフォルトで指定される。検索条件に合致するデータレコード数が maxlist に指定した数よりも多かった場合には break には true が返る。

rows パラメータのみを指定して rows_to パラメータを省略した場合は、rows に指定したレコード数分を検索対象とする。rows パラメータと rows_to を両方指定したときは、検索結果のデータレコード中からリターン値に返す最初のレコード番号が rows で rows_to に指定した値が最後のレコード番号になる。

- **使用例「事前に storage_set() 使用例に記述したスクリプトによってデータ登録済み」**

```
local channel = "EMP"
local uid_arr = stat_check(storage_search_str(channel,"JOB","SALES%"))
local data_arr = stat_check(storage_get(channel,uid_arr))
for i,v in ipairs(data_arr) do
  log_msg(v)
  wait_time(1)
end
```

上記スクリプトの実行結果ログ

```
{"EMPNO":7499,"ENAME":"ALLEN","JOB":"SALESMAN","MGR":7698,"HIREDATE":"1981/2/20" ,"SAL":1600,"COMM":300 ,"DEPTNO":30}
{"EMPNO":7521,"ENAME":"WARD", "JOB":"SALESMAN","MGR":7698,"HIREDATE":"1981/2/22" ,"SAL":1250,"COMM":500 ,"DEPTNO":30}
{"EMPNO":7654,"ENAME":"MARTIN","JOB":"SALESMAN" ,"MGR":7698,"HIREDATE":"1981/9/28"  ,"SAL":1250,"COMM":1400,"DEPTNO":30}
{"EMPNO":7844,"ENAME":"TURNER","JOB":"SALESMAN" ,"MGR":7698,"HIREDATE":"1981/9/8"  ,"SAL":1500,"COMM":null,"DEPTNO":30}
```

3.16 storage_search_int()

- **機能概要**

データ登録時に文字列で指定したインデックス値(整数値)を検索して条件に一致するキー(UID)一覧を取得する。

- **関数定義**

stat, uid_arr, break = storage_search_int(channel ,i_key
[,i_val_from [,i_val_to [,maxlist [,rows [,rows_to]]]]])

- **パラメータとリターン値**

stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。
uid_arr:Table [1..#n] of String　　キー(文字列)が格納された配列
break:Boolean　　検索結果が maxlist に指定した値(デフォルトは 5000) を超えた場合に

	は、true がセットされる。この場合には uid_arrに maxlist 分のデータが格納される。それ以外の場合には false が入る。
channel:String	チャンネル文字列。"" 空文字列を指定すると "_<g_hostname>" を使用する
i_key:String	インデックス登録時に指定したインデックスキー名
i_val_from:Number	インデックスの最低値(整数)
i_val_to:Number	インデックスの最高値(整数)
maxlist:Number	検索結果とするレコード数の最大値を指定する。
rows:Number	検索結果とするレコードの数または最初のレコード番号を指定する。
rows_to:Number	検索結果とするレコードの最後のレコード番号を指定する。

- **備考**

データ登録時に数値で指定したインデックス値を検索して条件に一致するキー(UID)一覧を取得する。

channel に "" 空文字列を指定すると "_<g_hostname>" をチャンネル名に使用する。(g_hostname はライブラリ関数を実行したコンピュータのホスト名)

i_key は storage_set() 関数で index パラメータに指定したインデックスのキー名です。このキー名に設定したインデックス値は整数型である必要がある。

i_val_from パラメータは検索対象となるデータのインデックス値の最低値を整数で指定する。指定した数値を含むそれ以上の i_val 値を持つデータを検索対象となる。パラメータを使用しない場合には省略もしくは nil を指定する。この場合には i_key のインデックス値を持ち且つ最も小さいインデックス値から検索する。

i_val_to パラメータは検索対象となるデータのインデックス値の最高値を整数で指定する。指定した数値を含むそれ以下の i_val 値を持つデータを検索対象となる。パラメータを使用しない場合には省略もしくは nil を指定する。この場合には i_key のインデックス値を持ち且つ最も大きいインデックス値までを検索する。

i_key の検索を数値範囲ではなく指定値のみにするときは、i_val_from と i_val_to に同じ数値を指定する。

i_val_from や i_val_toに指定した数値が浮動少数値の場合には、少数部を切り捨て整数部のみにしてから検索条件に使用される。

maxlist パラメータは検索結果の最大レコード数を指定する。このパラメータを省略した場合には 5000 がデフォルトで指定される。検索条件に合致するデータレコード数が maxlist に指定した数よりも多かった場合には break には true が返る。

rows パラメータのみを指定して rows_to パラメータを省略した場合は、rows に指定したレコード数分を検索対象とする。rows パラメータと rows_to を両方指定したときは、検索結果のデータレコード中からリターン値に返す最初のレコード番号が rows で rows_to に指定した値が最後のレコード番号になる。

- **使用例「事前に storage_set() 使用例に記述したスクリプトによってデータ登録済み」**

```
local channel = "EMP"
local uid_arr = stat_check(storage_search_int(channel,"SAL",2000,3000))
local data_arr = stat_check(storage_get(channel,uid_arr))
for i,v in ipairs(data_arr) do
  log_msg(v)
  wait_time(1)
end
```

上記スクリプトの実行結果ログ

```
{"EMPNO":7566,"ENAME":"JONES","JOB":"MANAGER" ,"MGR":7839,"HIREDATE":"1981/4/2"  ,"SAL":2975,"COMM":null,"DEPTNO":20}
{"EMPNO":7698,"ENAME":"BLAKE","JOB":"MANAGER" ,"MGR":7839,"HIREDATE":"1981/5/1"  ,"SAL":2850,"COMM":null,"DEPTNO":30}
{"EMPNO":7782,"ENAME":"CLARK","JOB":"MANAGER" ,"MGR":7839,"HIREDATE":"1981/6/9"  ,"SAL":2450,"COMM":null,"DEPTNO":10}
{"EMPNO":7788,"ENAME":"SCOTT","JOB":"ANALYST" ,"MGR":7566,"HIREDATE":"1987/7/13"  ,"SAL":3000,"COMM":null,"DEPTNO":20}
{"EMPNO":7902,"ENAME":"FORD","JOB":"ANALYST" ,"MGR":7566,"HIREDATE":"1981/12/3"  ,"SAL":3000,"COMM":null,"DEPTNO":20}
```

3.17 storage_search_datetime()

- **機能概要**

データ登録時に文字列で指定したインデックス値(日付時間値)を検索して条件に一致するキー(UID)一覧を取得する。

- **関数定義**

stat, uid_arr, break = storage_search_datetime(channel ,i_key
[,i_datetime_from [,i_datetime_until [,maxlist [,rows [,rows_to]]]]])

- **パラメータとリターン値**

stat:Boolean　成功した場合は true, 失敗した場合は false が返る。

uid_arr:Table [1..#n] of String　キー(文字列)が格納された配列

break:Boolean　検索結果が maxlist に指定した値(デフォルトは 5000) を超えた場合には、true がセットされる。この場合には uid_arrに maxlist 分のデータが格納される。それ以外の場合には false が入る。

channel:String　チャンネル文字列。"" 空文字列を指定すると "_<g_hostname>" を使用する

i_key:String　インデックス登録時に指定したインデックスキー名

i_datetime_from:String　インデックス値の最も古い日付時刻

i_datetime_until:String　インデックス値の最も新しい日付時刻。この日付時刻直前までが対象となる

maxlist:Number　検索結果とするレコード数の最大値を指定する。

rows:Number　検索結果とするレコードの数または最初のレコード番号を指定する。

rows_to:Number　検索結果とするレコードの最後のレコード番号を指定する。

- **備考**

  データ登録時に日付時刻値で指定したインデックス値を検索して条件に一致するキー(UID)一覧を取得する。

  channel に "" 空文字列を指定すると "_<g_hostname>" をチャンネル名に使用する。(g_hostname はライブラリ関数を実行したコンピュータのホスト名)

  i_key は storage_set() 関数で index パラメータに指定したインデックスのキー名で、このキー名に設定したインデックス値は日付型または日付時刻型である必要がある。

  パラメータに指定する日付時刻のフォーマットは "YYYY/MM/DD HH:MM:SS" または "YYYY/MM/DD HH:MM:SS.mmm" にする。日付のみを指定する場合は "YYYY/MM/DD" のフォーマットで指定することで、内部で日付時刻型の時刻部分が 0:0:0 の値に変換されて検索条件に使用される。

  datetime_from パラメータは検索対象となるデータのタイムスタンプの開始日付時刻を指定する。指定した日時を含むそれ以降のタイムスタンプを持つデータを検索対象となる。パラメータを使用しない場合には省略もしくは "" 空文字列を指定する。この場合にはストレージに登録されている最も古いデータから検索する。

  datetime_until パラメータは検索対象となるデータのタイムスタンプの終了日付時刻を指定する。指定した日付時刻を含まない直前までのデータが検索対象となる。パラメータを使用しない場合には省略もしくは "" 空文字列を指定する。この場合にはストレージに登録されている最も新しいデータまでを検索する。

  1日間(例えば 2001/1/1)を検索対象にする場合は、datetime_from に "2001/1/1" を指定して datetime_until には "2001/1/2" を指定する。

  maxlist パラメータは検索結果の最大レコード数を指定する。このパラメータを省略した場合には 5000 がデフォルトで指定される。検索条件に合致するデータレコード数が maxlist に指定した数よりも多かった場合には break には true が返る。

  rows パラメータのみを指定して rows_to パラメータを省略した場合は、rows に指定したレコード数分を検索対象とする。rows パラメータと rows_to を両方指定したときは、検索結果のデータレコード中からリターン値に返す最初のレコード番号が rows で rows_to に指定した値が最後のレコード番号になる。

- **使用例「事前に storage_set() 使用例に記述したスクリプトによってデータ登録済み」**

```
local channel = "EMP"
local uid_arr = stat_check(storage_search_datetime(channel,"HIREDATE",'1981/1/1','1982/1/1'))
local data_arr = stat_check(storage_get(channel,uid_arr))
for i,v in ipairs(data_arr) do
  log_msg(v)
```

```
  wait_time(1)
end
```

上記スクリプトの実行結果ログ

```
{"EMPNO":7499,"ENAME":"ALLEN","JOB":"SALESMAN","MGR":7698,"HIREDATE":"1981/2/20" ,"SAL":1600,"COMM":300 ,"DEPTNO":30}
{"EMPNO":7521,"ENAME":"WARD", "JOB":"SALESMAN","MGR":7698,"HIREDATE":"1981/2/22" ,"SAL":1250,"COMM":500 ,"DEPTNO":30}
{"EMPNO":7566,"ENAME":"JONES","JOB":"MANAGER" ,"MGR":7839,"HIREDATE":"1981/4/2"  ,"SAL":2975,"COMM":null,"DEPTNO":20}
{"EMPNO":7654,"ENAME":"MARTIN","JOB":"SALESMAN" ,"MGR":7698,"HIREDATE":"1981/9/28"  ,"SAL":1250,"COMM":1400,"DEPTNO":30}
{"EMPNO":7698,"ENAME":"BLAKE","JOB":"MANAGER" ,"MGR":7839,"HIREDATE":"1981/5/1"  ,"SAL":2850,"COMM":null,"DEPTNO":30}
{"EMPNO":7782,"ENAME":"CLARK","JOB":"MANAGER" ,"MGR":7839,"HIREDATE":"1981/6/9"  ,"SAL":2450,"COMM":null,"DEPTNO":10}
{"EMPNO":7839,"ENAME":"KING","JOB":"PRESIDENT" ,"MGR":null,"HIREDATE":"1981/11/17"  ,"SAL":5000,"COMM":null,"DEPTNO":10}
{"EMPNO":7844,"ENAME":"TURNER","JOB":"SALESMAN" ,"MGR":7698,"HIREDATE":"1981/9/8"  ,"SAL":1500,"COMM":null,"DEPTNO":30}
{"EMPNO":7900,"ENAME":"JAMES","JOB":"CLERK" ,"MGR":7698,"HIREDATE":"1981/12/3"  ,"SAL":950,"COMM":null,"DEPTNO":30}
{"EMPNO":7902,"ENAME":"FORD","JOB":"ANALYST" ,"MGR":7566,"HIREDATE":"1981/12/3"  ,"SAL":3000,"COMM":null,"DEPTNO":20}
```

3.18 storage_channel_list()

- **機能概要**

  ストレージに保存されている全チャンネル名を取得する

- **関数定義**

  stat, channel_arr = storage_channel_list()

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  channel_arr:Table [1..#n] of String　　チャンネル名(文字列)が格納された配列

- **備考**

  取得可能なチャンネル数上限は 100000 です。

- **使用例**

```
local ch_arr = stat_check(storage_channel_list())
for i,v in ipairs(ch_arr) do
  log_msg(v)
  wait_time(1)
end
```

上記スクリプトの実行結果ログ

```
<スペースチャンネル>
DEPT
EMP
```

```
MYCH1
MyCh1
OtherCH
R1
SALGRADE
TestCh
_raspi3
testch
試験チャンネル
```

3.19 storage_get_index()

- **機能概要**

  データ登録時に指定したインデックス情報を取得する

- **関数定義**

  stat, index_tbl = storage_get_index(channel, uid)

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  channel:String　　チャンネル文字列。"" 空文字列を指定すると "_<g_hostname>" を使用する

  uid:String　　データ保存時に指定したキー文字列

  index_tbl:Table of (String or Number)

  　　storage_set() で登録したインデックス(キーと値のペア)を格納したテーブル

- **備考**

  チャンネル内にデータを登録したときに使用された全てのインデックスが返る。

  index_tbl テーブルの構造は storage_set() コール時に指定した index_tbl テーブルと同等になる。ただし、コール時に浮動少数値をインデックス値に指定した場合には登録時に整数値に変換されているため、このライブラリ関数で得られるテーブルのインデックスの値も変換後の値になる。

  channel に "" 空文字列を指定すると "_<g_hostname>" をチャンネル名に使用する。(g_hostname はライブラリ関数を実行したコンピュータのホスト名)

- **使用例「事前に storage_set() 使用例に記述したスクリプトによってデータ登録済み」**

```
local channel = "EMP"
local uid_arr = stat_check(storage_search_str(channel,"JOB","PRESIDENT"))
local index = stat_check(storage_get_index(channel,uid_arr[1]))
```

```
for k,v in pairs(index) do
  log_msg('key:' .. tostring(k) .. '(' .. type(v) .. ') value:' .. tostring(v))
end
```

上記スクリプトの実行結果ログ

```
key:DEPTNO(number) value:10
key:EMPNO(number) value:7839
key:JOB(string) value:PRESIDENT
key:SAL(number) value:5000
key:HIREDATE(string) value:1981/11/17
key:ENAME(string) value:KING
```

3.20 storage_purge()

- **機能概要**

  古いタイムスタンプを持つデータを削除する

- **関数定義**

  stat, count = storage_purge(channel_or_all *[,*datetime_until *[,*force*]]*)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
count:Number	削除したデータ数が返る。
channel_or_all:String	チャンネル文字列。"" 空文字列を指定すると "_<g_hostname>" を使用する アスタリスク '*' 文字を指定すると全チャンネルが対象。
datetime_until:String	指定した日時よりも前のデータ更新日付を持つデータを削除する
force:Boolean	ロック中のデータを強制的に削除する場合は true を指定する。 false を指定して、かつ削除対象のデータがロック中の場合は削除されない。

- **備考**

  datetime_until パラメータに指定した日付よりも古いタイムスタンプを持つデータと削除対象となったデータと関連付けられたインデックスも削除する。

  datetime_until パラメータを省略、または '' 空文字列を指定した場合はチャンネル内に格納されている全てのデータが削除される。

  ロック中のデータは削除対象から除かれてロックされていないデータのみを削除する。force パラメータに true を指定すると(省略時は false)ロック中のデータを含めて削除する。

  channel_or_all に "" 空文字列を指定すると "_<g_hostname>" をチャンネル名に使用する。(g_hostname は

ライブラリ関数を実行したコンピュータのホスト名)

channel_or_all に "*" アスタリスク文字を指定すると全チャンネルが対象になる。例えばスクリプト内で、storage_purge('*') を実行すると、ストレージ内の全データを削除する。

- **使用例「事前に storage_set() 使用例に記述したスクリプトによってデータ登録済み」**

```
local cnt = stat_check(storage_purge('EMP'))
log_msg('removed:' .. tostring(cnt))
```

上記スクリプトの実行結果ログ

```
removed:14
```

3.21 storage_transfer()

- **機能概要**

リモートホストの abs_agentにデータの複製を作る

- **関数定義**

stat = storage_transfer(channel, uid, remote_host)

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
channel:String	チャンネル文字列。"" 空文字列を指定すると "_<g_hostname>" を使用する
uid:String	データ保存時に指定したキー文字列
remote_host:String	リモートコンピュータのホスト名または IP アドレス

- **備考**

リモートホストの abs_agentに、チャンネル,キーで指定したデータの複製を作る。

複製時は同一チャンネル・キー(UID)名でリモート側にデータを作成する。複製元にある index レコードも同時にリモート側に複製する。

複製元対象データがロック中の場合でもデータ複製は可能。複製先のデータには複製元データのロック文字列はコピーしないで常にアンロック状態で作成する。

複製先に既存データが存在する場合には、複製前に既存データとindex を削除してからデータを作成する。ただし、リモート側の既存の複製先データがロック中の場合は false を返し複製できない。

channel に "" 空文字列を指定すると "_<g_hostname>" をチャンネル名に使用する。(g_hostname はライブラリ関数を実行した複製元コンピュータのホスト名)

- **使用例 「事前に storage_set() 使用例に記述したスクリプトによってデータ登録済み」**

```
local channels = {"EMP","DEPT","SALGRADE"}
local remote_host = "192.168.100.11"
for i,ch in ipairs(channels) do
  local uid_arr = stat_check(storage_select(ch))
  for j,uid in ipairs(uid_arr) do
    stat_check(storage_transfer(ch,uid,remote_host))
  end
end
```

3.22 uid_arr_or()

- **機能概要**

  UID 配列(文字列配列)の和集合を作成

- **関数定義**

  uid_arr = uid_arr_or(uid_arr#1 [,uid_arr#2,...,uid_arr#m])

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  uid_arr:Table [1..#n] of String　　uid_arr#1..uid_arr#m 配列要素の和集合が格納された配列

  uid_arr#m:Table [1..#n] of String　キー(文字列)が格納された配列

- **備考**

  uid_arr#1 から uid_arr#m に格納されている文字列配列要素の和集合(union)配列を返す。重複した配列要素は取り除かれる。

- **使用例「事前に storage_set() 使用例に記述したスクリプトによってデータ登録済み」**

```
local channel = "EMP"
local uid_arr1 = stat_check(storage_search_str(channel,"ENAME","A%"))
local uid_arr2 = stat_check(storage_search_str(channel,"ENAME","B%"))
local uid_arr3 = stat_check(storage_search_str(channel,"ENAME","C%"))
local uid_arr = uid_arr_or(uid_arr1,uid_arr2,uid_arr3)
local data_arr = stat_check(storage_get(channel,uid_arr))
for i,v in ipairs(data_arr) do
  log_msg(v)
  wait_time(1)
end
```

上記スクリプトの実行結果ログ

```
{"EMPNO":7782,"ENAME":"CLARK","JOB":"MANAGER" ,"MGR":7839,"HIREDATE":"1981/6/9"  ,"SAL":2450,"COMM":null,"DEPTNO":10}
{"EMPNO":7499,"ENAME":"ALLEN","JOB":"SALESMAN","MGR":7698,"HIREDATE":"1981/2/20" ,"SAL":1600,"COMM":300 ,"DEPTNO":30}
{"EMPNO":7698,"ENAME":"BLAKE","JOB":"MANAGER" ,"MGR":7839,"HIREDATE":"1981/5/1"  ,"SAL":2850,"COMM":null,"DEPTNO":30}
{"EMPNO":7876,"ENAME":"ADAMS","JOB":"CLERK" ,"MGR":7788,"HIREDATE":"1987/7/13"  ,"SAL":1100,"COMM":null,"DEPTNO":20}
```

3.23 uid_arr_and()

- **機能概要**

  UID 配列(文字列配列)の共通集合を作成

- **関数定義**

  uid_arr = uid_arr_and(uid_arr#1 [,uid_arr#2,..,uid_arr#m])

- **パラメータとリターン値**

  stat:Boolean　　成功した場合は true, 失敗した場合は false が返る。

  uid_arr:Table [1..#n] of String　　uid_arr#1..uid_arr#m 配列要素の共通集合が格納された配列

  uid_arr#m:Table [1..#n] of String　キー(文字列)が格納された配列

- **備考**

  uid_arr#1 から uid_arr#m に共通して格納されている文字列配列要素の共通集合(intersection)配列を返す。

- **使用例「事前に storage_set() 使用例に記述したスクリプトによってデータ登録済み」**

```
local channel = "EMP"
local uid_arr1 = stat_check(storage_search_str(channel,"JOB","CLERK"))
local uid_arr2 = stat_check(storage_search_int(channel,"SAL",nil,1000))
local uid_arr3 = stat_check(storage_search_int(channel,"DEPTNO",20,20))
local uid_arr = uid_arr_and(uid_arr1,uid_arr2,uid_arr3)
local data_arr = stat_check(storage_get(channel,uid_arr))
for i,v in ipairs(data_arr) do
  log_msg(v)
  wait_time(1)
end
```

上記スクリプトの実行結果ログ

```
{"EMPNO":7369,"ENAME":"SMITH","JOB":"CLERK"  ,"MGR":7902,"HIREDATE":"1980/12/17","SAL":800 ,"COMM":null,"DEPTNO":20}
```

3.24 uid_arr_sub()

- **機能概要**

UID 配列(文字列配列)の差集合を作成

- **関数定義**

  uid_arr = uid_arr_sub(uid_arr#1 [,uid_arr#2,..,uid_arr#m])

- **パラメータとリターン値**

stat:Boolean	成功した場合は true, 失敗した場合は false が返る。
uid_arr:Table [1..#n] of String	uid_arr#1 の配列要素から uid_#2..uid_arr#m の配列要素を取り除いた集合が格納された配列
uid_arr#m:Table [1..#n] of String	キー(文字列)が格納された配列

- **備考**

  uid_arr#1の配列要素から uid_arr#2 .. uid_arr#m に格納されている文字列配列要素を取り除いた差集合(difference)配列を返す。

- **使用例「事前に storage_set() 使用例に記述したスクリプトによってデータ登録済み」**

```
local channel = "EMP"
local uid_arr1 = stat_check(storage_search_int(channel,"SAL",3000,nil))
local uid_arr2 = stat_check(storage_search_str(channel,"JOB","PRESIDENT"))
local uid_arr = uid_arr_sub(uid_arr1,uid_arr2)
local data_arr = stat_check(storage_get(channel,uid_arr))
for i,v in ipairs(data_arr) do
  log_msg(v)
  wait_time(1)
end
```

上記スクリプトの実行結果ログ

```
{"EMPNO":7902,"ENAME":"FORD","JOB":"ANALYST" ,"MGR":7566,"HIREDATE":"1981/12/3"  ,"SAL":3000,"COMM":null,"DEPTNO":20}
{"EMPNO":7788,"ENAME":"SCOTT","JOB":"ANALYST" ,"MGR":7566,"HIREDATE":"1987/7/13"  ,"SAL":3000,"COMM":null,"DEPTNO":20}
```

4 行列・ベクトル操作機能 (MATRIX)

行列・ベクトル機能は現在開発中です。ライブラリ関数は追加、修正中ですので本番運用には使用しないでください

4.1 セットアップ

abs_agent の行列操作機能は内部で CBLAS, LAPACKE ライブラリを利用しています。行列操作ライブラリ関数を有効にするためには、abs_agent が動作するコンピュータにこれらのライブラリパッケージと依存パッケージを予めインストールします。例えば、以下のコマンドでバイナリ・パッケージをインストールできます。

```
sudo apt-get update
sudo apt-get install libblas-dev liblapack-dev liblapacke-dev
```

次に、abs_agent 起動時に指定するコンフィギュレーションファイルの Convert/UseMatrix 項目を True(初期値は False) に設定した後、abs_agent を再起動させることで行列操作用ライブラリ関数が使用可能になります。
同じくコンフィギュレーションファイル中にある Convert/CBLAS_Library, Convert/LAPACKE_Library 項目にライブラリファイル名を指定すると、abs_agent がロードする CBLAS, LAPACKE 共有ライブラリパッケージを動作環境に合わせて変更することができます。

```
<Convert>
  <AutoOnline type="boolean">True</AutoOnline>
  <UseMatrix type="boolean">True</UseMatrix>
  <CBLAS_Library type="string">libblas.so</CBLAS_Library>
  <LAPACKE_Library type="string">liblapacke.so</LAPACKE_Library>
  <DGEEV_AutoScale type="boolean">True</DGEEV_AutoScale>
</Convert>
```

4.2 ベクトルと行列のフォーマット

行列データをテーブル配列に格納するときに、配列先頭から２つのエントリに行数と列数を表す整数値を入れます。続いて３つめのエントリから行列の要素データを、"Row Major"順で格納します。(左から右に水平方向にスキャン後、次の行(下)に移って再び水平方向のスキャンを最終行[下端]まで繰り返す)

ベクトルデータをテーブルに格納する場合は要素データのみを格納します。テーブルに格納するフォーマットでは、行ベクトルと列ベクトルの区別はありません。

行列・ベクトル操作 API関数では行列パラメータまたはベクトルパラメータのどちらで指定可能かは予め決められています。必ず一致したフォーマットで作成されたテーブルをパラメータに渡してください。また、行列・ベクトル操作APIでは演算結果の配列は左辺のリターンパラメータに返り、右辺のパラメータに渡した配列は変化しません。

例1：行列 ***A*** (m行 n列) の場合

$$\begin{matrix} a_{11} & a_{12} & .. & a_{1n} \\ a_{21} & a_{22} & .. & a_{2n} \\ .. & & .. & \\ a_{m1} & a_{m2} & .. & a_{mn} \end{matrix}$$

行列パラメータで指定するテーブル配列は以下になる。

$$A = \{m, n, a_{11}, a_{12}, \ \ldots \ , a_{1n}, a_{21}, a_{22}, \ \ldots \ , a_{mn}\}$$

例2：行ベクトル ***x*** の場合

$$x_1 \quad x_2 \quad .. \quad x_n$$

ベクトルパラメータで指定するテーブル配列は以下になる。

$x = \{x_1, x_2, \ldots, x_n\}$

例3：列ベクトル ***y*** の場合

y_1

y_2

..

..

y_m

ベクトルパラメータで指定するテーブル配列は以下になる。

$y = \{y_1, y_2, \ldots, y_m\}$

ライブラリ関数のリターン値で取得するベクトル値や行列値のフォーマットも前述のテーブル構造になります。可変数のベクトルや行列値を返す関数は、テーブルコンストラクタ内でコールすることでリターン値の処理がやりやすくなります。

例4: 下記の様に定義された matrix_egv() 関数をコールする場合に

stat,eg_r,eg_i,ev1_r,ev1_i,ev2_r,ev2_i, .. ,ev_n_r,ev_n_i = matrix_egv(A)

テーブルコンストラクタ内でコールする記述は以下になります

local ret = {matrix_egv(A)}

このとき、ret[1],ret[2],ret[3],ret[4],ret[5],ret[6]... には matrix_egv() のリターン値 stat,eg_r,eg_i,ev1_r,ev1_i,ev2_r... が格納されています。

4.3 matrix_print(), matrix_vprint()

- **機能概要**

  行列またはベクトルの内容をログサーバーに出力する

- **関数定義**

  matrix_print(***A*** *[,title]*)

  maxtrix_vprint(***x*** *[,title [,column_vector]]*)

- **パラメータとリターン値**

A:Table [1..(2+m*n)] of Number	行列(m行 n列)を格納した数値配列
x:Table [1..n] of Number	ベクトルを格納した数値配列
title:String	タイトル名
column_vector:boolean	列ベクトルのフォーマットで出力。省略時は false

- **備考**

  行列またはベクトルの内容をログサーバーに出力する。

  ベクトルフォーマットは行ベクトルとして出力します。column_vector を true に指定すると列ベクトルフォーマットに変更します。

- **使用例**

```
maxtrix_print({2,2,1,2,3,4})
maxtrix_vprint({1,2,3},"X_Param",true)
```

4.4 matrix_index()

- **機能概要**

  行列を格納したテーブル配列のインデックス値取得

- **関数定義**

  idx = matrix_index(***A*** ,i,j)

- **パラメータとリターン値**

idx:Number	行列のi行 j列要素にアクセスするテーブル配列のインデックス値
A:Table [1..(2+m*n)] of Number	行列(m行 n列)を格納した数値配列
i:Number	行列要素の行数。1 から m までの整数値
j:Number	行列要素の列数。1 から n までの整数値

- **備考**

- **使用例**

```
local A = {5,3,       -- 5x3 matrix
           1,2,3,
           4,5,6,
           7,8,9,
           10,11,12,
           13,14,15
}
matrix_print(A,"A")
local m,n = matrix_dim(A)
log_msg(string.format('(m,n)=(%d,%d)',m,n))
A[matrix_index(A,2,2)] = 0.000001
A[matrix_index(A,1,1)] = 99.99
A[matrix_index(A,5,3)] = 11.11
matrix_print(A,"A")
```

上記スクリプト実行結果

```
A = [
  1  2  3
  4  5  6
  7  8  9
  10  11  12
  13  14  15
]
(m,n)=(5,3)
A = [
  99.99  2  3
  4  1e-06  6
  7  8  9
  10  11  12
  13  14  11.11
]
```

4.5 matrix_dim()

- **機能概要**

行列を格納したテーブル配列から行数と列数取得

- **関数定義**

  m,n = matrix_dim(***A***)

- **パラメータとリターン値**

m:Number	行列の行数。
n:Number	行列の列数。
A:Table [1..(2+m*n)] of Number	行列(m行 n列)を格納した数値配列

- **備考**

- **使用例**

  matrix_index() の項を参照してください

4.6 matrix_dot()

- **機能概要**

  ベクトル x, y のドット積

  dot ← $\boldsymbol{x}^T\boldsymbol{y}$

- **関数定義**

  stat, dot = matrix_dot(***x***, ***y***)

- **パラメータとリターン値**

stat:boolean	正常終了時は true,エラー発生時は falseが返る。
dot:Number	ベクトル ***x***,***y*** のドット積
x:Table [1..n] of Number	ベクトルを格納した数値配列。
y:Table [1..n] of Number	ベクトルを格納した数値配列

- **備考**

  x,y 双方のベクトル要素数(n)は等しいこと。

- **使用例**

```
local dot = stat_check(maxtrix_dot({1,2,3},{4,5,6}))
```

4.7 matrix_nrm2()

- **機能概要**

  ベクトル x のユークリッドノルム

  nrm2 ← $\|\boldsymbol{x}\|_2$

- **関数定義**

  stat, nrm2 = matrix_nrm2(***x***)

- **パラメータとリターン値**

stat:boolean	正常終了時は true,エラー発生時は falseが返る。
nrm2:Number	ベクトル ***x*** のユークリッドノルム
x:Table [1..n] of Number	ベクトルを格納した数値配列

- **備考**

- **使用例**

```
local nrm = stat_check(maxtrix_nrm2({1,2,3}))
```

4.8 matrix_vmul()

- **機能概要**

  行列A とベクトルx の積にベクトルy を足す

  $\boldsymbol{v} \leftarrow \boldsymbol{Ax}$

  $\boldsymbol{v} \leftarrow \alpha \boldsymbol{Ax}$

  $\boldsymbol{v} \leftarrow \alpha \boldsymbol{Ax} + \boldsymbol{y}$

  $\boldsymbol{v} \leftarrow \alpha \boldsymbol{Ax} + \beta \boldsymbol{y}$

- **関数定義**

  stat, ***v*** = matrix_vmul(***A***, ***x*** *[, α [**y** [, β]]]*)

- **パラメータとリターン値**

stat:boolean	正常終了時は true,エラー発生時は falseが返る。
v:Table [1..m] of Number	演算結果ベクトルを格納した数値配列。
A:Table [1..(2+m*n)] of Number	行列(m行 n列)を格納した数値配列
x:Table [1..n] of Number	ベクトルを格納した数値配列
α:Number	***Ax*** をスカラー倍する値。 パラメータ指定時に定数倍しない場合は 1 を指定する。
y:Table [1..m] of Number	ベクトルを格納した数値配列
β:Number	***y*** をスカラー倍する値。

- **備考**

  行列A の列数 nは x ベクトルの要素数と同じにする。また、vベクトルの要素数は行列A の行数 mに等しくなり、y ベクトルの要素数と同じにする。

- **使用例**

```
local A = {5,3,       -- 5x3 matrix
```

```
  1,2,3,
  4,5,6,
  7,8,9,
  10,11,12,
  13,14,15
}
local x = {1,2,3}
local y = {10,10,10,10,10}
local v
matrix_print(A,"A")
maxtrix_vprint(x,"x",true)
maxtrix_vprint(y,"y",true)
v = stat_check(matrix_vmul(A,x))
maxtrix_vprint(v,"v1")
v = stat_check(matrix_vmul(A,x,2.0))
maxtrix_vprint(v,"v2")
v = stat_check(matrix_vmul(A,x,2.0,y))
maxtrix_vprint(v,"v3")
v = stat_check(matrix_vmul(A,x,2.0,y,10.0))
maxtrix_vprint(v,"v4")
```

上記スクリプト実行結果

```
A = [
  1  2  3
  4  5  6
  7  8  9
  10  11  12
  13  14  15
]
x = [
  1
  2
  3
]
y = [
  10
  10
  10
  10
  10
```

```
]
v1 = [  14  32  50  68  86  ]
v2 = [  28  64  100  136  172  ]
v3 = [  38  74  110  146  182  ]
v4 = [  128  164  200  236  272  ]
```

5 更新履歴

REV A.1.2b 2024/11/6

uid_arr_sub() 関数追加

REV A.1.1b 2024/10/8

誤字修正

REV A.0.1 2024/3/2

初版作成