# CATP: A Communication Protocol for CAT Games

Version 2.02
Issued: September 12, 2010

Jinzhong Niu, Albert Mmoloke, Peter McBurney and Simon Parsons

# Contents

# List of Figures

# 1 Introduction

The *Trading Agent Competition (TAC) Market Design Tournament*, also known as the *CAT Tournament*, supports multiple markets, each regulated by a *specialist*, that run in parallel and compete against each other to attract traders and make profit. Each entrant of the CAT Tournament runs a specialist, and sets the market rules so as to win the game according to a certain set of assessment criteria. The game between specialists is also commonly called a CAT *game*.

This document presents the specification of the communication protocol for interactions between a CAT server and CAT clients — either specialists or traders — in CAT games. The protocol is denoted as the CAT Protocol, or in short CATP. The open-source JCAT project [7] provides the software platform for CAT games and an API in Java for entrants to build their market mechanisms. The API abstracts away the low-level communication details illustrated in this document, so one does not have to read this document unless one wants to build a CAT client from scratch or in a programming language other than Java.

This document has been developed as a result of inputs and discussions from teams at the Graduate School and University Center, and Brooklyn College, the City University of New York (Kai Cai, Jinzhong Niu, Simon Parsons, Elizabeth Sklar), the University of Liverpool (Peter McBurney), the University of Southampton (Enrico Gerding, Albert Mmoloke), and the University of Essex (Steve Phelps, formerly at the University of Liverpool). The annual CAT Market Design Tournament is run under the auspices of the Trading Agent Competition (TAC), and sponsored by the UK EPSRC Project, *"Market Based Control of Complex Computational Systems"* (GR/T19742/01), a research project undertaken jointly by the Universities of Birmingham, Liverpool and Southampton, UK. Further information is available from the project web-site:

`http://www.marketbasedcontrol.com/cat/`.

The JCAT software and related documents, including the latest version of this document, are hosted at:

`http://jcat.sourceforge.net/`.

This document should be cited as [9].

# 2 Change Log

- Version 1.00 : first version by Jinzhong Niu in July, 2006.
- Version 1.01 : (Jinzhong Niu, 08/02/2006) additional diagrams by Albert Mmoloke showing possible interactions during a CAT game.
- Version 1.02 : (Jinzhong Niu, 08/06/2006) typos and inconsistencies fixed.
- Version 1.03 : (Jinzhong Niu, 08/28/2006) `HELLO` and `MATCH` messages renamed to `CHECKIN` and `TRANSACTION`; `SUBSCRIBE` message added.
- Version 1.04 : (Jinzhong Niu, 09/01/2006) `POST` messages used to notify of successful shouts and transactions instead of `ASK`, `BID`, and `TRANSACTION` (see Sections 8.5, 8.7, and 8.9).
- Version 1.05 : (Jinzhong Niu, 09/18/2006) typos fixed.
- Version 1.06 : (Simon Parsons, 10/05/2006) typographical fixes, some additional explanation added.
- Version 1.07 : (Jinzhong Niu, 11/07/2006)
  - Section 3.2 reorganized and updated;
  - Section 3.3.1 updated;
  - `Version` header added and protocol parameter in `CHECKIN` messages replaced with a `Version` header;
  - Section 8.2 updated with additional diagrams, and new game and day initialization;
  - Sections 8.3, 8.4, 8.5, and 8.6 updated with additional diagrams;
  - Sections 8.7, 8.8, and 8.9 updated reflecting changes on `ASK`, `BID`, and `TRANSACTION` processing;
  - `Tag` header added to allow detection of obsolete messages due to network delay; and
  - The section on possible dialogs in different scenarios added in Version 1.01 removed; diagrams are now distribute in other related sections.
- Version 1.08 : (Jinzhong Niu, 11/20/2006) list of figures added after table of contents.

- Version 1.09 : (Jinzhong Niu, 11/22/2006) notification of profits added at the end of each trading day through `OPTIONS DAYCLOSED` messages; `ROUNDOPENING` changed to `ROUNDOPENED` to be consistent with JCAT code.
- Version 1.10 : (Jinzhong Niu, 12/04/2006) `POST PROFIT` added to notify of specialist' profits at the end of each trading day (see Section 8.5); daily trader distribution announcement added at the end of each day through `OPTIONS DAYCLOSED` (see Section 8.2.6).
- Version 1.11 : (Jinzhong Niu, 12/22/2006) inputs from Peter McBurney added.
- Version 1.12 : (Jinzhong Niu, 02/28/2007) paragraph added describing client-specified client IDs in Section 7.1; typos fixed.
- Version 1.13 : (Jinzhong Niu, 04/15/2007) mistakes in examples in Section 8.2.6 corrected.
- Version 1.14 : (Jinzhong Niu, 04/20/2007) Section 3.1 updated and Section 8.2.8 added on the new `OPTIONS ROUNDCLOSING` message.
- Version 1.15 : (Jinzhong Niu, 05/15/2007) Section 8.1 modified and reconnection of clients not supported currently; Section 8.6 updated to include the case of `PROFIT`; Section 8.4 updated to reflect the fact that specialists are also allowed to subscribe to market information; Section 8.3 updated to reflect the fact that a trader is allowed to register with none of the specialists on a day.
- Version 1.16 : (Jinzhong Niu, 05/30/2007) More details added in Section 10.2 ; reconnection of failed clients supported and Section 8.1 updated.
- Version 1.17 : (Jinzhong Niu, 06/22/2007) `Time` header field added in `OPTIONS DAY*` and `OPTIONS ROUND*` requests to present concrete time information of game, and Sections 8.2 and 10.1 updated.
- Version 1.18 : (Jinzhong Niu, 07/03/2007) Section 8.2.4 added `Value` header field in a trader's response to `OPTIONS DAYOPENING` message, providing the daily entitlement of the trader to the game server; a note added in Section 8.2.5 regarding private value assignment in the case of entitlement larger than 1.
- Version 1.19 : (Jinzhong Niu, 06/12/2008) Sections 1, 3.2.2, 3.4, 6, 7, and 8.2.1 updated.

- Version 2.00 : (Jinzhong Niu, 04/10/2009)
  - Minor edits and additions to text in all sections;
  - JCAT site added in Section 1;
  - Section 3.3 on specialists added; and
  - illustration of configuring and allocating client IDs added in Section 8.1.
- Version 2.01 : (Jinzhong Niu, 06/24/2010)
  - Minor edits and additions to text in Section 8.7.
- Version 2.02 : (Jinzhong Niu, 09/13/2010)
  - Section 8.9 and Figure 8 updated on the use of `TRANSACTION` rather than `POST TRANSACTION` in notifying traders that their shouts are matched.
  - Section 5.1 updated and CATP version increased to 2.0 from 1.1.

# 3 CAT games

This section introduces the basic configuration of CAT games, helping make this document self-explanatory. For more information, please refer to [1].

A CAT game consists of a CAT server and several CAT clients, which may be trading agents or specialists (markets). CAT clients do not talk to each other directly; instead they connect to the CAT server through socket and the server responds to messages from clients and forward information if needed.

## 3.1 Timing

A CAT game lasts a certain number of *day*s and each day consists of *round*s. The CAT server in a game has the control on a central clock and notifies clients of the following events:

- game starting
- game started
- game over
- day opening
- day opened
- day closed
- round opened
- round closing
- round closed

In order to allow traders and specialists to exploit some deadline effects, the server also notifies clients of the length of a day — in terms of number of rounds — and the length of each round — in terms of the number of physical milliseconds

— before a game starts. The length of a game — in terms of the number of days — is however not disclosed by the server to any of the clients, aiming to make specialists to operate in a sustainable manner rather than a greedy one.

## 3.2 Trading Agents

In a CAT tournament, trading agents are provided by the CAT game. Each trading agent is associated with a *trading strategy* and a *market selection strategy*. The former decides the prices to buy or sell at, while the latter decides which market the agent should go to trade each day.

The CAT server assigns private values for the goods an agent is entitled to trade every day. The quantity of all the goods and the private values form the demand and supply schedules of the global market, and may change from day to day. However, private values remain constant during a day.

### 3.2.1 Trading strategies

The set of trading strategies that may be used by traders in CAT games include:

- ZI-C (Zero Intelligence with Constraint) [4]
- ZIP (Zero Intelligence Plus) [2]
- RE (Roth and Erev) [10]
- GD (Gjerstad and Dickhaut) [3]

The composition of the trader population, i.e., how many traders adopting each of these strategies, will however not be made public to entrants. More information about these strategies can also be found in [1].

### 3.2.2 Market Selection strategies

Traders transact with each others through specialists. When a trading day opens, a trader registers with one specialist (and only one specialist), then trades with other traders that also registered with the specialist on that day, and stays with that specialist until the day closes. Traders may register with different specialists on different days. A trader is allowed to make shouts and transactions only through the specialist it registers with.

The strategies that traders use in order to choose between specialists are based on the solution concept to the $n$-armed bandit problem (see Chapter 2 in [12]). The $n$-armed bandit problem model allows using different types of exploration

policies, including the $\epsilon$-greedy rule and the softmax rule, and different calculation of expected action payoff, including the all-time average rule and the discounted sum rule. Different combinations of these rules form different market selection strategies. An investigation on the effect these market selection strategies may have on the competition between markets can be found in [8].[1]

## 3.3 Specialists

Specialists facilitate trade by matching offers and determining the trading price in an exchange market. Each specialist operates its own exchange market and may choose its own auction rules — the aim of the CAT tournament is to create a specialist that optimizes a particular set of measures (to be discussed in more details in next section). JCAT provides a reference implementation of a parameterizable specialist that can be easily configured and extended to use policies regulating different aspects of an auction [6, 7]. This parameterizable framework of specialist includes the following components:

- *Matching policies* define the set of matching offers in a market at a given time.
- *Quoting policies* determine the ask quote and bid quote, which respectively specify the upper bound for offers to sell and the lower bound for offers to buy that may be placed in the market at a given time.
- *Shout accepting policies* judge whether a request by a trader to place an offer in the market should be accepted or rejected.
- *Clearing conditions* define when to clear the market and execute transactions between matched offers.
- A *pricing policy* is responsible for determining transaction prices for matched ask-bid pairs. The decision may involve only the prices of the matched offers, or more information including market quotes.
- *Charging policies* determine the charges a specialist imposes on a trading day. A specialist can set its fees, or *price list*, which are charged to traders and other specialists who wish to use the services provided by the specialist.

[1] The $n$-armed bandit problem assumes stationary distributions of expected payoff for actions. This does not completely fit into the market competition scenario, where market mechanisms may be adaptive. However, a discounting rule that gives more weights to recent dynamics to some extent alleviates this concern. Overall, the $n$-armed bandit problem with a dynamic environment is still an open research problem.

Each specialist is free to set the level of the charges for registration with a specialist, for making an offer, for completing a transaction, and so on.

Specialists may have adaptive strategies such that the policies change during the course of a game in response to market conditions for desired outcomes. The above framework is independent of CATP and entrants may or may not follow this framework in designing their market mechanisms.

### 3.3.1 Fees

A specialist can make a profit by charging:

- *registration fee* when a trader is registered with the specialist on a trading day;
- *information fee* when a trader or a specialist requests to receive information on shouts and transactions taking place through the specialist;
- *shout fee* when a trader successfully places a shout through the specialist;
- *transaction fee* when a trader makes a transaction; and
- *profit fee* upon the profit a trader makes in a transaction ().

A specialist is free to set the level of these charges (from zero up). The first four types of fees are each a flat charge, and the last one is a percentage charged on the profit made by a trader during a transaction. The *profit* is defined as the difference between the price the trader offered and the trading price.

Specialists are required to send price lists to the server when a trading day opens. Fees remain constant during a day, but are permitted to change as specialists adapt to the environment over time.

## 3.4 Evaluation

Specialists in a CAT game are evaluated across a number of performance metrics. One such metric is the profit they make through the game. This profit is the sum of all the fees the specialist receives. If a specialist pays to obtain information on shouts and transactions taking place in markets run by other specialists, the charges incurred are deducted from its profit.[2]

Prior CAT tournaments evaluate specialists by the profit that they make, the number of trading agents they attract, and the success rate of offers made in their

[2] A specialist may pay itself and thus incur no expense to obtain additional information on activities happening in its own market.

markets. And at the end of each trading day, the CAT server informs all the specialists of the profits they have made so far and trader distribution among specialists, which allows specialists to learn to adapt their mechanisms to improve their performance over the long term. See Section 8.5 for more information on these notifications.

The detailed evaluation process to be used in the CAT game is described in [1].

# 4 Notational Conventions

The following rules from the HTTP 1.1 Specification [5] are used throughout this specification to describe basic parsing constructs.

```
OCTET          = <any 8-bit sequence of data>
CHAR           = <any US-ASCII character (octets 0 - 127)>
UPALPHA        = <any US-ASCII uppercase letter "A".."Z">
LOALPHA        = <any US-ASCII lowercase letter "a".."z">
ALPHA          = UPALPHA | LOALPHA
DIGIT          = <any US-ASCII digit "0".."9">
CTL            = <any US-ASCII control character
                 (octets 0 - 31) and DEL (127)>
CR             = <US-ASCII CR, carriage return (13)>
LF             = <US-ASCII LF, linefeed (10)>
SP             = <US-ASCII SP, space (32)>
HT             = <US-ASCII HT, horizontal-tab (9)>
<">            = <US-ASCII double-quote mark (34)>

CRLF           = CR LF

LWS            = [CRLF] 1*( SP | HT )

TEXT           = <any OCTET except CTLs,
                  but including LWS>

HEX            = "A" | "B" | "C" | "D" | "E" | "F"
               | "a" | "b" | "c" | "d" | "e" | "f" | DIGIT

token          = 1*<any CHAR except CTLs or separators>
separators     = "(" | ")" | "<" | ">" | "@"
               | "," | ";" | ":" | "\" | <">
               | "/" | "[" | "]" | "?" | "="
               | "{" | "}" | SP | HT
```

# 5 Protocol Parameters

## 5.1 CATP Version

CATP uses a <major>.<minor> numbering scheme to indicate versions of the protocol. The protocol version information is intended to allow the sender to indicate the format of a message and its capacity for understanding further CATP communication, rather than the features obtained via that communication. No change is made to the version number for the addition of message components which do not affect communication behavior or which only add to extensible field values. The <minor> number is incremented when the changes made to the protocol add features which do not change the general message parsing algorithm, but which may add to the message semantics and imply additional capabilities of the sender. The <major> number is incremented when the format of a message within the protocol is changed.

```
CATP-Version   = "CATP" "/" 1*DIGIT "." 1*DIGIT
```

The major and minor numbers must be treated as separate integers and that each may be incremented higher than a single digit. See Section 8.1 for how CATP version is used in CHECKIN messages.

The current version of CATP is 2.0.[3]

# 6 Connections

A CATP connection is setup between a CAT server and a CAT client for message transmission.

A server listens on port 9090[4] upon startup.[5] Clients connect to the port and establish connections. CATP connections work in a way similar to a persistent HTTP connection. Normally they should be kept alive from the beginning of a game through the end and process all the request and response sessions.[6]

[3] The version number of CATP increased from 1.1 to 2.0 in releasing JCAT 0.17 and version 2.02 of this document. A specialist client that works with earlier versions of JCAT up to 0.10 and CATP 1.1 is expected to work normally with JCAT 0.17.

[4] The port number could be chosen arbitrary as long as it does not conflict with any existing known service.

[5] For the sake of convenience in doing experiments, JCAT supports three different ways of communication between the game server and game clients: asynchronous socket based, asynchronous message-queue based, and method invocation based. The method invocation method is usually the fastest to simulate and the easiest way to debug. For more details, you would need to check the configuration of JCAT.

[6] In JCAT, multiple games can be run in a batch mode, for example during a research experiment, and the CAT server will not attempt to close the connections with clients until the batch run is completed

In case a connection is broken while a game is on-going, it is the responsibility of the client involved to detect the event, establish a new connection to the server, and rejoin the game beginning from the next trading day.[7]

# 7 CATP Messages

In a CAT game, plain-text messages are transmitted over CATP connections for the CAT server and CAT clients to communicate with each other. Since a message is always transmitted over a point-to-point CATP connection, there is no need to include the identities of the sender and the receiver in the message. It is the server's responsibility to keep copies of messages from clients in a certain way for the purposes of security, robustness, and game-wide information integrity.

## 7.1 Client ID

Each client, trader or specialist, in a CAT game is assigned a unique ID by the server, which is used in messages whenever needed to specify a client. A client ID should be a `token`.

```
client-id   = token
```

A client is also allowed to propose an ID when it checks in after getting connected to the server. This helps to identify certain clients of concern. For more information, please refer to Section 8.1.

## 7.2 Shout ID

After a shout is placed by a trader and confirmed valid by the server, it is assigned a unique ID by the server, which follows the identical syntax to that of client IDs.

```
shout-id   = token
```

## 7.3 Transaction ID

After a transaction is made by a specialist and confirmed valid by the server, it is assigned a unique ID by the server, which again follows the identical syntax to that of client IDs.

```
transaction-id   = token
```

[7] When a client needs to re-establish a connection, it should check in first before synchronizing with the server on the next trading day. See Section 8.1.

## 7.4 Message Types

Following the convention of HTTP messages, CATP messages include requests and responses. However note that requests can be made by either a client or server in CATP, which is not true in HTTP.

```
message    = request | response
```

Both types of message consist of a `start-line`, zero or more header fields (also known as `headers`), an empty line (i.e., a line with nothing preceding the `CRLF`) indicating the end of the header fields, and possibly a `message-body`.

```
generic-message = start-line
                  *(message-header CRLF)
                  CRLF
                  [ message-body ]
start-line      = message-type CRLF
message-type    = request-type
                | response-type
```

Often, the `message-type` is sufficient to identify the purpose of a message. When this is not the case, a `Type` header is used to provide additional information (see Sections 8.1, 8.2, 8.5, 8.6, 8.7, 8.8, and 10.4 for more information).

## 7.5 Message Headers

CATP message header fields are included in messages to provide additional information.

Each header field consists of the field name followed by a colon (`:`) and the field value. Field names are case-insensitive. The field value may be preceded by any number of `LWS`s, though a single `SP` is preferred.

```
message-header = field-name ":" [ field-value ]
field-name     = token
field-value    = *( field-content | LWS )
field-content  = <the OCTETs making up the field-value
                 and consisting of either *TEXT or
                 combinations of token, separators,
                 and quoted-string>
```

The `field-content` does not include any leading or trailing `LWS` occurring before the first non-whitespace character of the `field-value` or after the last non-whitespace character of the `field-value`. Such a leading or trailing `LWS` may be removed without changing the semantics of the field value. Any `LWS` that occurs between `field-content` may be replaced with a single `SP` before interpreting the field value or forwarding the message downstream.

The order in which header fields with differing field names are received is not significant.

Multiple `message-header` fields with the same `field-name` may be present in a message if and only if the entire `field-value` for that header field is defined as a comma-separated list. It must be possible to combine the multiple header fields into one `field-name:field-value` pair, without changing the semantics of the message, by appending each subsequent `field-value` to the first, each separated by a comma.

### 7.6 Message Body

No CATP message currently uses a message body.

## 8 Requests

A request message requests the receiver to take certain actions or to notify the receiver of certain information. Types of request include:

```
request-type   = "CHECKIN"
               | "REGISTER"
               | "SUBSCRIBE"
               | "ASK"
               | "BID"
               | "TRANSACTION"
               | "GET"
               | "POST"
               | "OPTIONS"
```

### 8.1 CHECKIN

A `CHECKIN` request is used by a CAT client to take part in a CAT game once it connects to the server. A `Version` header, a `Type` header and a `Text` header are required in a `CHECKIN` request to provide the version of CATP the client supports, the type and the human-readable name of the CAT client.

EXAMPLE:

```
CHECKIN CRLF
Version: CATP/1.0 CRLF
Type: Specialist CRLF
Text: MetroCat from CUNY CRLF
CRLF
```

The type of a client must be a string starting with one of the following (case insensitive):

- `Buyer`: indicating the client is a trading agent that buys;

- `Seller`: indicating the client is a trading agent that sells; and
- `Specialist`: indicating the client is a specialist.

On receiving a `CHECKIN` request, the server allocates a client ID and sends back an `OK` response if the type of client is valid and the version of CATP supported by the server matches the version of CATP on the client side.

EXAMPLE:

```
OK CRLF
Id: Specialist5 CRLF
CRLF
```

Otherwise, an `INVALID` message is replied.

EXAMPLE:

```
INVALID CRLF
Type: Version CRLF
Text: catp versions don't match. CRLF
CRLF
```

In the case that a client loses the connection with the server and attempts to establish a new connection, the client needs to make a `CHECKIN` request again before getting back to the on-going game and starting to play from the next trading day. The `CHECKIN` request sent in this scenario should include an `Id` header to present the ID allocated to the client before, and no other headers are required.

EXAMPLE:

```
CHECKIN CRLF
Id: Specialist5 CRLF
CRLF
```

If the ID is recognized, the server simply sends back an empty `OK` response, or otherwise an `INVALID` response. Of course, the randomly allocated ID has to be remembered by a failed client so as to rejoin the game. This difficulty can be avoided since a CAT client is also allowed to propose an ID by using an `Id` header when it checks in the first time. If the ID does not conflict with IDs already allocated to other clients, the server will respond with an `OK` with an optional `Id` header; otherwise, the ID in the request is simply overlooked and a new ID allocated to the client and sent back as usual.

EXAMPLE:

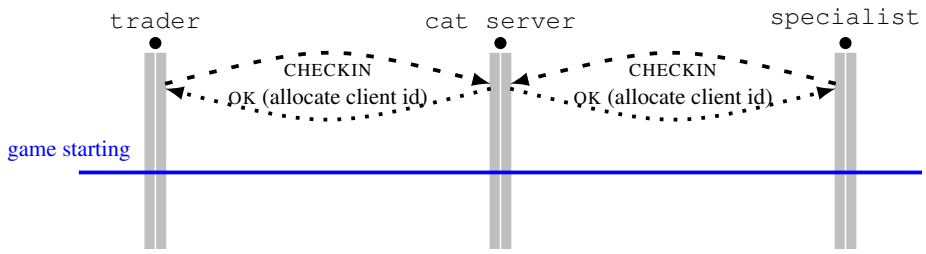


Figure 1: CHECKIN messages.

```
CHECKIN CRLF
Version: CATP/1.0 CRLF
Type: Specialist CRLF
Text: MetroCat from CUNY CRLF
Id: MetroCat CRLF
CRLF
```

Once a CAT game starts, it will not accept new clients to join. However, as for an actual CAT tournament, a list of expected specialist names are collected in advance and made known to the server, so that an expected specialist may still be able to join even if it checks in late. This white list enables entrants to deal with unexpected delays.

## 8.2 OPTIONS

OPTIONS requests are control messages sent by the CAT server to clients, involving timing, demand/supply schedule, etc. Types of OPTIONS messages are described in a Type header.

As Section 3.1 already stated, a CAT game lasts a certain number of days and each day consists of rounds. The CAT server notifies clients of the start and the end of a game, a day, and a round. There are both a *game starting* event and a *game started* event, between which initialization operations can be done. Similarly, there are both a *day opening* event and a *day opened* event on each trading day allowing daily initialization operations. There is however no such an initialization phase designed for a round, and there is only a *round opened* event. Instead, there are both *round closing* event and a *round closed* event at the end of each round allowing the CAT server and clients to perform necessary operations immediately before a round closes, e.g., a specialist clearing its market when a round is closing.

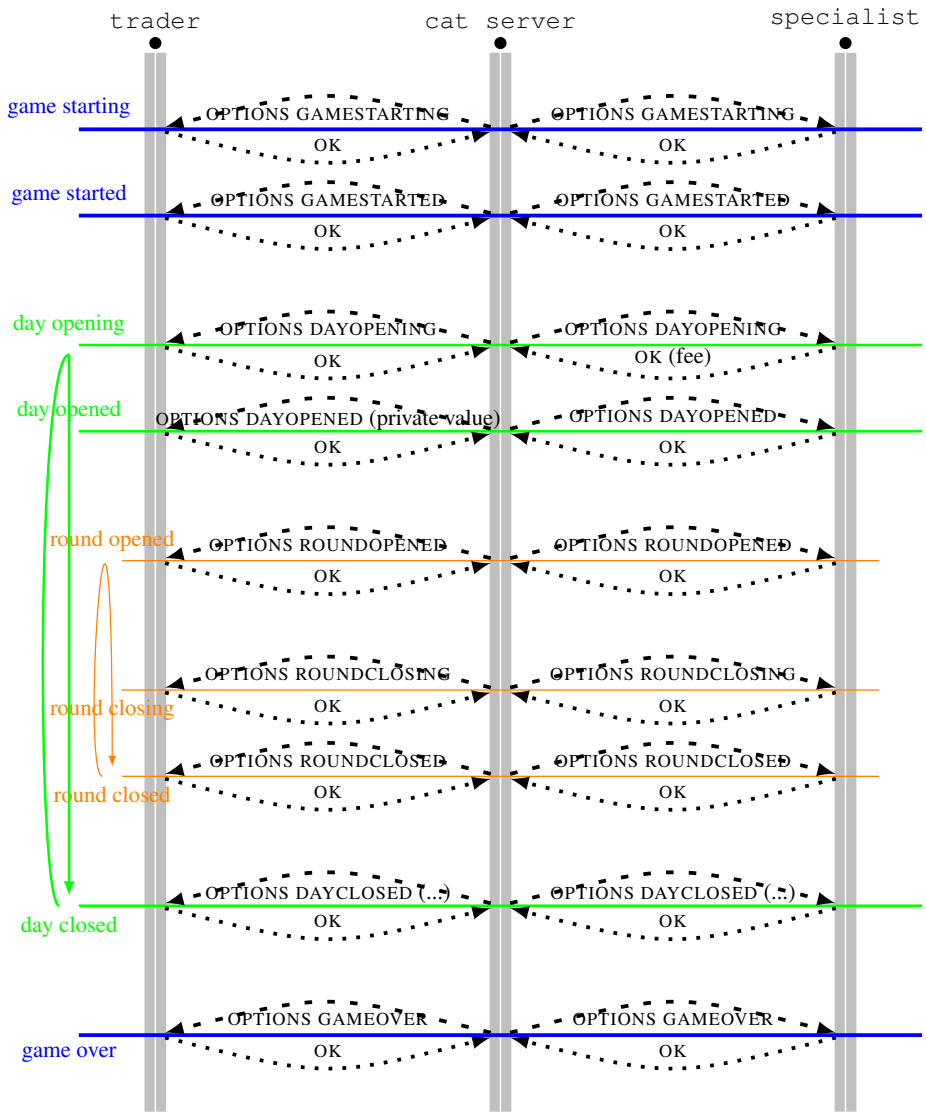


Figure 2: OPTIONS messages.

OPTIONS requests relating to day or round include a Time header presenting concrete time information about which day or round the messages are about.

Figure 2 shows the order of various OPTIONS messages from the CAT server to clients. These messages show the time control in a CAT game.

### 8.2.1 GAMESTARTING

An OPTIONS GAMESTARTING message notifies CAT clients of the starting of the game. A Value header with two integer values should be used to inform clients of the length of a day and the length of a round. An OK response from each client is expected.

EXAMPLE:

```
OPTIONS CRLF
Type: GAMESTARTING CRLF
Value: 20, 3600 CRLF
CRLF
```

The length of a game in terms of days is however not released to CAT clients to prevent specialists exploiting deadline effects due to an predetermined end of the game. After all, CAT games aim to promote the design of market mechanisms with sustainable behavior, and a good mechanism should work well over a long run.

The OPTIONS GAMESTARTING message is followed by a POST TRADER message and a POST SPECIALIST message.

### 8.2.2 GAMESTARTED

An OPTIONS GAMESTARTED message notifies CAT clients that the game just started. An OK response from each client is expected.

EXAMPLE:

```
OPTIONS CRLF
Type: GAMESTARTED CRLF
CRLF


OK CRLF
CRLF
```

### 8.2.3 GAMEOVER

An OPTIONS GAMEOVER message notifies CAT clients of the end of the game. An OK response from each client is expected.

EXAMPLE:

```
OPTIONS CRLF
Type: GAMEOVER CRLF
CRLF


OK CRLF
CRLF
```

### 8.2.4 DAYOPENING

An `OPTIONS DAYOPENING` message notifies CAT clients of the opening of a trading day.

EXAMPLE:

```
OPTIONS CRLF
Type: DAYOPENING CRLF
Time: 3, -1, -1 CRLF
CRLF
```

A `Time` header is used by certain types of messages, including `OPTIONS DAYOPENING` messages, to specify the concrete time when an action of concern in the context occurs or the message is about. The above example tells that it is day 3 that is opening. See Section 10.1 for more information.

For a specialist, the response should include a price list in a `Value` header. See Sections 3.3.1, 8.5, and 8.6.

EXAMPLE:

```
OK CRLF
Value: 2, 3, 6, 9, 0.2 CRLF
CRLF
```

A price list includes in order registration fee, information fee, shout fee, transaction fee, and profit fee as described in Section 3.3.1. In the above example, the specialist charges $2 on registration, $3 on subscription for information from the specialist, $6 on each shout successfully placed with the specialist,[8] $9 on a trader for each transaction it is involved in, and 20% of profit a trader makes in a transaction.

For a trader, the response should include the value of daily entitlement in a `Value` header. That is the number of goods the trader is entitled to trade each day. The reason for the server to collect this information is that the server is thus able to

[8] Only new shouts are charged, and shouts that modify existing ones will not be charged again no matter whether the modification is granted or not.

determine the supply and demand schedules of the global market for experimental and statistical purposes.[9]

EXAMPLE:

```
OK CRLF
Value: 3 CRLF
CRLF
```

### 8.2.5 DAYOPENED

An `OPTIONS DAYOPENED` message notifies CAT clients of the opening of a trading day. An `OK` response from each trader is expected. In `OPTIONS DAYOPENED` messages to traders, a `Value` field should be include to assign private values.

EXAMPLE:

```
OPTIONS CRLF
Type: DAYOPENED CRLF
Time: 3, -1, -1 CRLF
Value: 65 CRLF
CRLF


OK CRLF
CRLF
```

In theory, a trader may have different private values for multiple units of goods. Currently it is only supported that all units share a same private value, e.g., 65 in the above example. In the future, the `Value` field may contain a list of numbers, each as a private value for one unit of goods.

### 8.2.6 DAYCLOSED

An `OPTIONS DAYCLOSED` message notifies CAT clients that a trading day is closed. In the message, the CAT server also provides the number of traders being registered with each specialist by using an `Id` header with a list of specialists and a `Value` header with a list of integers. An `OK` response from each client is expected.

EXAMPLE:

[9] We assume that a trader always truthfully report its daily entitlement since all traders are configured and run by the game organizers in a CAT tournament. It is also possible that the entitlement information is configured in the game server and the game server distributes the information to traders, but this alternative would be clumsy since it is more natural to assume that traders themselves know their supply and demand.

```
OPTIONS CRLF
Type: DAYCLOSED CRLF
Time: 3, -1, -1 CRLF
Id: Specialist0, Specialist1, Specialist2, Specialist3 CRLF
Value: 20, 14, 53, 13 CRLF
CRLF


OK CRLF
CRLF
```

### 8.2.7 ROUNDOPENED

An OPTIONS ROUNDOPENED message notifies CAT clients of the start of a trading round. An OK response from each client is expected.

EXAMPLE:

```
OPTIONS CRLF
Type: ROUNDOPENED CRLF
Time: 3, 5, -1 CRLF
CRLF


OK CRLF
CRLF
```

### 8.2.8 ROUNDCLOSING

An OPTIONS ROUNDCLOSING message notifies CAT clients that a trading round is closing. An OK response from each client is expected.

EXAMPLE:

```
OPTIONS CRLF
Type: ROUNDCLOSING CRLF
Time: 3, 5, -1 CRLF
CRLF


OK CRLF
CRLF
```

### 8.2.9 ROUNDCLOSED

An OPTIONS ROUNDCLOSED message notifies CAT clients that a trading round is closed. An OK response from each client is expected.

EXAMPLE:

```
OPTIONS CRLF
Type: ROUNDCLOSED CRLF
Time: 3, 5, -1 CRLF
CRLF


OK CRLF
CRLF
```

## 8.3 REGISTER

A REGISTER request from a trader client to the server expresses its desire to trade through the specialist specified in an Id header.

EXAMPLE:

```
REGISTER CRLF
Id: specialist3 CRLF
CRLF
```

If the registration is successful, the server responds with an empty OK message.

EXAMPLE:

```
OK CRLF
CRLF
```

The server then notifies the corresponding specialist of the trader's registration with a REGISTER request, which uses an Id header to specify which trader is registered.

EXAMPLE:

```
REGISTER CRLF
Id: Seller3 CRLF
CRLF
```

If everything goes normal, the specialist replies with an OK response. Specialists do not have the right to reject a trader for registration.

REGISTER requests are expected to be sent by traders after a trading day is opened. If a trader is not willing to register with any specialist at all, an empty REGISTER request without the Id header or with an empty Id header should be used.

The processing of a REGISTER request is illustrated in Figure 3.

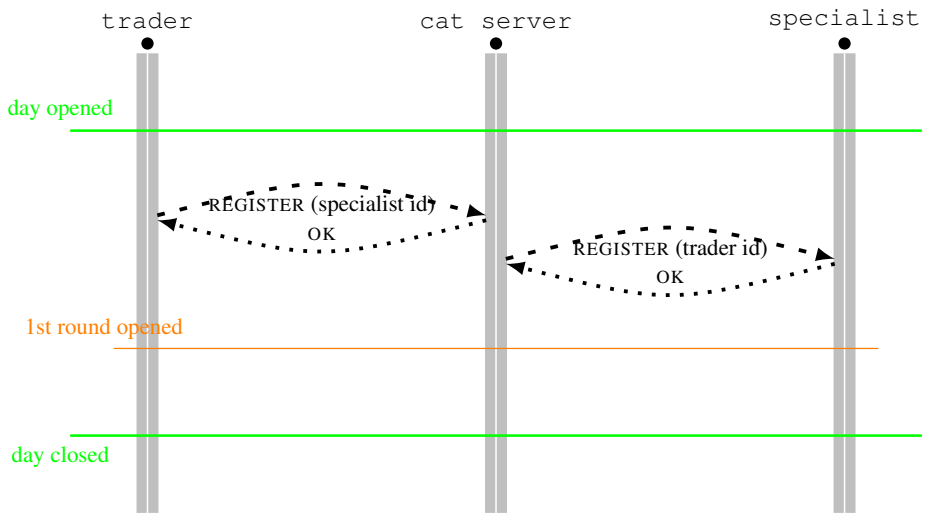


Figure 3: REGISTER messages.

## 8.4 SUBSCRIBE

A SUBSCRIBE request from a client to the server tells the latter that the client would like to subscribe for information regarding shouts and transactions made through some specialists. An Id header is used to give a list of specialists from which the client wants to purchase information.

EXAMPLE:

```
SUBSCRIBE CRLF
ID: specialist1, specialist3 CRLF
CRLF
```

The server responds to a SUBSCRIBE message with an OK message if all the specialists are active. If some of the specialists are not active due to broken connections or lack of price list announcement, an additional Id header is used to specify those specialists with which the client has successfully subscribed.

EXAMPLE:

```
OK CRLF
ID: specialist3 CRLF
CRLF
```

The server then notifies those active specialists involved by sending a SUBSCRIBE message with an Id header telling which trader subscribes.

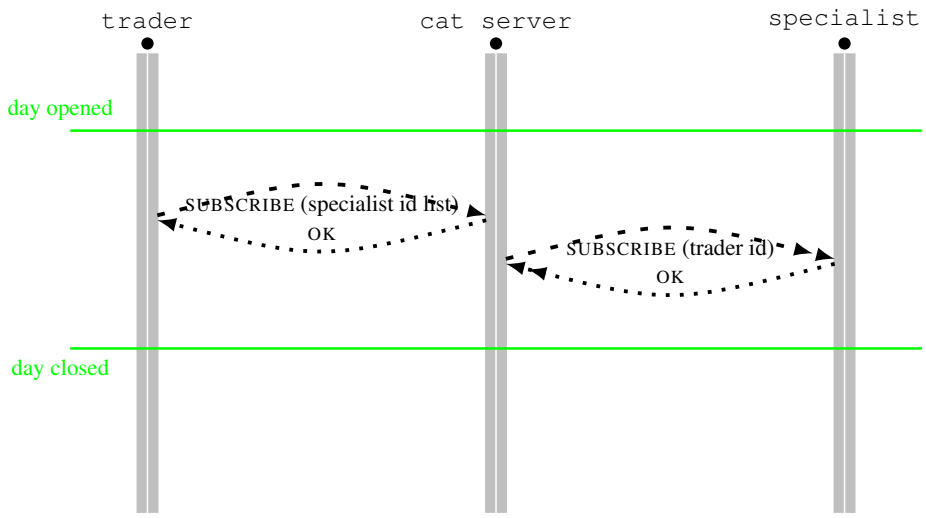


Figure 4: SUBSCRIBE messages.

EXAMPLE:

```
SUBSCRIBE CRLF
ID: trader2 CRLF
CRLF
```

A specialist should reply to a SUBSCRIBE message with an OK message. Similar to the registration scenario, a specialist does not have the right to reject a subscription. But different from the registration, which must occur before the first round each day, SUBSCRIBE requests can be sent any time during a trading day and the server is responsible for broadcasting the necessary information to subscribed traders.[10] Its processing is illustrated in Figure 4.

## 8.5 POST

POST messages are used to proactively provide (push) information to receivers. The information may also be requested otherwise by CAT clients through GET messages.

The following information may be provided through POST messages:

[10] Certain trading strategies, including ZIP and GD, require information on shouts and transactions in the market so as to perform normally. In JCAT, a trader using one of these strategies makes a SUBSCRIBE request immediately to the specialist that it just successfully registered with.

- TRADER and SPECIALIST: After a CAT game starts but before the first day opens, the game server sends two POST messages to each of the traders and the specialists providing the list of traders, and the list of specialists participating the game.

  EXAMPLE:

  ```
  POST CRLF
  Type: TRADER CRLF
  Id: Buyer0, Buyer1, Seller0 CRLF
  CRLF

  POST CRLF
  Type: SPECIALIST CRLF
  Id: Specialist0, Specialist1, Specialist2 CRLF
  CRLF
  ```

- FEE: As described in Section 8.2.4, specialists include their price lists in the responses to OPTIONS DAYOPENING messages from the server. After the server receives a price list, a POST FEE message will be sent to each client within the day initialization period, namely between DAYOPENING and DAYOPENED. The ID of the specialist that the price list belongs to should be specified in an Id header.

  EXAMPLE:

  ```
  POST CRLF
  Type: FEE CRLF
  Id: Specialist0 CRLF
  Value: 0, 0, 0, 0, 0 CRLF
  CRLF

  ...

  POST CRLF
  Type: FEE CRLF
  Id: Specialist2 CRLF
  Value: 3, 3, 6, 9, 0.3 CRLF
  CRLF
  ```

  In the above example, Specialist2 charges a trader $3 to register, a subscriber $3 for information from this specialist, a trader $6 on each shout it places successfully, a trader $9 on each transaction it is involved in, and 20% of profit a trader makes through a transaction.

- PROFIT: The server notifies clients of the profits made by specialists through POST PROFIT messages at the end of each trading day, just before the OPTIONS DAYCLOSED message. An Id header presents the list of IDs of specialists and a Value header presents the amounts of their profits up to the day in the order of specialists in the Id header.

  EXAMPLE:

  ```
  POST CRLF
  Type: PROFIT CRLF
  Id: Specialist0, Specialist1, Specialist2, Specialist3 CRLF
  Value: 123, 456, 789, 1023 CRLF
  CRLF
  ```

- ASK and BID: The server sends notification of successful asks and bids, see Sections 8.7 and 8.8.

- TRANSACTION: The server sends notification of successful transactions, see Section 8.9.

The processing of POST TRADER, POST SPECIALIST, POST FEE, and POST PROFIT messages is illustrated in Figure 5, while the scenarios involving POST ASK, POST BID, and POST TRANSACTION are illustrated respectively in Figures 6, 7, and 8.

## 8.6 GET

A GET request from a CAT client to the server asks for information that is typically broadcast by the CAT server through POST messages as described in Section 8.5. The GET messages are useful when a CAT client loses the information previously received.

A Type header should be used to specify what type of information is requested. It can be

- TRADER and SPECIALIST: for the list of traders and the list of specialists respectively. An Id header should be included by the server in the response providing the list of client IDs requested.

  GET TRADER and GET SPECIALIST requests can be made any time after a CAT game started.

- FEE: for fees charged by a specialist. The specialist ID should be specified in an Id header.

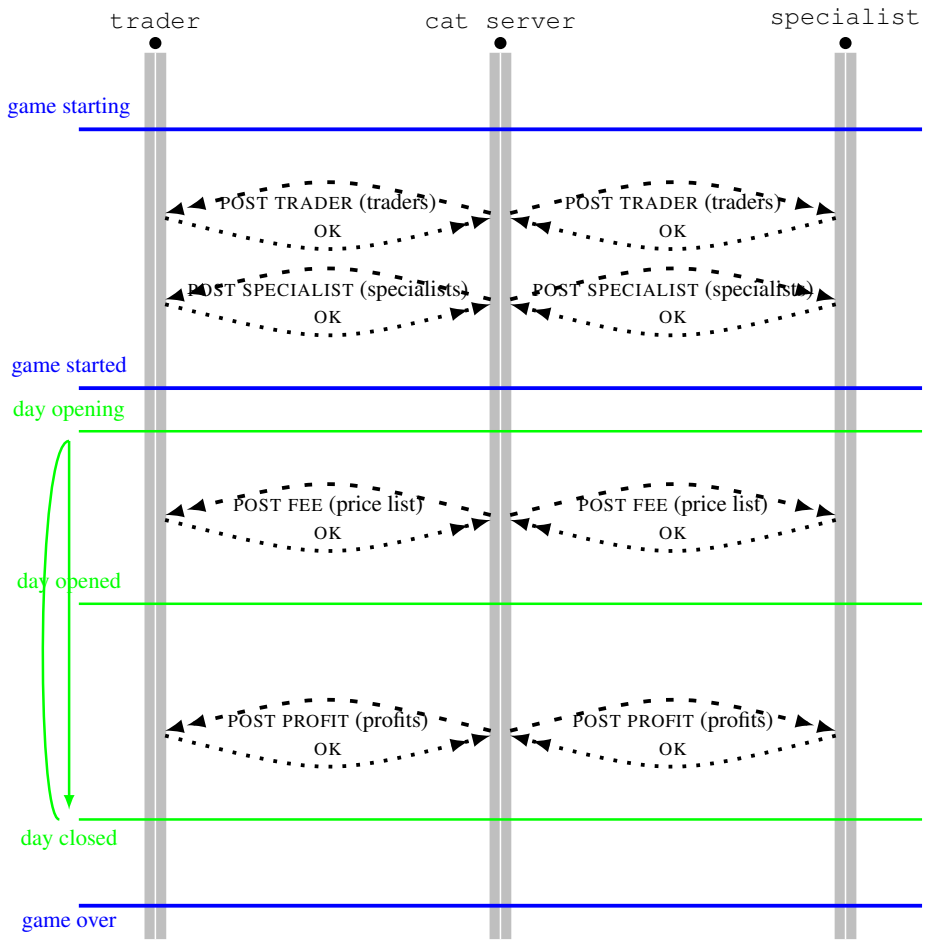


Figure 5: `POST` messages.

EXAMPLE:

```
GET CRLF
Type: FEE CRLF
Id: Specialist3 CRLF
CRLF
```

A positive response from the server should include a `Value` header presenting the price list with comma as separator and optionally the specialist ID.

EXAMPLE:

```
OK CRLF
Id: Specialist3 CRLF
Value: 2, 5, 4, 6, 0.2 CRLF
CRLF
```

A `GET FEE` request can only be made after a trading day is opened and before it is closed. The price list returned is the one the specialist of concern announced for the current trading day.

- `PROFIT`: for profits the specialists have made.

  EXAMPLE:

  ```
  GET CRLF
  Type: PROFIT CRLF
  CRLF
  ```

  A positive response from the server should include an `Id` header presenting the list of IDs of specialists and a `Value` header presenting the amounts of their profits up to the day in the order of specialists in the `Id` header.

  EXAMPLE:

  ```
  OK CRLF
  Id: Specialist0, Specialist1, Specialist2, Specialist3 CRLF
  Value: 123, 456, 789, 1023 CRLF
  CRLF
  ```

## 8.7 ASK

An `ASK` request from a seller to the server expresses an offer to sell through the specialist the seller registered with. A `Value` header is required to present the ask price.

EXAMPLE:

```
ASK CRLF
Value: 87 CRLF
CRLF
```

When a new ask arrives at the server, the server first checks the validity of the ask. If the ask is found invalid — for example, the price will make a loss to the trader — the server sends an `INVALID` response.

EXAMPLE:

```
INVALID CRLF
CRLF
```

Otherwise, the server allocates a unique ID for the ask and forwards the ask request to the specialist the seller registered with.

EXAMPLE:

```
ASK CRLF
Id: ask2 CRLF
Value: 87 CRLF
CRLF
```

After an `ASK` message is received by the specialist, it may be accepted or rejected depending upon the rules of the specialist. If the ask is accepted, the specialist sends an empty `OK` response to the server. The server then notifies the seller and those subscribers for information from this specialist, but in a different way. To the seller, an `OK` response is sent with an `Id` header, which includes the ask ID.

EXAMPLE:

```
OK CRLF
Id: ask2 CRLF
CRLF
```

To the information subscribers, a `POST` message is used with `ASK` in a `Type` header, the price in a `Value` header, and the ask ID, the seller ID, and the specialist ID in order in an `Id` header. The receivers should simply respond with an empty `OK` message to confirm.

EXAMPLE:

```
POST CRLF
Type: ASK CRLF
Id: ask2, seller0, specialist1 CRLF
Value: 87 CRLF
CRLF
```

If an ask is rejected by a specialist, an `INVALID` response should be sent to the server with optional `Type` and `Text` headers giving the type of reason and detailed description.

EXAMPLE:

```
INVALID CRLF
CRLF
```

The server then sends an INVALID response to the seller with SPECIALIST in a Type header. Since the ask is not accepted, only the seller is informed.

EXAMPLE:

```
INVALID CRLF
Type: SPECIALIST CRLF
CRLF
```

Figure 6 illustrates how an ask is processed in different scenarios.

As the above process shows, a specialist does not know which seller made the request to place an ask. Therefore it makes sense for a specialist to subscribe for information from itself, including, for example, the POST ASK message above, so as to receive additional information and make better decisions. In this case, the specialist receives the information for free since it pays to itself.

## 8.8 BID

A BID sent from a buyer to the server expresses an offer to buy through the specialist the buyer registered with. The processing of BID messages is similar to that of ASK messages. Figure 7 illustrates how bids are processed in different scenarios.

## 8.9 TRANSACTION

A TRANSACTION request from a specialist to the server proposes to make a transaction. An Id header is included to provide the corresponding ask ID and bid ID in order and a Value header for the transaction price.

EXAMPLE:

```
TRANSACTION CRLF
Id: ask3, bid2 CRLF
Value: 90 CRLF
CRLF
```

When the server receives a TRANSACTION request, it first checks whether this transaction is valid or not. If the transaction is valid, the server will allocate a unique transaction ID for the transaction and respond with an OK message with the ID in an Id header to the specialist.

EXAMPLE:

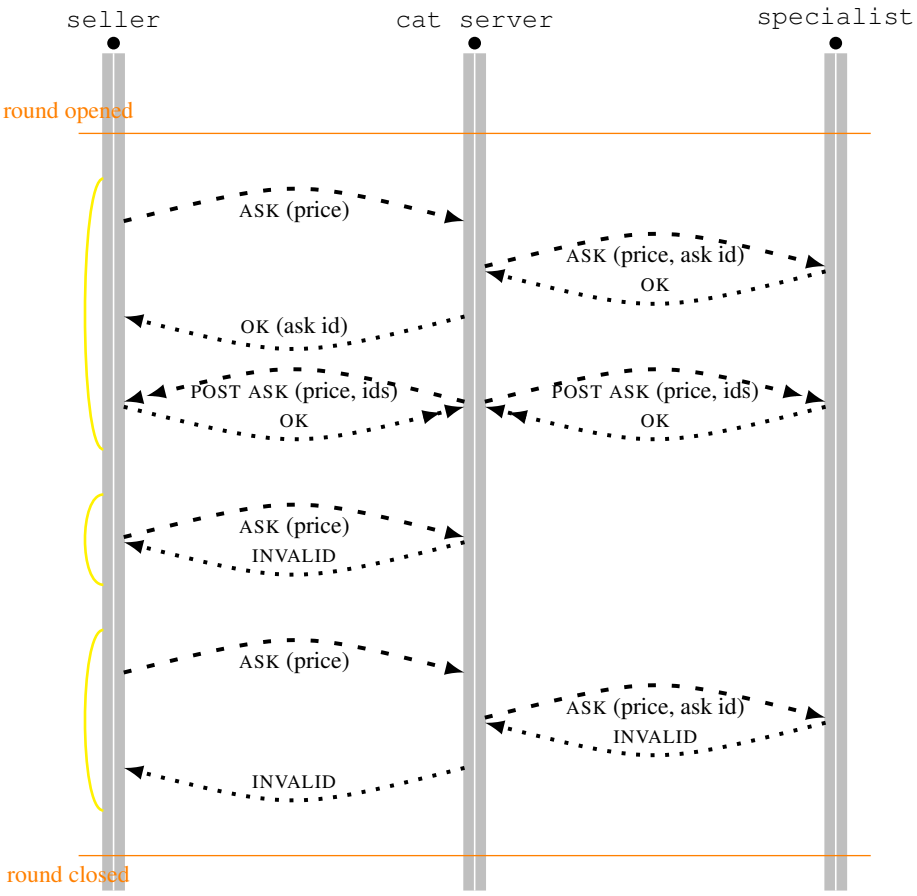


Figure 6: Dialogs regarding asks.

```
OK CRLF
Id: transaction2 CRLF
CRLF
```

The server then sends a `TRANSACTION` message to both the buyer and the seller involved in the transaction and a `POST` message to every information subscriber with the specialist. The `POST` message includes `TRANSACTION` in a `Type` header. Both the `TRANSACTION` message and the `POST` message include the transaction ID, the ask ID, the bid ID, the specialist ID in order in an `Id` header, and the transaction price, ask price, and bid price in order in a `Value` header.

EXAMPLE:

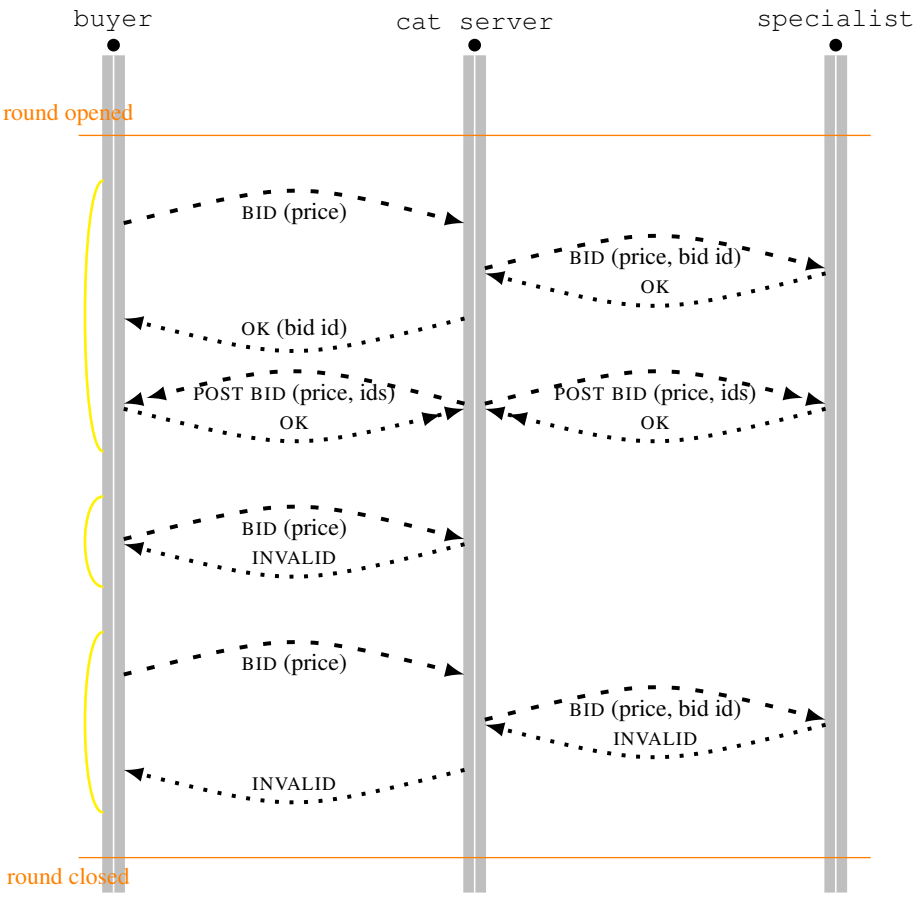


Figure 7: Dialogs regarding bids.

```
TRANSACTION CRLF
Id: transaction2, ask3, bid2, specialist0 CRLF
Value: 90, 67.5, 100.3 CRLF
CRLF

POST CRLF
Type: TRANSACTION CRLF
Id: transaction2, ask3, bid2, specialist0 CRLF
Value: 90, 67.5, 100.3 CRLF
CRLF
```

If the buyer or the seller is also one of the information subscribers, it receives both messages. A client that receives either message should respond with an empty `OK`

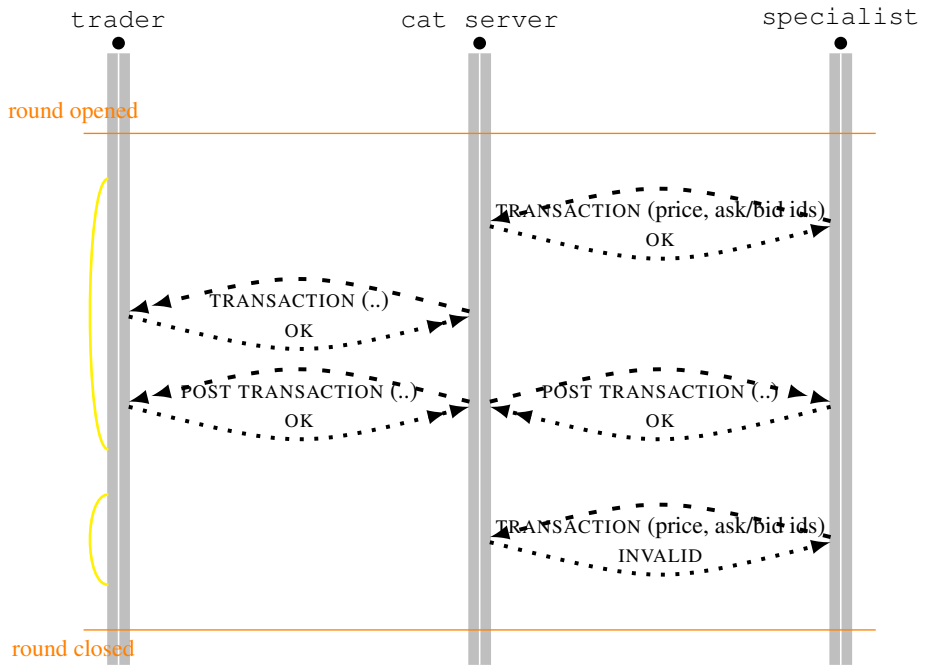


Figure 8: Dialogs regarding transactions.

message.

If the transaction is found invalid, the server should send back an `INVALID` message optionally with optional `Type` and `Text` headers giving respectively the reason of the message being invalid and detailed description. A transaction would be found invalid if the ask and the bid to be matched are not both active, or the transaction price does not fall into the bid-ask spread.

Note that a transaction cannot be rejected by the traders involved in a transaction. Its validity is guaranteed by the CAT server. This is different from some automated markets, for example the ones in the Santa Fe auction tournament [11] where traders are allowed to decide whether to accept a matching offer.

Figure 8 illustrates possible dialogs regarding transactions.

# 9 Responses

After a request message is received, a response message should be composed and sent back. Types of response include:

```
response-type   = "OK"
                | "INVALID"
                | "ERROR"
```

## 9.1 OK

An OK response acknowledges the success of a request. For example, a seller may receive a response like the following:

```
OK CRLF
Id: ask3
CRLF
```

informing that an ASK request is successful and the ask is associated with the ID ask3.

## 9.2 INVALID

An INVALID response indicates that the corresponding request is invalid, meaning that the request is syntax-correct but not expected or logically sound. For example, a seller sends out a BID request, or a seller sends out an ASK request that however fails to beat the corresponding market quote.

The INVALID message may need a Type header to tells why the request is invalid. The value of a Type header can be:

- WRONGTIME: An INVALID response with a WRONGTIME type is used when a request is made by a client at a wrong time. For example, an ASK request is made when a trading day is closed and the next day is not opened yet.

  EXAMPLE:

  ```
  INVALID CRLF
  Type: WRONGTIME CRLF
  CRLF
  ```

- SPECIALIST: When a shout is rejected by a specialist, the server sends an INVALID response to the trader with a SPECIALIST type. Please refer to Section 8.7 for more information.

  EXAMPLE:

```
INVALID CRLF
Type: SPECIALIST CRLF
CRLF
```

### 9.3 ERROR

An ERROR response informs the sender of the corresponding request that an error occurred. The ERROR message may include an optional Text header to give additional information. An ERROR response is used when either the request is syntactically wrong or some internal error occurs in the game server.

## 10 Message Headers

This section gives detailed descriptions of CATP header fields, as well as links to related message types.

### 10.1 Time

A Time header field specifies the time when an event occurred. Only CATP requests from the server may use a Time header. It is required to include a Time header in OPTIONS DAYOPENING, OPTIONS DAYOPENED, OPTIONS DAYCLOSED, OPTIONS ROUNDOPENED, OPTIONS ROUNDCLOSING, OPTIONS ROUNDCLOSED, POST ASK, POST BID, and POST TRANSACTION messages.

The content of a Time is a list of three numbers separated by commas, specifying respectively the day, the round, and the ticks into the round.

EXAMPLE:

```
Time: 2, 3, 156
```

When one of the three entries does not apply to the event of concern, -1 will be used. For example, a OPTIONS DAYOPENING message has -1 in the last two entries since only the day entry is needed to tell which day it is.

### 10.2 Tag

A Tag header field carries a tag that tells whether the current message is sent on the current day or not. Sometimes, due to network delay, a message sent on the previous day may arrive on the day next. The Tag header is used to detect and disregard obsolete messages.

The CAT server generates a tag when the following events occur and includes the tag in a Tag header in every message it sends out until a new tag replaces it:

- the game is starting;
- a day is opening; and
- the game is over.

Each client maintains a tag from the server and updates it when the above events are informed through OPTIONS messages. Every message a client sends out also have a Tag header presenting the current tag, with one exception that CHECKIN requests do not need such a header since the game has yet to start. Whenever a message from a client arrives, the server checks the tag. If the tag does not match the current tag of the server, the server simply disregards the message. The Tag header enables the game server and certain clients to synchronize with each other after these clients fail to keep up with the pace the server proceeds due to typically network delay.

### 10.3 Id

An Id header field provides one or more IDs, separated by commas if necessary. The number and semantics of IDs are determined by the context of the current message. See Sections 8.7, 8.8, 8.9, and 9.1 on how Id header fields are used in different scenarios.

### 10.4 Type

A Type header field gives the type of an entity. See Sections 8.1, 8.2, 8.5, 8.6, 9.2, and 9.3 on how Type header fields are used in different scenarios.

### 10.5 Value

A Value header field provides one or more numeric values. The number and semantics of values depend upon the context of the current message. See Sections 8.7, 8.8, and 8.9 on how Value header fields are used in different scenarios.

### 10.6 Text

A Text header field carries a plain-text string providing additional information. See Sections 8.1, 9.2, and 9.3 on how Text header fields are used in different scenarios.

### 10.7 Version

A Version header field tells the version of CATP being supported. For now, it is used only when a CAT client checks in. See Section 8.1.

# References

[1] Kai Cai, Enrico Gerding, Peter McBurney, Jinzhong Niu, Simon Parsons, and Steve Phelps. Overview of CAT: A market design competition. Technical Report ULCS-09-005, Department of Computer Science, University of Liverpool, Liverpool, UK, 2009. Version 2.0.

[2] Dave Cliff and Janet Bruten. Minimal-intelligence agents for bargaining behaviours in market-based environments. Technical Report HPL-97-91, Hewlett-Packard Research Laboratories, Bristol, England, August 1997.

[3] Steven Gjerstad and John Dickhaut. Price formation in double auctions. *Games and Economic Behavior*, 22:1–29, 1998.

[4] Dhananjay K. Gode and Shyam Sunder. Allocative efficiency of markets with zero-intelligence traders: Market as a partial substitute for individual rationality. *Journal of Political Economy*, 101(1):119–137, 1993.

[5] Hypertext Transfer Protocol – HTTP/1.1: Notational Conventions and Generic Grammar.

[6] Jinzhong Niu, Kai Cai, Simon Parsons, Enrico Gerding, and Peter McBurney. Characterizing effective auction mechanisms: Insights from the 2007 TAC Mechanism Design Competition. In Padgham, Parkes, Müller, and Parsons, editors, *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 1079–1086, Estoril, Portugal, May 2008.

[7] Jinzhong Niu, Kai Cai, Simon Parsons, Enrico Gerding, Peter McBurney, Thierry Moyaux, Steve Phelps, and David Shield. JCAT: A platform for the TAC Market Design Competition. In Padgham, Parkes, Müller, and Parsons, editors, *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, pages 1649–1650, Estoril, Portugal, May 2008. Demo Paper.

[8] Jinzhong Niu, Kai Cai, Simon Parsons, and Elizabeth Sklar. Some preliminary results on competition between markets for automated traders. In *Proceedings of AAAI-07 Workshop on Trading Agent Design and Analysis (TADA-07)*, Vancouver, Canada, July 2007.

[9] Jinzhong Niu, Albert Mmoloke, Peter McBurney, and Simon Parsons. CATP: A communication protocol for CAT games. Technical report, Department of

Computer Science, Graduate School and University Center, City University of New York, New York, NY, 2009. Version 2.0.

[10] Alvin E. Roth and Ido Erev. Learning in extensive-form games: Experimental data and simple dynamic models in the intermediate term. *Games and Economic Behavior*, 8:164–212, 1995.

[11] John Rust, John H. Miller, and Richard G. Palmer. Behaviour of trading automata in a computerized double auction market. In Daniel Friedman and John Rust, editors, *The Double Auction Market: Institutions, Theories and Evidence*, Santa Fe Institute Studies in the Sciences of Complexity, chapter 6, pages 155–199. Westview Press, Perseus Books Group, Cambridge, MA, 1993.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.