OpenBarter

openBarter is a database defining a barter market. It implements the economic mechanisms of a regular market and allows exchange between more than two partners (the buyer and the seller) in a single transaction. It does not require any central monetary standard to quantify the value exchanged and to express prices to be compared.
The low liquidity of barter is the historical reason why money expended while barter decreased. By allowing exchanges with more than two partners the barter market becomes nearly as liquid as a regular market using money. The competition between groups of more than two partners fosters cooperation inside groups, as well as economic performance.

Clients of the database are external organisations called depositories. A depository delegates to the database the ownership management of values. It sends to the database orders of owners: exchange proposals called bids, or movements of values from and to the database on which these bids are based on. The database finds possible relations between bids and proposes draft agreements for approval by owners. A depository is accountable of the relation between the physical value and the information that describe it.

1 Economics

We propose to define a bid as a statement of an owner to provide a value defined by a couple (quantity,quality) in exchange of an other quality for a given $\boldsymbol{\omega}$ ratio between provided and received qualities.

1.1 Draft agreement formation

From bids submitted by depositories, the database finds possible relations between them. This search is done each time a new bid is submitted, and consists in the following steps:

A) Find sets of bids where matchings between quality provided and required form an exchange cycle,

B) Select the best among these cycles, using a best price rule generalized to multilateral agreements,

C) Finds a compromise between the bids of the selected cycle, and records the resulting agreement as a draft to be submitted to owners.

1.2 Multilateral exchange

Let's consider a set of $\boldsymbol{n}$ such bids where the quality offered by one equals the quality required by the other. $\boldsymbol{\omega_i}$ are prices of these $\boldsymbol{n}$ bids, with $i \in [0, n-1]$.

Let's also suppose an agreement exists between partners, and that each partner provides a given quantity $\boldsymbol{q_i}$ to an other partner. We should have:

$$\omega_i = \frac{q_i}{q_{i-1}} \; for\; i \in [1, n-1]$$

$$\omega_0 = \frac{q_0}{q_{n-1}}$$

If we make the product of those ***n*** expression,

we obtain: $$\prod_{i=0}^{n-1} \omega_i = \frac{q_0}{q_{n-1}} * \prod_{j=1}^{n-1} \frac{q_i}{q_{i-1}} = 1$$

We see that if an agreement exists, we have:

[1] $$\prod_{i=0}^{n-1} \omega_i = 1$$

1.3 Bilateral case

Using the regular definition of price for an exchange between a seller providing 10 Kg of apples in exchange of 20 pounds, the price is 2 pounds/Kg.

If a seller provides a quantity ***g*** of goods to a buyer in exchange of a quantity ***m*** of money, the regular definition of price is: $p = \frac{m}{g}$.

Buyer and seller have usually different ideas of prices, We note the buyer price ***p_b*** and the seller price ***p_s***. Agreement is the result of a compromise between these prices.

Using the definition of price ***ω*** given earlier (a ratio between provided and received quantities) the buyer price is ***ω_b*** such as $\omega_b = \frac{m}{g}$ and the seller price is ***ω_s*** such as $\omega_s = \frac{g}{m}$.

We see that $\omega_b = p_b$ and $\omega_s = \frac{1}{p_s}$

Agreement

Agreement on price between the buyer and the seller exists when $p_b = p_s$. An equivalent statement using expressions ***ω_b*** and ***ω_s*** of price is:

$$\omega_b = \frac{1}{\omega_s} \Rightarrow \omega_b * \omega_s = 1$$

It is the definition of agreement given earlier [1] applied to the bilateral case.

Competition

Using the traditional definition of price, the best price rule is for the buyer the minimum ***p_s***, and for the seller the maximum ***p_b***.

Using the new definition of price; since $\omega_s = \frac{1}{p_s}$ and $\omega_b = p_b$, it is the maximum ***ω_s*** and the maximum ***ω_b***, the best price rule is the selection of the best:

[2] $$\omega_s * \omega_b$$

Compromise

Barter is required when buyer and seller do not agree on their prices ***p_b*** and ***p_s***. We propose to define it as the geometric mean of ***p_b*** and ***p_s***: $p' = \sqrt{p_b * p_s}$. This represents for each partner the same ratio between initial and final price. It is also a simple way to balance offers of partners.

Expressed using expressions ω of prices, we obtain new values ω'_s and ω'_b such as their product is 1:

$$\omega_s' = \frac{\omega_s}{\sqrt[2]{\omega_s * \omega_b}}$$

$$\omega_b' = \frac{\omega_b}{\sqrt[2]{\omega_s * \omega_b}}$$

or more simply:

[3] $$\omega' = \frac{\omega}{\sqrt[2]{\omega_s * \omega_b}}$$

1.4 Extension to the multilateral case

We note the product $\Omega = \prod_{i=0}^{n-1} \omega_i$.

The best price rule [2] is extended to the multilateral case by choosing the cycle of bids maximizing:

[4] $$\Omega$$

Likewise for [3], a compromise can be obtained for a cycle of bids with:

[5] $$\omega' = \frac{\omega}{\sqrt[n]{\Omega}}$$

[4] is the simplest expression of [2] to the multilateral case . However it is *not* the unique solution.

2 Model

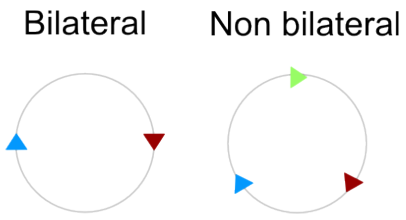
We describe here objects and functions of the model.

2.1 Value

A *value* is a couple (*quantity,quality*), where *quality* is a name, and *quantity* is an integer. It can be used to define an amount of mineral, of pollution, or of money. The *owner* of such a value can use the market to exchange it for an other quality.

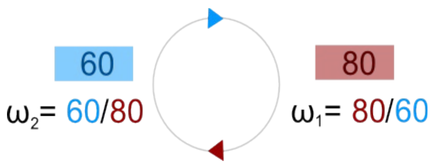
2.2 Exchange

This market allows exchanges between two partners or more in a single transaction. An exchange forms a cycle of partners where each provides a value and receives an other value of different quality. An exchange with more than two partners is called *non-bilateral*.



2.3 Price

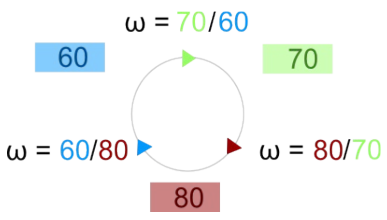
The *price* is defined for a couple (quality provided,quality required). It is the ratio ω between provided quantity and received quantity. It measures how much we accept to provide of the



quality provided when we receive a unit of the quality required.
On a regular market, the expression of price is the same for the buyer and the seller and are equal when they agree on the price. In this model, the price of parters are different even when they agree on their prices.

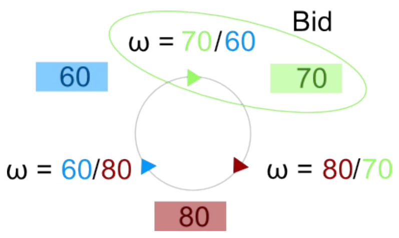
2.4 Agreement on price

When a set of bids form an exchange cycle, a draft agreement can be formed when there is an agreement on prices. It occurs when the product of prices ω of bids of the exchange cycle equals to ***1***. When this condition is not met a barter is required. The agreement is obtained on the example.



2.5 Bid

It is an unilateral commitment of the owner to exchange a value he owns for another quality at a given price. A bid is defined by a value provided, the quality of the value required, and a price ω. Two bids match when the quality provided by one is the quality required by the other. When some bid matchings form a cycle, an exchange can be formed from this cycle.



2.6 Best price rule

It is used to select the best among possible exchange cycles. The best cycle is the one having the maximum product of prices ω.

2.7 Barter

When there is no agreement on price, an automatic barter is performed where prices ω are divided by the geometric mean of ω of the cycle.

$$\omega'_j = \frac{\omega_j}{\sqrt[n]{\prod_{i=1}^{n} \omega_i}}$$

This division converts ω to ω' in such a way that the product of ω' equals to ***1***.

Draft agreement creation also needs adjustments on quantities that need to be integers. The agreement on price is used to obtain quantities q_i' by computing the maximum flow of value along the cycle of bids using ω' and the quantity provided by each bid. This maximum flow is a tuple of real numbers q_i' rounded to integers Q_i'. We consider the vector $\vec{q}$ of coordinates q_i', and $\vec{Q}$ of coordinates Q_i', and minimize the rounding error represented by the angle ε between the two vectors. It is done by maximizing:

$$\cos(\alpha) = \frac{\vec{q} * \vec{Q}}{\|\vec{q}\| * \|\vec{Q}\|}$$

However, the solutions $\vec{Q}$ such as some Q_i' are null should be excluded, because a draft agreement where some partners don't provide anything would be unfair.

3 Implementation

The market is seen as a directed graph where bids define nodes, and matchings between bids define arrows. This graph is maintained acyclic by transforming cycles into draft agreements as soon as they appear. This occurs each time a bid is added. A competition occurs between possible cycles when more than one cycles are found. Draft agreements are produced from these cycles. Values corresponding to these agreements reduce values of bids in such a way that cycles disappear.

Due to limits of computational resources on time and memory, limits are defined on the graph traversal that should be tuned.

3.1 Technical overview

openBarter is an extension of postgreSql. Stored procedures act on a model representing qualities, owners, values, and draft exchange agreements.

The main time consuming primitives of the server are:

- read the best price for a couple of qualities requested and provided,
- make an bid.

A single simplifying hypothesis is used to limit computation load: the number of partners of contracts is limited to ***openbarter.maxarrow***. This is why the liquidity of this barter market is not strictly the same as on a bilateral stock market.

3.2 Limits

openbarter.cachesize (memory size >= 1M, default 4M)

Limits memory used by berkeleydb. The memory is allocated on the server for each backend process. The number of mega should be ***openbarter.maxarrow/395*** when page size is 8182 bytes.

openbarter.maxarrow (int >= 128, default 1580)

When bid relations are explored, a graph is produced. It is the maximum number of arrows of this graph.

openbarter.maxcommit (int in [3,8], default 8)

Limits the number of commits of agreements. Consequently, it also limits the number of partners. This value should not be modified after the database is created.

The following is added at the end of postgresql.conf:

```
custom_variable_classes = 'openbarter' # list of custom variable class

#------------------------------------------------------------------------

# OPENBARTER

#------------------------------------------------------------------------

openbarter.cachesize = 4MB

openbarter.maxarrow = 1580                # 395*4
openbarter.maxcommit = 8
```

3.3 Users

Users of openBarter are clients of the database. It uses ssl authentication capabilities of postgreSql to enforce security of openBarter. Refer to postgreSql documentation to set up a client and server ssl certificate. Roles are the followings:

- depository, external institution acting on behalf of owners, It can move values from and to the database only for qualities he owns, make bids on them, and accept or refuse drafts using these qualities.
- admin, executes actions of registration and provides accounting reports.
- market, executes all other necessary tasks.

3.4 Database model

The database is described by pg/openbarter.sql. It consists in two schemas: ***ob*** is the internal state of the market, and ***market*** is the public representation of the market.

3.4.1 Vocabulary

depositary	user of the database with the role depositary, for example ***depos_com***
quality	The name of a quality, for example ***depos_com>sand***. This quality is a value standard precisely defined by the depositary ***depos_com***. This quality is owned by a single depository.
owner	The owner of a value, for example ***luc@depos_com***
value	It is a tuple (quality, quantity,owner) for example (***depos_com>sand***,***124***,***luc@depos_com***). It means the ownership management of this value has been delegated by the depository to the market.
account	is a position at a given time, with history, for a given (owner,quality)

3.4.2 Schema ob

Tables	*Description*
ob.tdraft,ob.tcommit	Description of draft agreements. An agreement is a ob.tdraft row. A ob.tcommit row refers to it and describes the commitment of a single owner for this agreement. ***openbarter.maxcommit*** is the maximum number of commits related to a given draft.
ob.tomega, ob.tlomega	History of prices. A couple (quality provided,quality required) refers to a single row of ob.tomega, while rows of ob.tlomega refer to a row of ob.tomega.

ob.tmvt	History of movements on ob.tstock, when the ownership of values are modified.
ob.tnoeud	Description of bids, refers to a quality provided, and a value in ob.tstock
ob.towner	Description of owners
ob.tquality	Description of qualities
ob.tstock	Description of values.
ob.tuser	Description of depositories

A row of the table ob.tstock describes a value – a tuple (quantity, quality, owner of the value). The quality of this value is owned by a depository, ownership management is delegated by the depository owner refers to a ob.tquality, an ob.towner, and has a quantity, a version, and a type that can be A,D or S.

It is a type A when the value is moved to the database. It is a stock_A.

It is a type S when the value is referred by a bid. It is a stock_S.

It is a type D if the value is referred by a draft agreement. It is a stock_D.

3.5 Application programming interface

Clients act through stored procedures that must be integrated in transactions by the client.

A depository can just create a quality, move values of these qualities in and out of the database, and act on behalf of owners of these values to create/remove bids and accept/refuse drafts including these qualities.

Other procedures are used by roles market or admin.

A quality is created and used by a single depository in order to separate responsibility of ownership management of each quality.

Even if value ownerships are exchanged, the depository that guaranties these values remains unchanged.

Function	
market.fcreate_quality	records a new quality
market.fadd_account	moves a value to owner's account
market.fsub_account	moves a value from owner's account
market.finsert_bid	inserts a bid
market.finsert_sbid	insert a bid based on an other
market.fdelete_bid	removes a bid
market.faccept_draft	accepts a draft

market.frefuse_draft	refuse a draft
market.fbatch_omega	
market.fstats	gives global stats
market.vowned	Gives quantity owned for each couple (quality,owner).
market.bvalance	List of values owned by user group by quality name.
market.vdraft	List of drafts where the owner is partner.
market.vbid	List of bids
market.vmvt	List of movements
market.fcrt_start_renewal	Start certificate renewal
market.fcrt_read_renewal	
market.fcrt_accept_renewal	Accept certificate renewal
market.fcrt_abort_renewal	Abort certificate renewal
market.fcrt_getuser	

In case of error, an exception is raised, with the code "38000", with comments about the error. In the following, int is used form 32 bit integer, and int8 for 64 bits integer.

3.5.1 market.fcreate_quality

```
int market.fcreate_quality(name text);
```

records a new quality with the name: ***user>name*** where ***user*** is the name of the client of the database. Requires the role depositary.

Returns:

- 0 when the quality is created,
- <0 on error.

example:

```
select * from market.fcreate_quality('sand');
```

If the name of the depository defining the quality is ***depos_com,*** this creates a quality ***depos_com>sand***

3.5.2 market.fadd_account

```
int market.fadd_account(_owner text,_quality text,_qtt int8);
```

conditions :

- ***quality*** exist,

- ***qtt*** >=0.

moves the value to owner's account defined by a couple ***[owner,quality]***

account and owner are created when they do not exist. The movement is recorded. Requires the role depositary.

Returns:

- 0 when the account is credited,
- <0 on error.

3.5.3 market.fsub_account

```
int market.fsub_account(_owner text,_quality text,_qtt int8);
```

conditions :

- o***wner*** and ***quality*** exist,
- ***qtt*** >=0 and <= ***qtt*** of the account(***_owner,_quality)***

Moves the value to owner's account defined by the couple ***(owner,quality)***

Account is deleted when empty. The movement is recorded. Requires the role depositary.

Returns:

- 0 when the account is debited,
- <0 on error.

3.5.4 market.finsert_bid

```
int8 market.finsert_bid(
        _owner text,
        _qualityprovided text,
        _qttprovided int8,
        _qttrequired int8,
        _qualityrequired text
)
```

inserts a new bid based on a new stock_D. Requires the role depositary.

conditions :

- ***owner*** exists,
- ***qualityprovided*** and ***qualityrequired*** are defined,
- ***qttprovided*** >0,
- ***qttrequired*** >0.

Returns:

- >=0 the number of drafts created,

- <0 on error.

3.5.5 market.finsert_sbid

```
int8  market.finsert_sbid(
        bid_id int8,
        qttprovided int8,
        qttrequired int8,
        qualityrequired text
)
```

Inserts a bid based on an other bid. The value proposed by this new bid is the same as the one referred by the bid ***bid_id***. Requires the role depositary.
conditions :

- ***bid_id*** exists,
- ***qttprovided*** >0,
- ***qttrequired*** >0,
- ***qualityrequired*** is defined.

Returns an int8:

- the number of drafts created (>=0),
- < 0 error.

3.5.6 market.fdelete_bid

```
market.fdelete_bid(bid_id int8)
```

conditions :

- the bid exists

Delete bid and related drafts. Requires the role depositary.

Delete related stock_S if it is not related to an other bid.

The quantity of this stock_S is moved back to the account.

A given stock_S is deleted by the market.fdelete_bid() of the last bid it references.

3.5.7 market.faccept_draft

```
int market.faccept_draft(draft_id int8,owner text)
```

conditions :

- draft_id exists with status D

returns:

- 0 the draft is not yet accepted by all partners,
- 1 the draft is executed,
- < 0 error.

3.5.8 market.frefuse_draft

```
int market.frefuse_draft(draft_id int8,owner text)
```

conditions :

- draft_id exists with status D

the draft D is cancelled. Requires the role depositary.

Values of stock_D booked for this draft are moved back to stock_S of bids.

returns:

- 1 the draft is cancelled,
- < 0 error.

3.5.9 market.fbatch_omega

```
int market.fbatch_omega()
```

utility calling market.fread_omega(nr,nf) for the couple (nr,nf) that needs refresh the most:

- a couple (nr,nf) such as market.tomega[nr,nf] does not exist,
- if not found, oldest couple (cflags&1=0),
- if not found, oldest couple such as (cflags&1=1)

Should be called by a cron.

3.5.10 market.fstats

```
market.tret_stats market.fstats()
```

gives general informations about the model:

Column	Type	Meaning
mean_time_drafts	int8	mean of delay of drafts

nb_drafts	int8	number of drafts
nb_noeuds	int8	number of bids
nb_stocks	int8	number of stocks
nb_stocks_s	int8	number of stocks type=S
nb_stocks_d	int8	number of stocks type=D
nb_stocks_a	int8	number of stocks type=A
nb_qualities	int8	number of qualities
nb_owners	int8	number of owners

all following columns should be zero

Column	**Type**	**Meaning**
unbananced_qualities	int8	number of qualities with accounting problems
corrupted_draft	int8	number of inconsistent drafts
corrupted_stock_s	int8	number of stocks_S not related to a bid
corrupted_stock_a	int8	number of couples (quality,owner) where stocks_A is not unique

Example:

```
select * from market.fstats()
```

3.5.11 market.vowned

Gives quantity owned for each couple (quality,owner).

Column	Type	Meaning
qown	text	owner of the quality
qname	text	quality name
owner	text	owners name
qtt	int8	sum(qtt) for couples (quality,owner)
created	timestamp	min(created)
updated	timestamp	max(updated?updated:created)

examples

```
SELECT * FROM market.vowned WHERE owner='jack@depos_com';
```

total values owned by the owner '***jack@depos_com***'

```
SELECT * FROM market.vowned WHERE qown='depos_com';
```

total values of owners for qualities of ***'depos_com'***

```
SELECT o.qname FROM market.vowned o
        INNER JOIN market.tquality on q.name=o.qname
        GROUP BY o.qname WHERE q.qtt != 0;
```

returns 0 lines when accounting is correct   for each quality

3.5.12 market.bvalance

List of values owned by user group by quality name.

Column	Type	Meaning
qown	text	owner of the quality
qname	text	quality name
qtt	int8	sum(qtt) for this quality
created	timestamp	min(created)
updated	timestamp	max(updated?updated:created)

examples:

```
SELECT * FROM market.vbalance WHERE qown='depos_com';
```

Total values owned by the user '***depos_com***'

```
SELECT count(*) FROM market.vbalance WHERE qtt!=0 AND qown='depos_com'
```

Returns 0 if accounting is correct for this user.

3.5.13 market.vdraft

list of drafts by owner.

Column	Type	Meaning
did	int8	id of draft
status	char	always D
owner	text	owner providing the value
cntcommit	int	number of commits of the draft
flags	int4	bit 0 set when accepted by owner; bit 1 set when refuse by owner
created	timestamp	

usage:

```
SELECT * FROM market.vdraft WHERE owner='paul@depos_com'
```

list of drafts for the owner '***paul@depos_com***'

```
SELECT owner,flags&1 as accepted,flags&2 as refused FROM market.vdraft
WHERE did=100
```

list of partners of the draft 100 with their decisions.

3.5.14 market.vbid

List of bids.

Column	Type	Meaning
id	int8	id of bid
owner	text	name of owner
required_quality	text	
required quantity	int8	
omega	float	the ration provided_quantity/required quantity
provided quality	text	

provided_quantity	int8	
sid	int8	stock id of the bid
qtt	int8	sum(qtt)
created	timestamp	

usage:

```
SELECT * FROM market.vbid WHERE owner='luc@depos_com'
```

list of bids of the owner '***luc***'

3.5.15 market.vmvt

returns a list of movements related to the owner.

Column	Type	Meaning
id	int8	id of the movement.
did	int8	When an agreement is executed, all movements produced have the same did. It is not NULL for a draft executed even if this draft where deleted. It is NULL when the movement is not due to the execution of an agreement.
provider	text	name of provider
nat	text	quality of moved value
qtt	int8	quantity moved value
receiver	text	name of receiver
created	text	timestamp

usage:

```
SELECT * from market.vmvt where 'luc@depos_com' in (provider,receiver)
```

list of movements for this owner.

/* subscription process

admin runs:

```
SELECT ob.fadduser(dn,10000);
```

the user connects.

3.5.16 Authentication of client

It uses potgreSql ssl authentication with a certificate on the server and client side. These certificates have a limited live time.

To use the service of openBarter, a client need to be registered, and at to renew it's certificate when necessary.

3.5.16.1 certificate registration

Registration is required to use any function. We suppose here the ***commonName*** of the client certificate is ***depos_com***, and that no client is registered with this name.

The administrator executes:

```
SELECT ob.fcrt_adduser('depos_com',10000);
```

The first client connecting with the commonName ***depos_com*** is then registered, and the dnIssuer and serial number are associated with this commonName. All future use of this commonName will verify dnIssuer and serial number match with the recorded informations.

3.5.16.2 certificate renewal process

When the certificate is too old, the user looks for a new certificate. The process is defined in such a way that the client remains the owner of the commonName and associated qualities, even in case a malicious actor would attempt to steal this ownership. The process does not need administrator's action. The client just needs the old and new certificate.

with the old certificate:

```
        SELECT market.fcrt_start_renewal();
```

with the new certificate:

```
        SELECT market.fcrt_getuser(2);
```

with the old certificate:

```
        SELECT market.fcrt_read_renewal();
```

returns a value dnIssuer:serialNumer of the new certificate The client compares this value to that of the new certificate. If the value is the same:

```
        SELECT market.fcrt_accept_renewal(dnIssuer:serialNumer)
```

At any time between fcrt_start_renewal and fcrt_accept_renewal,

```
        SELECT market.fcrt_accept_renewal();
```

abort the renewal process.

3.6 Installation

3.6.1 Build from sources

Following instructions apply to a linux 32 bits achitecture.

3.6.1.1 Build Berkeleydb

On most linux distributions, the library is already installed. If it is not, upload berkeleydb db-4.8.30 to a directory,say 'sw', and uncompress sources:

```
>> cd sw/db-4.8.30

>> mkdir build_unix

>> cd build_unix
```

Choose directories where you will install include files, libraries and executable.

We suppose here you choosed ***/usr/include /usr/lib*** and ***/usr/bin*** respectively. In the directory ***sw/db-4.8.30/build_unix*** execute:

```
>> ../dist/configure --enable-smallbuild --disable-shared --libdir=/usr/lib
--includedir=/usr/include --bindir=/usr/bin

>> make

>> sudo make install
```

This will install the following files:

- /usr/include/db.h and db_cxx.h
- /usr/lib/libdb.a and libdb-4.8.a
- /usr/bin/db_* (11 files)

With these files, we can compile a C program with the command:

```
>> gcc program_to_compile -ldb-4.8 -lpthread
```

3.6.1.2 Build Postgres

Download postgres v9.0.4 . Copy the sources to ***sw/postgresql-9.0.4***:

```
>> cd sw/postgresql-9.0.4

>> ./configure --prefix=/usr --with-openssl

>> make
```

Refer to the postgreSql manual for more information.

3.6.2 Installation

3.6.2.1 Database

```
su root
gmake install
adduser postgres
mkdir /usr/local/pgsql/data
chown postgres:postgres /usr/local/pgsql/data
```

add PATH and PGDATA to /etc/bash.bashrc :

```
PATH=”/usr/bin:$PATH”
PGDATA=”/usr/local/pgsql/data”
export PGDATA
```

Create the database:

```
su - postgres
/usr/bin/initdb -D $PGDATA
```

start the server:

```
/usr/bin/pg_ctl -D $PGDATA -l logfile
exit
```

Verify the server is running:

```
/usr/bin/createdb test
/usr/bin/psql test
test=#SHOW ALL;
test=#\q
```

3.6.2.2 openbarter

In the directory contrib of postgres sources, Uncompress openbarter and move it to the directory contrib of postgres sources,

```
>> mv openbarter sw/postgresql-9.0.4/contrib

>> cd sw/postgresql-9.0.4/contrib/openbarter/pg

>> make

>> make install
```

In postgresql.conf, change the followings:

```
shared_preload_libraries = 'openbarter'          # (change requires restart)

custom_variable_classes = 'openbarter' # list of custom variable class

#------------------------------------------------------------------------

# OPENBARTER

#------------------------------------------------------------------------

openbarter.cachesize = 4MB

openbarter.maxarrow = 1580                # 4* 395

openbarter.maxcommit = 8
```

3.6.2.3 ssl_info

Install the module sslinfo of postgres

```
cd sw/postgresql-9.0.4/contrib/sslinfo

make

make install

pg_ctl restart

psql marketdb < sslinfo.sql
```

3.6.3 tests

Tests must be run with the user postgres.

```
>> cd sw/postgresql-9.0.4/contrib/openbarter/pg
>> make installcheck
```

These tests should provide positive results.

3.6.4 Production site

Except the super user postgres that was used to install the model, all users of postgres are authentified using ssl protocol. Certificates signed by the server are installed on the server and client side.

3.6.4.1 Set a certificate authority

The following command gives the place where openssl.cnf is installed on your system:

```
>> openssl version -d
```

This file contains the directory where the certificate authority keeps it's files. Change dir to ./ca using root access like this:

```
####################################################################
[ CA_default ]
#dir            = ./demoCA          # Where everything is kept
dir             = ./ca              # Where everything is kept
```

Then:

3.7 Releases

3.7.1 0.1.0

First release. Tests units are functional.