Communications Protocol Collaborative Spreadsheet

Richard Timpson Jabrail Ahmed Tyler Brewster Conner Grimes
Zack Muhlestein Peter Forsling

March 6, 2019

Contents

1 Introduction

The following document outlines a communications protocol for a client-server multi-user spreadsheet desktop application. The following protocol gives no information for either technologies used to implement the application on the client or server side, nor any details as to how the applications should be implemented. The protocol is split into two aspects, logging into the server and choosing a spreadsheet to edit, and editing a spreadsheet. All of the data transfer between client and server will happen using TCP and Sockets, and the data sent back and forth will be JSON.

1.1 Overview and Technologies

Put some more specific information here about the technologies

1.2 Connection Overview

Give an overview about the connection

1.3 Editing Overview

Give an overview about editing

2 Connection

2.1 UML Diagram

Figure 1 shows what the initial connection of the client to the server looks like as a process. The initial connection is broken up into two different phases; logging into the server and choosing a spreadsheet to edit.

2.2 Socket Connection

Before a user logs into the server, there must be an initial connection from the client to server so that they can begin communicating back and forth with each other. The connection will be made

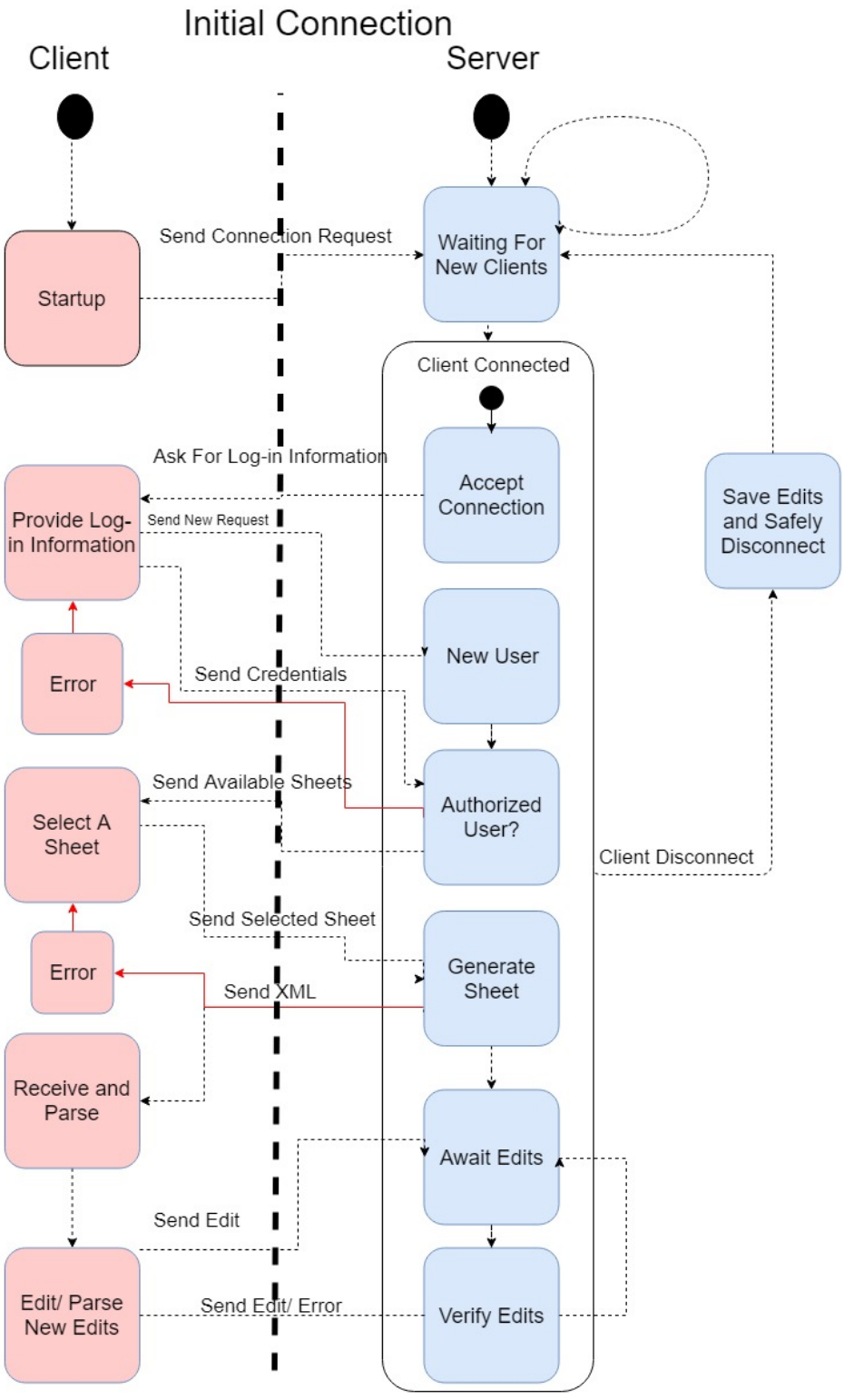

Figure 1: The UML diagram for the initial connection

using standard networking practices with a socket connection built over TCP/IP. On the initial start of the server, it should begin listening for socket connections at whatever IP address and port are sufficient. It is important that the client knows what specific IP address and port number to connect to. With this information, the client should complete the socket connection with the server. If there are any errors with the connection, TCP should report them and both the client and server can handle the exceptions accordingly. If the connection is successful, the socket should stay connected to begin the transfer of data.

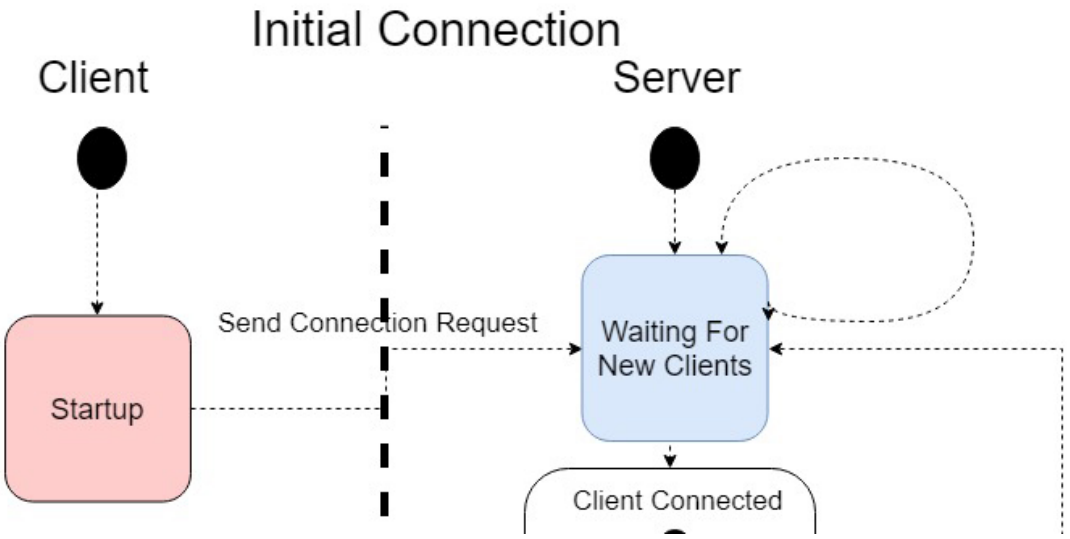

Figure 2: The socket connection in 1

2.3 Loggin In

Once the connection has been made, the server will expect the client, at some point in time, to send the information for either logging in, or creating a new user. This data will be sent as a JSON string terminated by a “n” character, as was discussed in the introduction (need to make sure to talk about our data representation in the beginning of the document). Because the user can either log in or create a new user, there will be two different types of strings. For logging in, it will be formatted like so.

```
{
    "name": "the users name",
    "password": "the users password",
}
```

If the client wants to create a new user, the string will formatted as such.

```
{
    "new_user": "true",
    "name": "the users name",
    "password": "the users password",
}
```

It is the responsibility of the server to parse either string and perform the correct functionality based on that string, such as saving or checking the username and password. The server then needs to send a response back to the client, letting them know if the client successfully logged in or created a new user. If for any reason the server could not validate the credentials, it should send an error message back to the client so that it knows that it failed to connect. The format for the success message is as follows...

```
{
    "success": "a boolean value that is true if successful, false if not",
    "message": "a message to the client letting them know what the error was
        if there was one, or if the user was successfully verified or created"
}
```

If the client failed to log in, the server will expect another login message from the client, and will continue in this state until it receives valid login credentials.

2.4 Picking a Spreadsheet

If the client did send valid login credentials, immediately after sending a success message, the server will send a JSON string containing an array of objects that contain information of the spreadsheets on that server that the user can access. The following string represents one valid spreadsheet object.

```
{
    "name": "the spreadsheets name",
    "id": "the spreadsheet id",
    "last_edit_date": "a date time  string in the format dd-mm-yyyy hh:mm:ss",
}
```

The client then needs to send back to the server information about the spreadsheet it wants to

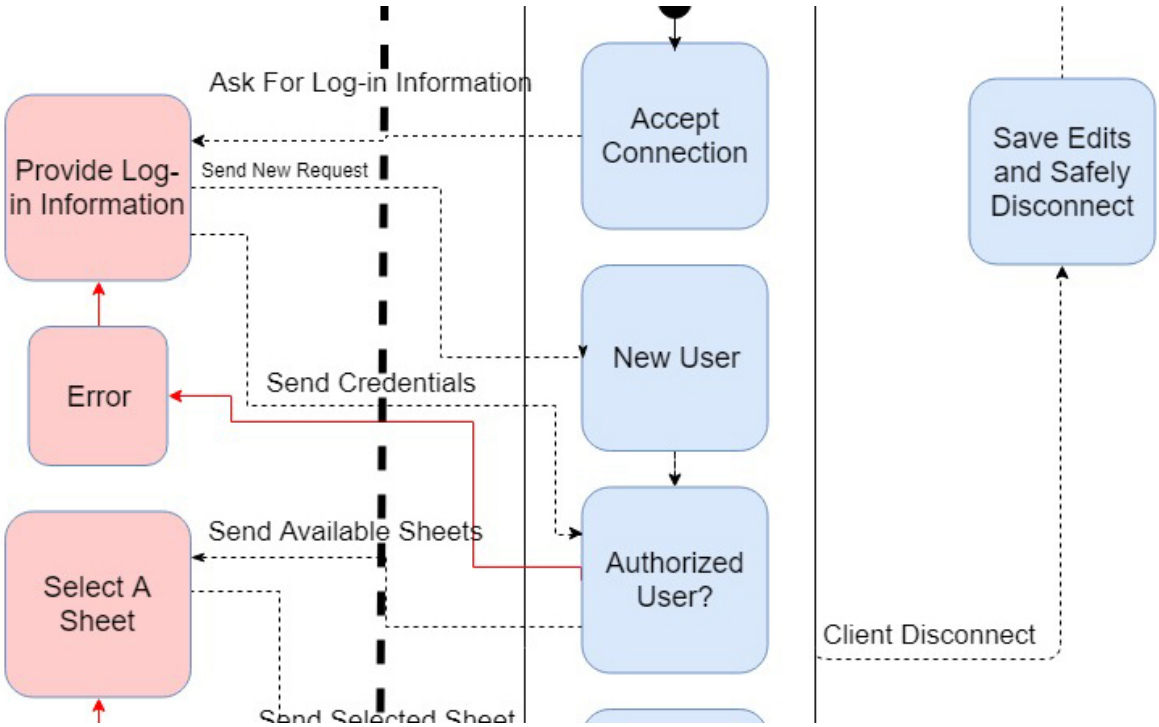

Figure 3: Logging in in figure 1

select. As with logging in, the client can send two options for spreadsheet selection; either picking an existing spreadsheet or creating a new one. If the client wants to create a new spreadsheet, it need only send the name of the spreadsheet that it wants to create, like so.

```
{
    "name": "name of the spreadsheet",
}
```

If it wants to pick an existing spreadsheet, it should send the id of the spreadsheet that it would like.

```
{
    "name": "name of the spreadsheet",
}
```

Again the server needs to send a message back to the client letting it know if it was successfully able to validate a spreadsheet for the client to edit. The JSON string should be the same as it is with logging in.

```
{
    "success": "a boolean value that is true if successful, false if not",
```

```
3            "message": "a message to the client letting them know what the error was
                 if there was one, or if the user was successfully verified or created"
4        }
```

Once the client has successfully chosen a spreadsheet to edit, the editing process will begin with the server sending all of the cells containing values as edits.

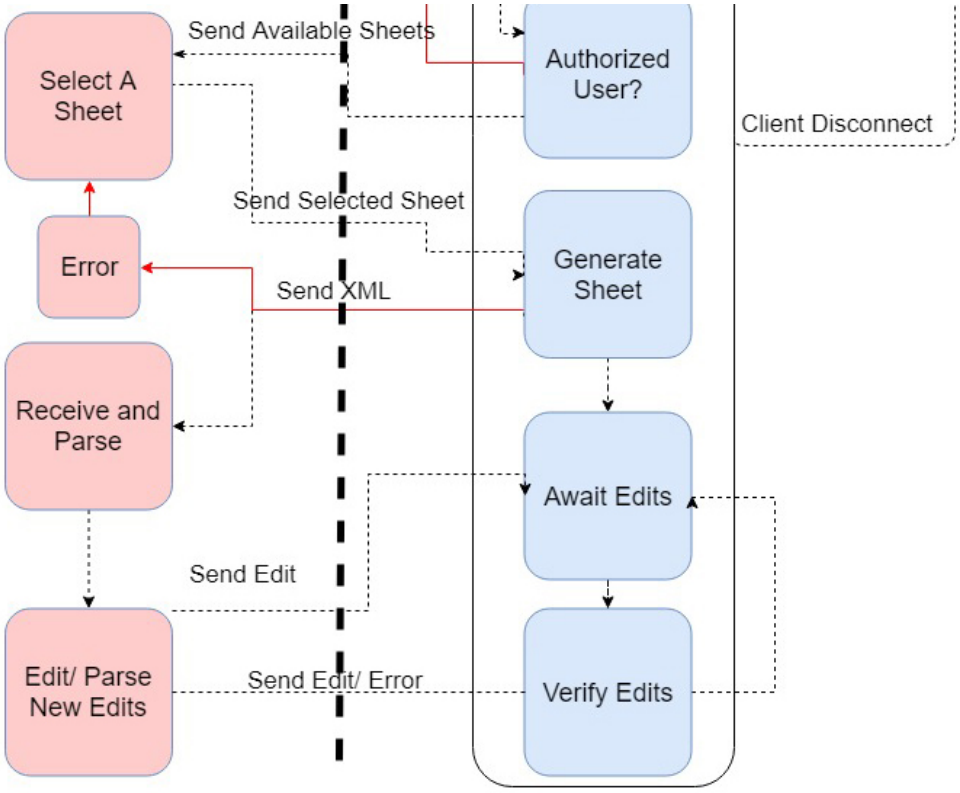

Figure 4: Logging in in figure 1

3 Editing

3.1 Overview

The basic process for editing a cell is as follows: the client enters a value into a cell and sends the cell's information to the server. The server then checks to see if what the user has entered is a valid edit or not. After the server authorizes that the user has submitted a valid edit, it sends that edit to all clients and records that edit. Figure 5 shows editing the spreadsheet as a process of communications

between the client and the server.

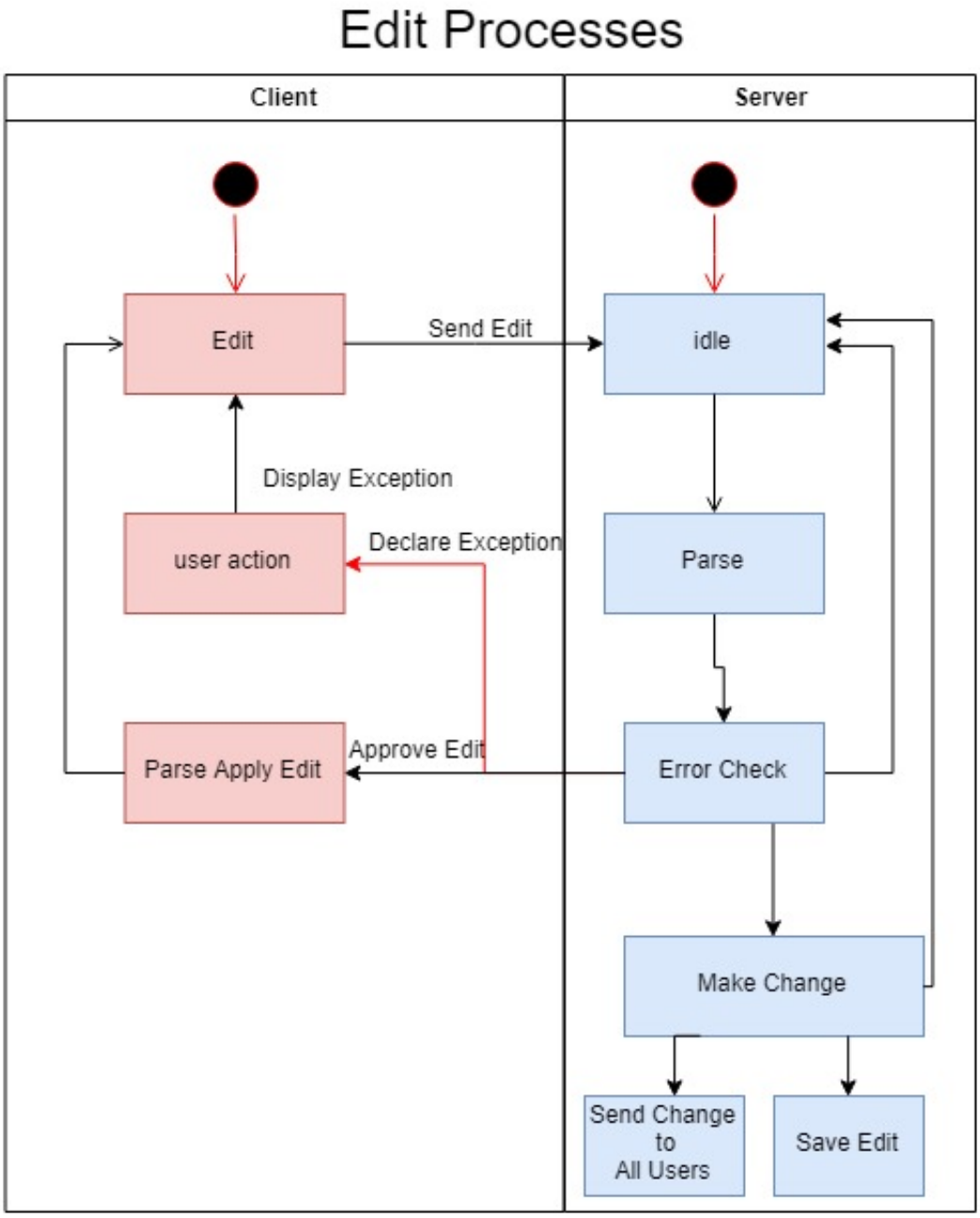

Figure 5: The editing diagram

3.2 Generating a saved spreadsheet

The server first sends the function code 0 followed by a JSON array containing all of the active users. Next, the server sends all of the non-empty cells as basic edits, following which it will send the locations of all other users for the client to display.

This is an example of the start up message from the server to the client:

```
0
[
    {
        "cell": "two characters i.e a1",
        "users_active": ["userID1", "userID2", etc.],
        "change": "",
        "value": "",
    },
    {
        "cell": "two characters i.e a1",
        "users_active": ["userID1", "userID2", etc.],
        "change": "",
        "value": "",
    },
]
```

3.3 Basic Edit

After the client makes an edit locally it will send that edit to the server. The server will then validate the edit before sending that edit to all connected clients. If the edit is invalid it will say “formula error” or send it back as an exception. The server will also record this edit if it is not an exception. An example of a basic edit message from the client to the server:

```
1
{
    "user_name": "user’s name",
    "cell": "A1",
    "change": "=5+5",
},
```

And this is an example of a server to client message for basic edits:

```
2
{
    "cell": "A1",
    "users_active": "",
    "change": "=5+5",
    "value": "10",
},
```

3.4 Undo

After an undo option is called, the server will go back one edit chronologically. This will NOT create a new edit. The latest edit will be deleted from record. This new change will then show up on the clients.

- When undo is called on an unedited spreadsheet, the server doesn't need to do anything.
- A user should be able to call undo on a spreadsheet enough times to get it back to its original state (empty spreadsheet) from creation, regardless of the sheet being closed between creation and now. It is expected the server handles this functionality.

The following is an example of an undo request from client to the server:

```
2
{
    "user_name": "users name",
    "cell": "A1",
    "change": ""// can also be seen as null,
},
```

3.5 Revert

When the client calls for revert, the server will change the contents of the cell to its earlier contents. This new edit will be counted as a new edit. This change will then be evaluated and sent out as a normal edit.

- When revert is called on an unedited cell, the server doesn't need to do anything.
- A user should be able to call revert on a cell enough times to get it back to its original state (empty cell) from creation, regardless of the sheet being closed between creation and now. It is expected the server handles this functionality.

When to people are editing a cell at the same time, the first edit sent will be the first edit parsed and the second edit will override the first edit. The following is an example of a revert request from the client to the server:

```
3
{
    "user_name": "users name",
    "cell": "A1",
    "change": ""// can also be seen as null,
},
```

3.6 Errors(Invalid Formulas)

When invalid formula error occurs we will give a notification in the cell block that says "Formula Error". The following is an example message from the server to the clients:

```
3
{
    "cell": "A1",
    "users_active": "",
    "change": "A1/+5"// can also be seen as null,
    "value": "Formula Error: Formatting",
},
```

3.7 Exceptions(circular dependencies, divide by 0)

Exceptions do not count as edits and are not sent to all users. When divide by zero error occurs we give an exception to the user that entered the divide by zero exception. When a circular dependency occurs we must give a exception to the user that entered the circular dependency to notify them of the exception. The following is an example message from the server to the client:

```
4
{
    "cell": "A1",
    "users_active": "",
    "change": "Exception"// can also be seen as null,
    "value": "",
},
```

3.8 Saving

When the client decides to save the current state of the spreadsheet the client should then send the server a request to save so the server can authenticate that there are no exceptions and then send the current state of the spreadsheet and save using a JSON objects. Also the spreadsheet should be saved so that when the server unexpectedly crashes and make sure that the last processed edit made by any user was saved.

```
5
{
    "user_name": "users name",
    "cell": "", // can also be seen as null,
    "change": "" // can also be seen as null,
},
```

3.9 Data Transfer

The following is the format of messages for our protocol. It follows the Json string format to simplify communication. No compression is expected for ease of communication between languages. The following shows the expected format and fields with what the function number should be for different expected results.

3.9.1 Client to Server

Function Code	Course of Action	Value of Change
0	Editor Location Changed	0
1	Normal Cell Change	string
2	Undo Request	0
3	Revert Request	0
4	Client Changes Sheet	0
5	Save	0

3.9.2 Server to Client

Function Code	Course of Action	Value of users active	Value of change	Value of value
0	Start Setup	Array of current users	Null	Null
2	Edit Cell	Null	String input	Calculated Value
3	Error	Null	String input	Formula Error
4	Exception	Null	"EXCEPTION"	0
5	Add/Remove Editor	Array of current users	Null	Null

Function Code	Course of Action	Cell	UserName
1	Change where someone is	Cell Number	User Number

4 Conclusion