

# Protocol

openBarter allows exchange of the ownership of measurable values stored in depositories. The ownership of values move between depositories, but not the values themselves. Each depository defines it's own quality standards for the values it stores, and measure them using it.

The protocol occurs between a depository and the market place server, called here the server.

Connexion settlement is initiated by the depository. When established and the identity of the depository defined for the connexion, commands are sent by the depository, and responses are sent back to the depository. The server can also send information, called Notification, when an event need to be sent to the depository.

## **Definitions**

A name <depository> defines an institution bookkeeping the reality of values. This institution records information about values; a couple (quality,quantity) and the associated identity of the owner. The protocol only modifies the ownership of values. A depository can be a central bank when value is a money.

A name <depository>@<quality> is a quality standard used by this depository to measure the values it stores. <depository> can be omitted in the context of a depository.

An integer <quantity> defines the quantity of a value.

A name <owner> describe the name of an owner. This name is unique and can be used for any depository.

The couple (<quantity>,<depository>@<quality>) defines a value. When the depository declares to an owner he stores a given quantity of that value, the depository guaranties to the owners this value is available for immediate withdrawal.

A bid is an offer made by an owner on a market place to provide a value he owns in exchange of an other quality.

Omega is a floating point number given by an owner to measure the price of it's bid. It is the ratio between provided and received quantities.

## **Commands and responses**

A command is sent by a depository to the server. The response is sent back to the repository. Commands are presented in the following text by a first line, and responses follow immediately.

A bid is done by delegating the management of the ownership of the value to the server.

The PUT command delegates the ownership management of a value to the server:

```
PUT <quality> <quantity> <owner>  
<valueId>
```

This means that a value <quality>@<depository>, <quantity> owned by <owner> is delegated. <depository> is omitted since it is always the name of the initiator of the connexion.

The response is an Id given by the server to this value.

The BID command is sent by the depository to execute an order of an owner of a value <valueId> provided by a PUT command. This owner offers to provide this value and requires the quality <depository>@<quality> for the price <omega>.

```
BID <valueId> <depository>@<quality> <omega>
<bidId>
```

Such bids stored by the server form a graph. When a bid creates one or more cycles, they are converted into drafts to maintain the graph acyclic.

The depository can obtain the list of drafts containing values he PUT by the following command LISTDRAFTS:

```
LISTDRAFTS
<draftId1> <valueId11> <depository11>@<quality11> <quantity11> <owner11>
<draftId1> <valueId12> <depository12>@<quality12> <quantity12> <owner12>
.....
<draftId1> <valueId1k> <depository1k>@<quality1k> <quantity1k> <owner1k>
<draftId2> <valueId21> <depository21>@<quality21> <quantity21> <owner21>
<draftId2> <valueId22> <depository22>...
.....
<draftIdn> <valueIdn1> <depositoryn1>@<qualityn1> <quantityn1> <ownern1>
<draftIdn> <valueIdn2> <depositoryn2>@<qualityn2> <quantityn2> <ownern2>
.....
<draftIdn> <valueIdnp> <depositorynp>@<qualitynp> <quantitynp> <ownernkp>
```

Each line of the response represents an owner providing a value in a draft:

- <draftId> is the Id of the draft,
- <valueId> is the Id of the value,
- <depositoryn1>@<quality> <quantity> is the value exchanged,
- <owner> is the new owner, and a participant of the draft.

Lines of the same draftId are listed in the order of the exchange cycle; the owner of a given line receiving the value provided by the previous line, except the owner of the first line receiving the value provided by the last line.

A participant (owner participating to the draft contract) can accept or refuse a draft.

The ACCEPT command is issued by the depository when a partner signs a draft. The response provides the number of participants that did not sign yet for this draft.

```
ACCEPT <draftId> <owner>
<nbNotAccepted>
```

The REFUSE command is issued by the depository when a partner refuses a draft. The response provides the number of participants that already signed the draft.

```
REFUSE <draftId> <owner>
<nbAccepted>
```

The LISTVALUES command is sent by a depository to read the list of values it manages.

## LISTVALUES

```
<valueId1> <quality1> <quantity1> <owner1>  
...  
<valueIdn> <qualityn> <quantityn> <ownerIdn>
```

<valueId> is the Id of the value,  
<quality> is defined by the depository sending the command,  
and could be written <depository>@<quality>,  
<quantity> is the quantity of the value,  
<owner> is the owner of the value.

The GET command takes back the value to the depository.

```
GET <valueId>  
<quality> <quantity> <owner>
```

An error occurs when the value is not managed by the depository sending the command.

## **Notifications**

A notification is initiated by the server when an unpredictable event occurred on the server side. A notification is sent when the connexion is settled with the server.

It can be:

A syntax error of the command sent,

An internal error of the server that cancelled current command,

When a draft has been accepted by all partners, the notification is sent to the depositories of participants of the signed draft.

## **Copying Information**

This document is published with the licence GPL V2.  
<http://fsf.org/>