```
/*** Licensed under the Agpl: http://www.gnu.org/licenses/agpl-3.0.html ***/
/*** Author: http://namzezam.wikidot.com/                              ***/
/*** Preamble: http://namzezam.wdfiles.com/local--files/start/rcoin.txt ***/
/*** Document's Structure:
~Concept; ~Terminology&principles;~Tables&Legend;~Issuing-coins;~Coin's-Calculus;
~Authentication;~code;~InProcess, where  *** this is folded */
#ifndef defined_rcoin
#define defined_rcoin /*** ~Concept: rcoin - A coin of respect is       ***
a time limited and equally re-distributed cyclic and communal coin.
It is daily and gradually losing its value, which is equally gained by the
community members. It is not money, nor an equivalent to money, but still a
medium of exchange, a credit for exchange and an inner community evaluation tool.
Use it to build up your economy while bringing more social justice into your
communities for, by advancing community members to get more than the others,
only as they are automatically sharing something of their gain with their
community members.
        In simple words: When I have 10 rcoins for 50 yeas in a 4 people's
community, then in the next year I have only 9.80 rcoins and each member gets
additional 0.05 rcoin and so, as I earn more than you in your community, you,
as any other community's member, would get some equal share of it. So it make
those who can earn more to be more supported because the other would earn form
something of that. Additionally, when non of us spend rcoin even though the
rcoin nature is of losing value, only those of us having more would lose, as
the others are earning, but only until we got equal and hence it is useful for
saving communally and not in isolation, for our common interest.
        Use case: A coffee-shop and a bakery are 2 divisions in an association
named here "ring". The ring buy chocolate and give it for rcoins to
the bakery and the bakery making from that a cake give it for coins of respect
to the coffee-shop, as the ring (not the coffee-shop) sells the cake+coffee in
money to an outer/tourist client or give it in rcoins to its
community members. When it is desired to become more open, such ring might
become comcomized: http://is-with.wikidot.com/6-points .
        About the rate of exchange of the rcoin, we are not concerned here, as
we assume the exchange is not a direct but a substitutable one, meaning when
the exchange between currencies is done through the price of goods services
and holdings.
        In parallel and just by papers: The rcoin can be used by having on the
paper the  StartValue, LastDate, Lifetime and Id of the rcoin together with a
(trademaked) stump of the community and signatures of the hands it was
passing thorough:
 SidaA(constant)                 SideB(variables)
A1.Group's Stamp.                B1. Number of signatories coin
A2.LastDate                      B2. Signature of checker of sideB
A3.LifeTime                      B3. checking date
A4.StartValue                    B4. checker id
A5.CoinId                        B5++. Signatures and id/pin of the coin owners
A6.Signature of the issuer

The rcoin in allegory to compost pool:
  A) The lost of value is like the release of the energy to the air
     after being bad distributed as bobbles in the pool;
  B) The energy, effective only in the pool, is as the coin in the
     community and
  C) The well distributed dividend is like injecting bottom-up some clean
     instance (like cool water) for having the volume (or height) of the medium
      be unchanged or somewhat controlled, as is the case in printing local
      money.
Some other way to describe this system: the rcoin relates the exchanging of
(money/energy) to the time and the distribution, which makes some equivalents
to wave theories, where StartValue==amplitude/wave_max, LastDate==front_wave
and lifetime ==Wavelength ( faster/lighter<-> slower/heavier).

The study of changes in a line value of the coin's holder versus in a plane
value of other members can be made in 3d and 1 colour, where X = CoinAge,
Y = CoinValue and Z = MembersCount or in 2d and 2 colours: one of the owner
and the other of members (and as X = CoinAge and Y= CoinValue).*/

/* The use of the composition is as a key for hashed picture. The composition is
a set of the pair of arguments: (diagonal, angle), together defining a sequence
of rectangles, such that:
 Each rectangle is hashed separately in the crop defined by its arguments;
 The composition is given separately (in a specific transaction) and all
  rectangles in their order define together the reference to the
  picture and reference could be hashed again for to be squeezed again
  to a predefined size (used as an id);
 Overlapping means hashing part of hidden rectangle with the shown one;
 In each cycle the shortest rectangle on wall defines the highest of all others;
 The first two zeros define the containing rectangle and each other zero define
  changing direction until the pair of zeros that define the end after which
  the next 2 pairs are of the original picture (first of the position of its
  top left corner and the second of itself) and then terminate.
eg: (0,0)(0,0)(45,400) is the non-overflowing composition of only the
original being a square of which diagonal=400 pixels.*/

/*** ~Terminology&Principles for the rcoin:                                 ***
* The Communication of threads between members is by asymmetric keys, where *
*  threads, per each message n in an otr conversation, are defined so that  *
* t[n]=(message[n],date,hash(t[n-1])),                                      *
* hash(t[n]) is indexed by public_key(sender) and                           *
* t[0]=(id(receiver),date,x), where, as in hashcach K(hash(t[0]))==0,       *
* K is defined by number of bits to be examined and x is a random.          *
* The triplepin is the unique id of the member, which is given only in      *
* community depended conditions, such as only after having some             *
* recommenders for the uniqueness and the form of meeting with the member   *
* and of the recommenders.                                                  *
* id(coin) is a unique&random int.                                          *
* Payer is the previous Owner of a coin.                                    *
* id(member)=hash(pic(member));Changeable + retrievable by triplepin(member)*
*  The owner in payment should first see the payer then type the triplepin  *
*  by which the pic is retrieved, and only after the pic matches the payer, *
*  that pic should be hashed and used/compared as an id. Hence such protocol*
* is based on a human recognition (and not on the one of machine).          *
* Rand, used as a transaction id in the distributed log, is a unique and    *
*  random number, which is used as a receipt. It is produced by the Payer   *
*  distributing that record; So that in payment, when paying and after      *
*  proving ownership, the payer sign the new owner's id with the payer's    *
*  rand, to create her/his new distributed record(rans).                    *
***                                                                        ***/
/*** ~Tables&Legend, search: ?{?=>[?]}@                                    ***
* table{key=>[col(value1,value2)|col2(value3)]                              *
*      }@db                                                                 *
* related tables:category=<table1@db [connection] table2@db>                *
* pic          means compressed image                                       *
* si[x](data)  means detached-signed by x == (data,aep[x](hash(data)))      *
* se[x](data)  means symetrically encrypted data by the key of              *
* ae[x](data)  means asymetrically encrypted data by the public key  of x   *
* aep[x](data) means asymetrically encrypted data by the private key of x   *
*                                                                           *
* The data is stored in encrypted directory including this app in 2 db:     *
*  the My_db and the Op_db, where                                           *
*  the My_db is used by any member,                                         *
*  the Op_db is used by one or more operators being members.                *
*  (and hence when each member is also an operator, the app is a p2p app,   *
*  when being operator is rotated between members the app is democratic,    *
*  otherwise centric).                                                      *
* Here are the 3 data categories:                                           *
*  Movements    = <Coins@op[id(coin)] Wallet@My [Rand]Log@op,CoinsId@op>    *
*  Values       =<Worth@My [same-format]Treasury@op>                        *
*  Identities   =<Self@My[pic]Payers@My , Users@op[register]Profiles@op>    *
*  |-----------------------------------------------------------------------|*
*  |  category     |     @op               |    @My        |  using time   |*
*  |---------------|-----------------------|---------------|---------------|*
*  |Movements      |Coins ,  CoinsId, Log  |Wallet         | Payment       |*
*  |Values         |Treasury               |Worth          | Calculation   |*
*  |Identities     |Users,   Profiles      |Self,  Payers  | Authentication|*
*  |-----------------------------------------------------------------------|*
* Any access/modification in sensitive and common area is resulted in       *
*  parallel notification to all other members or operators                  *
* note: in big communities it my be considered to use hierarchies of hubs   *
* being operators for schemas of notification such as peer to op as peer to *
* next op etc.                                                              *
||------------------------||------------------------||------------------------||
|| table:                 ||             @Op        ||          @My           ||
||Key=>                   ||                        ||                        ||
||[col1(value1,value2)|   ||                        ||                        ||
||col2(value3)]           ||                        ||                        ||
||------------------------||------------------------||------------------------||
|| Movements in           || Coins: 2blob of all    ||                        ||
|| recycling              || valid and expired        ||                        ||
||                        || Hash(Id(coin))           ||                        ||
||------------------------||-------------------------||------------------------||
|| Movements in           || CoinsID:hash(id(coin))<id(coin)>                  ||
|| Payment                || =>[(Rand,N)]             || Wallet:hash(id(coin))=>||
||                        ||          ----            ||[(Rand, RandPrev, Coin, ||
||                        || Log:hash(Rand)=>   <Rand>  id(payer),              ||
||                        ||[(Nhash(id(Coin)),        ||   pub-key(id(payer)))] ||
||                        ||si[Payer](Rand,           ||                        ||
||                        ||id(Owner)),               ||                        ||
||                        ||Chain)]                   ||                        ||
||------------------------||-------------------------||------------------------||
||  Values in Calculation ||Treasury:LastDate=>     <*> Worth:LastDate=>        ||
||                        ||  [CoinLifetime(          || [CoinLifetime(         ||
||                        ||   SumStartValue,         ||  SumStartValue,        ||
||                        ||   TheirAmount)|...|]     ||  TheirAmount...)|...|] ||
||------------------------||-------------------------||------------------------||
|| Identities in          ||    <register>      <triplepin>        <pic>        ||
||  Authentication        ||                         ||                         ||
||                        ||Users:hash(triplepin)=>  ||Payers:hash(triplepin)=>|
||                        ||[(register=ALL(hash(     ||[(id=hash(pic),pubkey)]||
||                        ||pic(member)),,))]        ||         ----            ||
||                        ||          ----           ||Self:CreatingDate =>     ||
||                        ||Profiles:hash(register)=>||[(pic of mine)]          ||
||                        ||[(personal info in common||                         ||
||                        || pubkey, id=hash(pic,,)] ||                         ||
||------------------------||-------------------------||-------------------------||
```

Movements = <Coins@op[id(coin)] Wallet@My [Rand]Log@op,CoinsId@op>

Coins@op = the Movements of all coins starts and ends here. It consists of One table having one record having 2 blob: 1 of all Hash(Id(coin)) of valid coins (parallel to CoinsId@op)and the other of those which are expired, to be used for maintenance of their uniqueness before issuing new coins.

Q?? should n't Coins@op have id LastDate Lifetime of coin for no collusion and keeping coherence without having StartValue??

Wallet@My={hash(id(coin))=>[(Rand,RandPrev,coin,id(payer),pub-key(id(payer)))] }@My (of this member's coins), where only by id(coin) the access to the value of the coin is given!

CoinsId@op={hash(id(coin))=>[(Rand,N)] }@op (of all coins), Rand is the last Rand of the translation made with that coin for to insure no twice payment and, used for the prove of continuity, N is increased by 1 with each transaction of the coin.

Log@op={hash(Rand)=>[(Nhash(id(Coin)),si[Payer](Rand,id(Owner)),Chain)], }@op (of all coins), where Chain=(hash(ChianPrev),hash(RandPrev,Id(Owner))) and the hash(pic)==id(user) and a unique triplepin is used as a key for all such pic, making each pic able to be changed Not as in the biometric info! The Prove of ownership by id(coin), where op has in CoinsId@op Rand equals the Rand the owners pull from Log@op by her/his Wallet@My: the Rand or RandPrev has the id(coin) and the owner verifies the signature and produces both: the Chain and the hash(Id(Owner) of the ChianPrev, which is the Chain in RandPrev. N is the number of hashes implemented on itself beginning in id(coin) and ending in Nhash , for creating a prove of continuity. So, having the id(coin) and N you can create the Nhash of the N, where Nhash(N) =hash(Nhash(N++)) .

Protocol of Payment: payer send pub-enc to all op 1) prove ownership, 2) new transaction and 3) new Rand and N to replace the one in Coins@op and of which hash indexes the transaction as a new record in Log@op. Each op before creating the record verifies the transaction and only on success sends success-signal to other op and only after all op agree on success they create the transaction as a new record in Log@op. The verification is successful only when Nhash all the way from the id(coin) and until N is coherent and N-1 and si[payer] together with hash(Id(Owner,RandPrev) (included in Chain) on RandPrev are reproduced and coherent.

Protocol of Issuing-coins: see ~Issuing-coins.

Values=<Worth@My [same-format]Treasury@op|...|>

Treasury@op={LastDate=>[CoinLifetime(SumStartValue,TheirAmount)]|...|, }@op (of all coins), where SumStartValue = Sum(CoinStartValue) is only a statistical info (separated from their id) of the coins. It can be used for liquidizing by issuing some rcoins as rcoin-to-currency_X, of which dividend is paid in currency_X by using the additional Treasury as Treasury_X. e.g. Treasury_dollar for dividending in dollars. it can

```
         also be integrated with liquidizing to rcoin of other held/holding
         rings or to money given to exchange by newcomers or members.
Worth@My={(is_mine)LastDate=>
        [CoinLifetime(SumStartValue,TheirAmount,
          "-"SumStartValue/CoinLifetime,
          "+"SumStartValue/(CoinLifetime*MembersAmount) ,
            List(StartValue,id(coin)),
            SumValue)]|...|,
       }@My (of this member's coins), where
       SumValue = SumStartValue
             -((SumStartValue *(TodayDate+Lifetime-LastDate))/Lifetime)
       is the only the one which is daily changing.
Protocol of calculation: see ~Coin's-Calculus.

Identities=<Self@My[pic]Payers@My, Users@op[register]Profiles@op>
Self@My= {date-of-creation =>[(pic of mine)],
      }@My (of this member)
Payers@My={hash(triplepin)=>[(id=hash(pic),pubkey)],
      }@My (of authenticated members by this member),
     Used as in WebOfTrust, such that the payer sends
     enc(hash(triple),id,pic) and these 3 conditions has been met:
     1. id = hash(pic),
     2. pic match the payer being recognized by the payee and
     3. triplepin of the payer in Users@op+Profile@op is verified.
Users@op= {hash(triplepin) =>[(register=ALL(hash(pic(member)),,))],
      }@op (of all members), payers ae to be verified.
Profiles@op={hash(register)=>[(personal info in common: pubkey, id=hash(pic,,)],
      }@op (of all members)
```

Protocol of Authentication (also see ~Authentication): The payer delivers both: triplepin and pic, by typing and handing and/or by sending the information encrypted with pubkey of the seller (to become owner of the coin). Only after the seller recognize the payer in the pic, the seller hashes the pic and uses the triplepin in Payers@My or Users@op+Profiles@op to verify by matching the hash=id.

***/

/*** ~Issuing-coins: Movements = <Coins@op[id(coin)] Wallet@My [Rand]Log@op, CoinsId@op> **

before issuing new coins in Coins@op by creating or modifying one record in Treasury@op, their StartValue and ThierAmount should be considered in distributing them to all Values=@My,Wallet@My, Log@op. When issuing new coins we should care for making no collusion of the hash and for unique random. In issuing we will add to values after grouping amount of items in groups of StartValue.

"printing" coins of respect can be done when issuing new cycle of old coin or creating new cycle as the ring creates its coins to projects its (new) Gini, by both:

A) maintaining its social obligation amounted to Mini guaranteeing minimal amount of coins per each of its members and

B) by delivering an additional equal dividend D to each of its members,

such that

-1 <= min_Injustice <= Gini - Justice <= max_Injustice <= 1 and
0 <= min_stress <= Gini / Justice <= max_stress,

where

Gini is the ratio of the areas on the Lorenz curve diagram used as a measurement for inequality in the ring, as 0<= Gini<=1,
X is the number of ring's members,
Y is the amount of ring's coins,
D is an equal Dividend per ring's member, as D*X is added to Y per each round,
A is the Average of coins per ring's member, as A=Y/X,
Mini the Minimal amount of coins guaranteed per each of the ring's members,
Justice = Mini/A, Justice as in ring's social Justice, as 0<=Justice<=1, since Mini<=A.

Notes:

The rings "printing" coins is to be done with specific coins's-lifetime, D and Mini, using limitation such as min_Injustice, max_Injustice, min_stress, or max_Stress to response to specific changes of the Justice and Gini in the ring, for to meet some policy, which are to be made automatically and/or directly under decision made by people.

Even when the rcoin are only in the ring tradable, still the tradability out of that ring is optional by peer coin, which is the value of accountability-and-ownership of one peer owner, as peer coin is measured by rcoins of other ring, money or money's equivalent.

```
****/
/*** ~Coin's-Calculus: Values=<Worth@My [same-format]Treasury@op,,>
: CoinAge, starting in zero,
  is the time in days for the lifetime of that coin, where in each day
  CoinAge is increased by one as long as CoinAge is smaller than CoinLifetime
  and where MembersCount is the number of members in the community issuing
  the coin, such that
  CoinValue             = CoinStartValue * (1- CoinAge/CoinLifetime),
  MemberDividend        = (CoinStartValue/MembersCount)*(CoinAge++/CoinLifetime)
  and CoinAge++, so that daily CoinValue  -= OwnerDailyLost, where
  OwnerDailyLost        = CoinStartValue     / CoinLifetime and
  MemberDailyEarning    = OwnerDailyLost     / MembersCount.

The functions .CoinValue running on <Worth@My ,Treasury@op> to reduce
and add value are used for to update the member Wallet values on a daily bases.

Periodically, when the accumulated MemberDividend becomes higher enough there
should be  a new issuing of such coins. Issuing of such coins should be
triggered by collecting the Dividend from all members and can be depended on
some regulations such as of big Dividend per member and/or  time period
defined by default or some decisions.
*/
/*** ~Authentication: Identities=<Self@My[pic]Payers@My, Users@op[register]Profiles@op> ***
Hashed Pic Id Authentication as a simple practice for member's
authentication (from http://namzezam.wikidot.com/blog:5):
    In initiation, The members exchange an encrypted asymmetrically pic
    (as the pic of payer is added to Payers@My of the payee) and
compressed image showing only the member as that image is indexed by its
fingerprint or by other token (e.g. triplepin), where the hash of the image
is used as the id of the member and the (12 hex-digits) hash of that id is
used as the fingerprint of that image.
    In authentication, the members are able to see each other, in physical
presence or via internet in a real-time visual and dynamic communication, and
the identifying member checks if both conditions are met:
        1) the seen member is the one being shown in the image and
        2) the hash of that image is identical to the id of the seen member.
Not as in the biometric info, per each user, the id(member)=hash(pic) can
be changed, whereas the triplepin(user) remains unchangeable as the triplepin
of self is optionally shared in transaction.
       It is used when pic is image showing the member for authentication,
       in which the members are able to see each other, in physical
       presence or via internet in a real time visual and dynamic
       communication, where the identifying member checks if both
        conditions are met:
        1) the seen member is the one being shown in the image and
        2) the hash of that image is identical to the id of the seen member.
*/
/*** ~code.. ***/
int rcoin_New(void);   /** as a Constructor */
int rcoin_Escape(void);/** as a C++ Destructor*/
int rcoin_open(void);  /**Opening db in the (encrypted) directory of the app.*/
int rcoin_sql(char *); /**Executing SQL statement*/
typedef struct coin_info_type {/** 16bytes constant values per coin*/
 int StartValue,    /** Its 10 LSB (Least significant bits) indicates the
  minor monetary unit (like cent), where the other bits indicates the major
  monetary unit (as coin), such that the number of coins represented by
  StartValue is the StartValue's major unit equal (StartValue/1024) or
  (StartValue>>10), as the StartValue's minor unit (like cent) equals
  (StartValue&0x3ff).*/
     LastDate,      /** In days elapsed since Epoch, such that
         int CoinAge = (((time_t)time(NULL)) / 86400)+Lifetime-LastDate; */
     Lifetime,      /** In days from BirthDate until LastDate, where
  BirthDate=LastDate-Lifetime, as 0=<CoinAge<Lifetime. Should be reconsidered
  for 'printing' coins parameters for effecting Gini and Justice. Note that 25
  years (or 9125 days) Lifetime is equivalent to 4% inflation and is used as
  default.*/
     Id;            /** unique&random, id of coin : unique & random of which
  sid (time, noise hahs(news) and result is unique in the db. in it there is
  no info(coin) the info encrypted in the Wallet of the owner and in another
  table able to be restored.*/
} coin_info_type;   /**...constant parameters of the coin used For The Calculus of the Coin*/
#include <\
time.h>       /** as the coin's value are time dependent.
```

```
 int daysSinceEpoch=(((time_t)time(NULL))/86400);*/
float rcoin_CoinValue(coin_info_type *);  /** */
float rcoin_MemberDividend(coin_info_type * );  /** */
typedef struct coin_calc_type{
 int LastCalculatedDate,TodayDate;// =(((time_t)time(NULL)) / 86400);
 //int Age;// CoinAge = r.calc.TodayDate+r.calc.info->Lifetime-r.calc.info->LastDate;
 //(((time_t)time(NULL)) / 86400)+Lifetime-LastDate
 int MembersAmount;
 float (*CoinValue)(coin_info_type *);
 float (*MemberDividend)(coin_info_type * );
}coin_calc_type;
/***/
typedef struct rcoin_type{ /**as a c++ class rcoin, but initialized as c file global:*/
int (*Escape)(void);        /** rcoin_Escape Destructor*/
int (*New)(void);           /** rcoin_New Constructor*/
char **man;
char member_is_operator;/**           sqlile members:*/
sqlite3 *My_db,*Op_db;       /** using only these db*/
char *Err;                  /** error msg by sqlite*/
int (*open)(void);          /** rcoin_open*/
int (*sql)(char *);         /** rcoin_sql*/
/***/
coin_calc_type calc;    /** rcoin calculus:*/
}rcoin_type;
#endif                       /** end of defined_rcoin*/
/******--------- ~InProcess: to sort out from here----------******/
/** The format of TablesOfCoins is,
             table-name    = Coins(is_mine)LastDate,
             key           = CoinStartValue,
             column-name   = CoinLifetime and
             Value         = blob of a sequence of 4 bytes int Id(coin), where
             Amount(Field) = sizeof(Field)/4;
             CoinValue     = StartValue -((StartValue * CoinAge)/Lifetime));
             CoinAge       = TodayDate  + Lifetime - LastDate;
             and where the format of their TableOfValues is
*//***[no need for Treasury in format TablesOfCoins in My_db, but instead of
 Treasury in op.db in format TableOfValues having no coins of is_mine==1. *//**???
             key           = (is_mine)LastDate
             column-name   = CoinLifetime
             Value         =  (+or-)daliychange, currentvalue,
                  where   CoinSStartValue
                  = SumAll(CoinStartValue(inTableOfValues)
                      *Amount(FieldInTableOfValues)
            as the sum in each record and then of
            all records gives one number value.


  Op_db has 3  tables
   (used in Common by any, between 1 and all, members being operators) :
Log:  (at least 2 per each coin)
          {hash(Rand)=>(id(Owner),si[Payer](Rand,id(Owner)),Chain)}
               where Chain=hash(ChianPrev,RandPrev,Id(coin))
Users:(of all members)
          {hash(triplepin) =>(register=ALL(hash(pic(member)),,))}
Profiles:(of all members){hash(register)=>(private info in common)}



  My_db has 4 + n CoinsTables  (used in Private by each member) :

     Self(of member's Pic):     {date-of-creation =>pic}
           the blobs  of pic which are/were used for id of the member.

     Authen(of authenticated members):{hash(pic)=>triplepin}.
           as in WOT used to get the info of the payer
           in Op_db by users and profiles.

     Wallet(of member's coins): {hash(id(coin))=>(Rand,RandPrev,coin)}
           used in Log@Op_db for proving ownership
           and its continuity over the coin, by the ability
           to reproduce the ChainPrev(RandPrev).

     Values(of all coins): {format of TableOfValues,
                        where is_mine=1 and is_mine=0}
           used for to evaluate coins of the member, each blob
           is of the table made in updating per day.
           The updating is to run in
             1. exchange, as member pay or paid, only for
                the coins being exchanged
             2. (TodayDate==LastDate) only Table of that Lastdate and
             3. changing the MembersAmount (all Treasury
                of which is_mine==0).
           When updating MemberDailyEarning by the coins of others,
             it is only an estimation activated on issuing new such coins,
             where the issuing could be monthly made,
             whereas the updates in the wallet are daily made.
             -= OwnerDailyLost *amount-of-such-coins
             += MemberDailyEarning*amountof suchcoins.
                table-name=OwnerDailyLost=
                      CoinStartValue/CoinLifetime: key(id),CoinValue, LastDate,
 to finish the Values so that the Coinvalue is the -change memeberdedens..
        and the const change ...
                 table-name=MemberDailyEarning:CoinValue, LastDate

     Treasury: ( of all coins,n tables in format of TablesOfCoins, has
member's coin, when is_mine==1 and when the Treasury of others, as is_mine==0,
could be made a virtual one): Do need this? community depended: default yes,
as this provides certainty.and if so should it have (id(coin),id(owner) or
only id(coin)?
  Treasury, used for updating the Wallet and are in the TablesOfCoins format.
      (is_mine)->LastDate->CoinStartValue->CoinLifetime->Id(coin)
       If the coins are of others, then is_mine=0, otherwise is_mine=1.


*//**

Definitions, where ae[member](record) is the default way to distribute a
record for the 4 db in the encrypted directly including this app the 2
Privately db per each member and 2 Commonly by member/s acting as operator/s
(at least 1 in comunity), where any access/modification is resulted in
parallel notification to all other and where
  pic           means compressed image,
  si[x](data)  means signed by x == (data,aep[x](hash(data))),
  se[x](data)  means symetrically encrypted data by the key of x,
  ae[x](data)  means asymetrically encrypted data by the public key  of x.
  aep[x](data) means asymetrically encrypted data by the private key of x:
---se[member]PrivatelyCoins-db ---(of any coin in Treasury and in others of
members'coin)
1.Treasury: {id(coin)=>coin}  <-of any-other, for adding value
by.MemberDividend)
2.Treasur:  {id(coin)=>coin}  <-of the-member, for reducing value
by .CoinValue)
3.Wallet:   {CoinValue=>coin} <-of the-member, daily updated)
---se[member]PrivatelyWho-db ---(of self and her/his customers)
1.Self:     {id(pic)=> pic}  <-of the-member,id(pic) as in register, id(user)==hash(pic) <- Not as in the biometric info, per each user, the id(user)=hash(pic) can be changed, whereas the triplepin(user) remains unchangeable as the triplepin of self is shared in trunsaction)
2. Authen:  {hash(pic)=>12digittriplepin} <-of those who pay to the member )
----se[member]CommonlyCoin-db: (of-any coin)----
1.Proves:        ae[member]{hash(id(coin))=>se[Owner](receipt[n])} of all coins<-)
2. Log:  ae[member]{hash(receipt[n])=>id(Payer),si[Payer](id(Owner))} <-)
----se[member]CommonlyWho-db: (of-any member)----
1.Users:       ae[member]{hash(triplepin) =>(register=ALL(hash(pic),,))}<- order as in Self)
2.  Profiles:     ae[member]{hash(register)=>(private info in common)}
=================
???1.Users:         ae[member]{hash(id(member))=>triplepin,ae[Owner](rand[n],hash(rand[n-1]))}
????2. Profiles:      ae[member]{{hash(rand[n])=>info(user, patrly ae[user])}
=====================
--ae[member]PrivatelyCoins-db----
1.Treasury: {id(coin)=>coin}  <-of any-other, for adding value by.MemberDividend)
2.Treasur:  {id(coin)=>coin}  <-of the-member, for reducing value by .CoinValue)
3.Wallet:   {id(coin)=>coin}.<-of the-member, daily updated)
---se[member]PrivatelyWho-db ---
1.MemberPic: table for some pic of self sorted by id(pic)
    for that each user can have meny id(user),
    where all id(user) are able to identified by payee,
    assuming that the ownership defined by the user id in the log is a valued property of the payer and as the triplepin is still unique
```

```
2.Payers:    ae[member]{hash(12digittriplepin)=>compressed-pic},
    of self and her/his customers (potentially of all users),
    and as each user can transfer her/his data in the transaction,
    demand deleting it after identification and can change the pic added to profile,
    where each user has different triplepin and id(user)==hash(pic,triplepin,id(pic)) and
    where the pic is used by the payee to identify the payer in the transaction for achieving unique authentication.

--ae[member]CommonlyCoins-db---
1.Proves: ae[member]{hash(id(coin))=>ae[owner](receipt[n])},
    of all coins and as only receipt of last transaction is kept,
    where receipt[n] = (random[n] signed by Owner,hash(receipt[n-1]))
    and for to prevent collision in creation of new coins, as the hash(id(coin)) is sent to all users, each user cheak if exist, so when exists reahsing the hash.
2.   Log:      ae[member]{{hash(receipt[n])=>id(Payer),si[payer](id(Owner))}
--ae[member]CommonlyWho-db-----
3.Users:   ae[member]{hash(id(member))=>triplepin,ae[Owner](rand[n],hash(rand[n-1]))}
4.   Profiles:   ae[member]{{hash(rand[n])=>info(user, patrly ae[user])}
*/
```