

CA110

Space API

David Gray
Version 1.0

4th March 2015

Contents

1 HTTP Requests	2
2 Common JSON Objects	2
2.1 Reason	2
2.2 3-D Coordinates	2
2.3 Player's Details	2
2.4 Inventory	3
3 Responses	4
4 Discovery API	5
4.1 getServers request	5
5 Authentication API	6
5.1 register request	6
5.2 completeRegister request	7
5.3 newPassword request	8
5.4 completeNewPassword request	9
5.5 authenticate request	10
5.6 version request	11
6 Trade API	12
7 Game API	13
7.1 Connection Handshake	13
7.2 Chatting	16
7.3 Other Players	18
7.4 Moving	19
7.5 Inventory	22

1 HTTP Requests

For the HTTP/JSON APIs, all requests use HTTP GET.

2 Common JSON Objects

2.1 Reason

A **Reason** object is a JSON object with the following fields:

`code`: An integer value as described in Table 1.

`reasonText`: A text description of the reason.

For example:

```
{
  "code":106,
  "reasonText":"Expired Token"
}
```

2.2 3-D Coordinates

A **3-D Coordinates** object is a JSON object with the following fields:

`x`: A real number.

`y`: A real number.

`z`: A real number.

For example:

```
{
  "x":42363.5374374,
  "y":3947394796.215,
  "z":846.26732
}
```

- For **positions**, the coordinates measure **meters**.
- For **directions/orientations** the coordinates measure **radians**.

2.3 Player's Details

A **Player's Details** object is a JSON object with the following fields:

`name`: The name of the player.

`ship`: The name of the player's ship.

`position`: The players position as a **3-D Coordinates** object.

orientation: The players orientation as a **3-D Coordinates** object.

For example:

```
{
  "username": "Master Yoda",
  "ship": "astratis_v1",
  "position": {
    "x": 626246,
    "y": 23526.2664,
    "z": 25.125
  },
  "orientation": {
    "x": 0.2,
    "y": 1.4,
    "z": 0
  }
}
```

2.4 Inventory

A **Inventory** object is a JSON object with the following fields:

Deferred

3 Responses

All HTTP/JSON requests respond with a JSON object containing a boolean `success` field that has one of two values:

1. `true`: The request has been successful and the JSON object will have other fields containing the results of the request.
2. `false`: The request has failed and the JSON object will have one other field:

`error`: A **Reason** object.

For example,

```
{
  "success":false,
  "error" : {
    "code":101,
    "reasonText":"Missing id parameter in http request"
  }
}
```

Code	Reason
0	Okay
100	Other error
101	Missing parameter in request
102	Unknown parameter in request
103	Unknown request
104	Server not ready or busy
105	?????
106	Authentication failure
107	Sender not the originator of chat message
108	Unknown player
109	Out of bounds
...	...

Table 1: Reason Codes

4 Discovery API

4.1 getServers request

4.1.1 Parameters

None

4.1.2 Response fields

addresses: A JSON object containing the URLs of the three servers:

```
authServer : URL string
gameServer : URL string
tradeServer : URL string
```

4.1.3 Semantics

The addresses of the servers to be used.

4.1.4 Example Exchange

Request: `http://???.getServers`

Response:

```
{
  "success":true,
  "addresses": {
    "authServer": "https://1.1.1.1:3000",
    "gameServer": "https://2.2.2.2:3001",
    "tradeServer": "https://2.2.2.2:3002"
  }
}
```

5 Authentication API

5.1 register request

5.1.1 Parameters

username: A string

password: A string¹

email: A string

5.1.2 Response fields

None

5.1.3 Semantics

The user's information is stored² in a database along with a registration token that allows the user to complete registration. If the user does not complete registration within `REGISTRATION.TIME.LIMIT` hours, **all** information about the user is deleted.

The user is sent an email that enables the use to perform a `completeRegister` operation using this registration token.

5.1.4 Example Exchange

Request:

```
http://???/register?username=Yoda
                        &password=sillypassword
                        &yoda@starwars.ie
```

Response:

```
{
  "status": "okay",
}
```

¹This is a plaintext password.

²The plaintext password should not be stored in the database.

5.2 completeRegister request

5.2.1 Parameters

`token`: A string

5.2.2 Response fields

None

5.2.3 Semantics

The registration `token` is a string that was associated with the user when they registered. Receipt of this registration `token` confirms that the user can access the registered email address and their registration is completed. The `token` is discarded.

5.2.4 Example Exchange

Request:

```
http://???/completeRegister?token=1324597283grgr12387g821gzz932e83246213
```

Response:

```
{  
  "status": "okay",  
}
```

5.3 newPassword request

5.3.1 Parameters

username: A string

5.3.2 Response fields

None

5.3.3 Semantics

A password `token` that allows the user to change their password is created and stored in the database. If the user does not complete changing their password within `PASSWORD_TIME_LIMIT` hours, the password `token` is discarded.

The user is sent an email that enables them perform a `completeNewPassword` operation using this password `token`.

5.3.4 Example Exchange

Request:

`http://???/newPassword?username=Yoda`

Response:

```
{
  "status": "okay",
}
```


5.4 completeNewPassword request

5.4.1 Parameters

`token`: A string

`password`: The new password for the user

5.4.2 Response fields

None

5.4.3 Semantics

The password `token` is a string that was associated with the user when they performed a `newPassword` operation. The user's password is updated to `password`. The password `token` is discarded.

5.4.4 Example Exchange

Request:

`http://???/completeNewPassword?token=4312rh92gp583h3295gh3t42qger2`

Response:

```
{
  "status": "okay",
}
```

5.5 authenticate request

5.5.1 Parameters

username: A string

password: A string

5.5.2 Response fields

token: A string

5.5.3 Semantics

If the `username` and `password` match, an authentication `token` that the use can use to authenticate with the Game and Trade APIs is returned to the user.

The authentication token is stored in the database entry for the user and is discarded if the use issues another `authenticate` operation or after `AUTHENTICATION.TIME.LIMIT` hours.

5.5.4 Example Exchange

Request:

```
http://???/register?username=Yoda
                        &password=sillypassword
                        &yoda@starwars.ie
```

Response:

```
{
  "status": "okay",
}
```

5.6 version request

5.6.1 Parameters

None

5.6.2 Response fields

major: The major version of the API.

minor: The minor version of the API.

5.6.3 Semantics

API version information

5.6.4 Example Exchange

Request: `http://??/version`

Response:

```
{
  "success":true,
  "major":0,
  "minor":2
}
```

6 Trade API

TBD.

7 Game API

The Game API is implemented on a bidirectional, stream-oriented connection³ over which messages are transferred. Each message has a **name** and a **content**.

1. A message's name is a string that identifies the **category** of message.
2. A message's content is a JSON object⁴.

7.1 Connection Handshake

When a connection is established⁵ the user must send a **start** message and *must not send any further message until they receive an **accepted** message from the server*. If the sever returns a **rejected** messages then the user should close the connection⁶.

7.1.1 start message

The JSON object for a **start** message has the following fields:

username: The user's name.
token: The user's authentication token.

For example:

```
{
  "username": "Hans Solo",
  "token": "98786vs8g5bsg875w6g57gdg"
}
```

³The connection must support the transfer of JSON objects belonging to different categories. For example, `engine.io.protocol` over TCP is a suitable protocol.

⁴Primitive JSON types and arrays cannot be used as a message's content.

⁵`engine.io.protocol` has its own handshake protocol used when a TCP connection is established.

⁶Depending on the semantics of the underlying stream-oriented connection, the server may also need to close the connection after sending the **rejected** message.

7.1.2 accepted message

The JSON object for an **accepted** message has the following fields:

timestamp: Unix timestamp in milliseconds.

major: The major version of the API.

minor: The minor version of the API.

position: A **3-D Coordinates** object.

orientation: A **3-D Coordinates** object.

For example:

```
{
  "timestamp": 368389679893479,
  "major": 0,
  "minor": 2
  "position": {
    "x": 42363.5374374,
    "y": 3947394796.215,
    "z": 846.26732
  },
  "orientation": {
    "x": 1.457,
    "y": 0.525,
    "z": 0.2546
  }
}
```

7.1.3 rejected message

A **rejected** message is a JSON **Reason** object

For example:

```
{
  "code": 106,
  "reasonText": "Expired Token"
}
```

7.1.4 disconnect message

If a user or the server wishes to close a connection they send a **disconnect** message. On receipt of a **disconnect** message, the server or user must close the connection⁷.

For example:

⁷Depending on the semantics of the underlying stream-oriented connection, the entity sending the **disconnect** message may also need to close the connection.

```
{  
  "code":0,  
  "reasonText":"Game over"  
}
```

7.2 Chatting

In general, a **chat** message is sent from an **originator** to the server and from the server to the **recipient(s)**. However, it is also possible for a server⁸ to send a **chat** message to recipients. The server does not acknowledge **chat** messages unless there is an error, in which case, the server sends a **chatError** message to the originator.

7.2.1 chat message

The JSON object for a **chat** message has the following fields:

timestamp: Unix timestamp in milliseconds.
originator: The originator's name.
recipient: An array of recipient names.
text: The chat text.

For example:

```
{
  "timestamp":368389679893492,
  "originator":"Master Yoda",
  "recipient":["Hans Solo","r2d2"],
  "text":"Welcome to Dagobah"
}
```

If the list of recipient names is empty, then the message is sent to all players, for example:

```
{
  "timestamp":368389679893492,
  "originator":"Master Yoda",
  "recipient":[],
  "text":"May the force be with you"
}
```

7.2.2 chatError message

The JSON object for a **chatError** message has the following fields:

error: A **Reason** object.
original: A copy of the original chat message.

For example:

⁸The server will need to have a unique name.


```
{
  "error":{
    "code":107,
    "reasonText":"Sender not originator"
  },
  "original": {
    "timestamp":368389679893492,
    "originator":"Master Yoda",
    "recipient":["Hans Solo","r2d2"],
    "text":"Welcome to Dagobah"
  }
}
```

7.3 Other Players

Once a connection has been accepted, the server will send **otherPlayers** messages to each user to update their list of visible users.

The JSON object for an **otherPlayers** message has the following fields:

`players`: An array of Player's Details objects.

For example:

```
{
  "players": [
    {
      "username": "Master Yoda",
      "ship": "astratis_v1",
      "position": {
        "x": 626246,
        "y": 23526.2664,
        "z": 25.125
      },
      "orientation": {
        "x": 0.2,
        "y": 1.4,
        "z": 0
      }
    },
    {
      "username": "Hans Solo",
      "ship": "Millennium Falcon",
      "position": {
        "x": 234567,
        "y": 2222.2664,
        "z": 25.125
      },
      "orientation": {
        "x": 0.7,
        "y": 1.4,
        "z": 1
      }
    }
  ]
}
```

7.4 Moving

Once a connection has been accepted, users may send **move** messages to the server to report their current position. These **move** message will be forwarded to other users as appropriate.

move messages are **not** acknowledged. However, a server will return a **moveError** message if there is a problem, e.g., a user tries to move to an impossible location. After receiving a **moveError** message, a user may send one of two messages:

1. A **disconnect** message to terminate the connection.
2. A **moveSync** message to resynchronise with the server, i.e., after sending a **moveError** message, a server will discard all **move** messages from the user until it receives a **moveSync** message. After receiving a **moveSync** message, the server will start processing **move** messages as normal.

7.4.1 move message

The JSON object for an **move** message has the following fields:

timestamp: A Unix timestamp in milliseconds.

username: The player's name.

position: A **3-D Coordinates** object.

orientation: A **3-D Coordinates** object.

For example:

```
{
  "timestamp":36838967347821
  "username":"Hans Solo",
  "position": {
    "x": 626246,
    "y": 23526.2664,
    "z": 25.125
  },
  "orientation": {
    "x":0.2,
    "y":1.4,
    "z":0
  }
}
```

7.4.2 moveError message

The JSON object for an **moveError** message has the following fields:

error: A **Reason** object.

original: A copy of the original **move** message.

position: The **3-D Coordinates** of the new position of the user.

orientation: The **3-D Coordinates** of the new orientation of the user.

For example:

```
{
  "error":{
    "code":108,
    "reasonText":"You have gone where no one has gone before"
  },
  "original": {
    "timestamp":36838967347821
    "username":"Hans Solo",
    "position": {
      "x": 626246,
      "y": 23526.2664,
      "z": 25.125
    },
    "orientation": {
      "x":0.2,
      "y":1.4,
      "z":0
    }
  },
  "position": {
    "x": 626222,
    "y": 23300,
    "z": 49
  },
  "orientation": {
    "x":0.2,
    "y":1.4,
    "z":0
  }
}
```

7.4.3 moveSync message

The JSON object for an **moveSync** message has the following fields:

timestamp: A Unix timestamp in milliseconds.

username: The player's name.

For example:

```
{  
  "timestamp":36838967352821  
  "username":"Hans Solo"  
}
```

7.5 Inventory

Once a connection has been accepted, the user and server may send **inventory** messages.

Deferred