Protocol Jakkpot

Andrew Porter, Jacob Smith, Kimball Johnston,
Kobe Riddle, Preston Powell, Tyler Jones

Table of Contents

0 Introduction

This document will detail the communication protocol between the server and the clients of a multi-user spreadsheet. This spreadsheet's basic functionality is that of the single-user spreadsheet developed in CS3500. It consists of a set of cells, each of which contains contents that can be a plain string, a number, or a formula (no circular dependencies are permitted). A cell's value is determined by its contents. If a cell's contents are a string or a number, then its value is its contents. If a cell's contents are a formula, then its value is either the numerical result of that formula or a formula error (if the formula referenced a cell whose value is not a number or involved division by zero).

The server and the client will communicate via sockets using the TCP protocol. During the initial handshake between server and client, preliminary information is exchanged and the client selects a spreadsheet to open (or opens a new spreadsheet). At this point, the client is free to begin selecting cells and making edit requests, and the client will see any selections and edits performed by other clients on the spreadsheet. The server will save all edits to a spreadsheet, and clients may undo and revert previous changes to the spreadsheet. Below, this document contains detailed information on the handshake, opening files, selecting cells, changing cells' contents, undoing changes, and all other procedures outlined below.

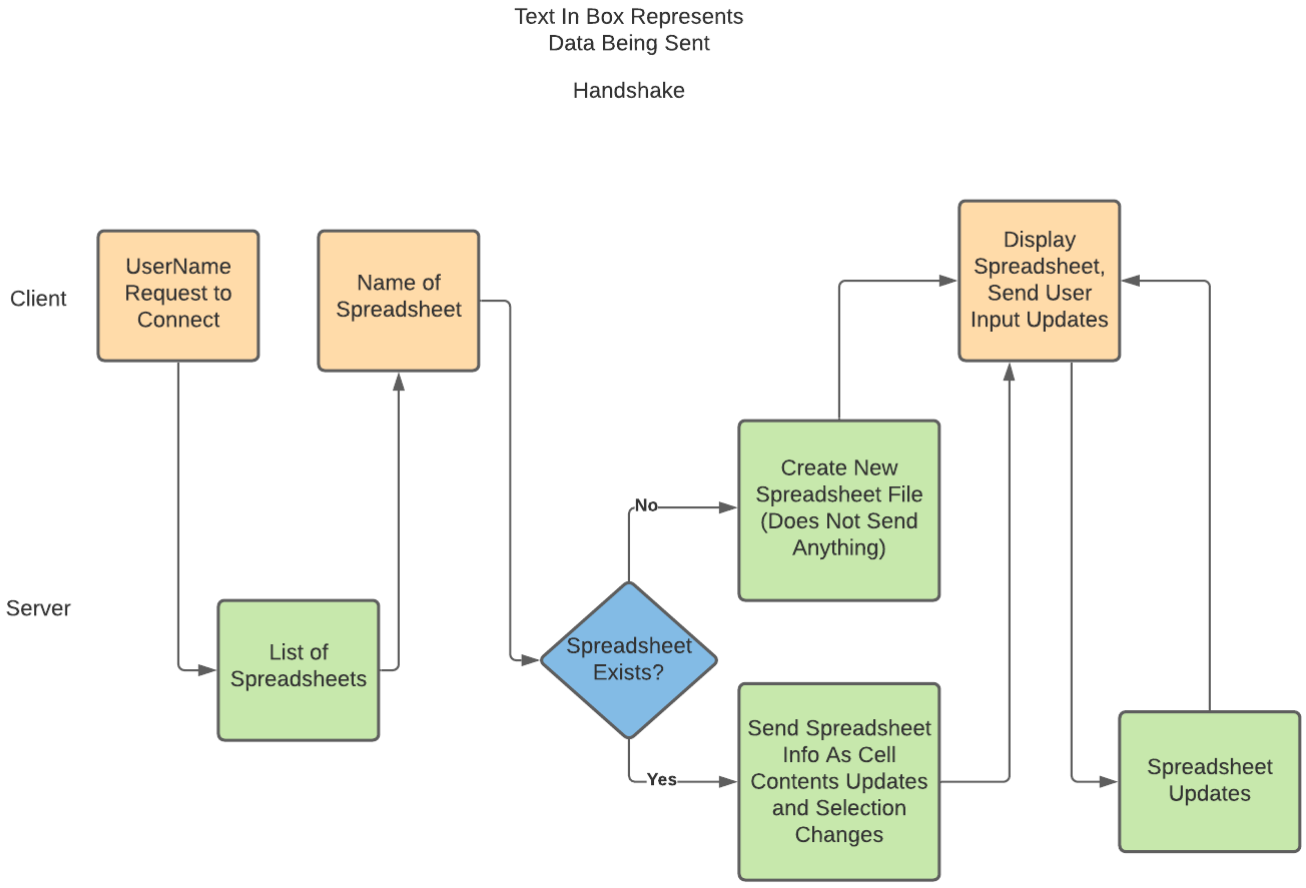

Figure 0: Overall Communication Flow

1 Initial Connection (Handshake) And Closing

1.1 Connecting to the Server

First, the client opens a TCP socket to the server on port 1100 using UTF-8. The client will then request access to the spreadsheets by sending a message containing a userName as a string to the server, ending with a newline character.

1.2 Receiving List of Available Spreadsheets

Upon receiving the message, the server will send an alphabetically ordered list of strings of the existing file names on the server. Each file name will be separated with the newline character "\n." After the final file name, the server will send an additional newline character. When two newline characters are encountered together, the client will know that all available spreadsheets will have been sent to the client.

Note that clients can only know about spreadsheets that existed when they first connected. If they wish to view any spreadsheets that were created after their connection, they must restart the handshake to view new spreadsheets. Sending the name of a spreadsheet that was created after the client has received the list of spreadsheets will still open the created spreadsheet and load any edits made up until that point.

1.3 Creating and/or Opening a Spreadsheet

The client then sends a filename to the server followed by a newline character. If a filename does not exist on the server, a new file of that name is created. Names are required to be unique and cannot contain new line characters themselves.

1.4 Entering the Editing and Updating Loop

As soon as the filename is sent, the client should begin listening for cell update and cell selection changes from the server. See the section on basic editing for the format of these messages. The server should then send the contents of every non-empty cell as a cellUpdate message and should send a cellSelected message for each other client on the spreadsheet indicating which cell they are currently selecting. The final message from the server for the handshake will be the unique client ID number. The ID will be an integer followed by a newline and clients should not make any edit requests before receiving their ID. Edits by any users on the spreadsheet made during the time the server is sending the initial spreadsheet data to a new client will not be processed and instead will be queued up to send all clients afterwards. As a result, the server will not apply any changes to the spreadsheet while a new client is being sent the initial spreadsheet data.

1.5 Closing a spreadsheet

To close a spreadsheet, the client simply must disconnect from the server by closing its socket. When the server detects that its socket connection to a client is closed, it notifies all clients that this client has disconnected by sending a client disconnected message.

disconnected	`{messageType: "disconnected", user: "<ID#ofUser>"}`

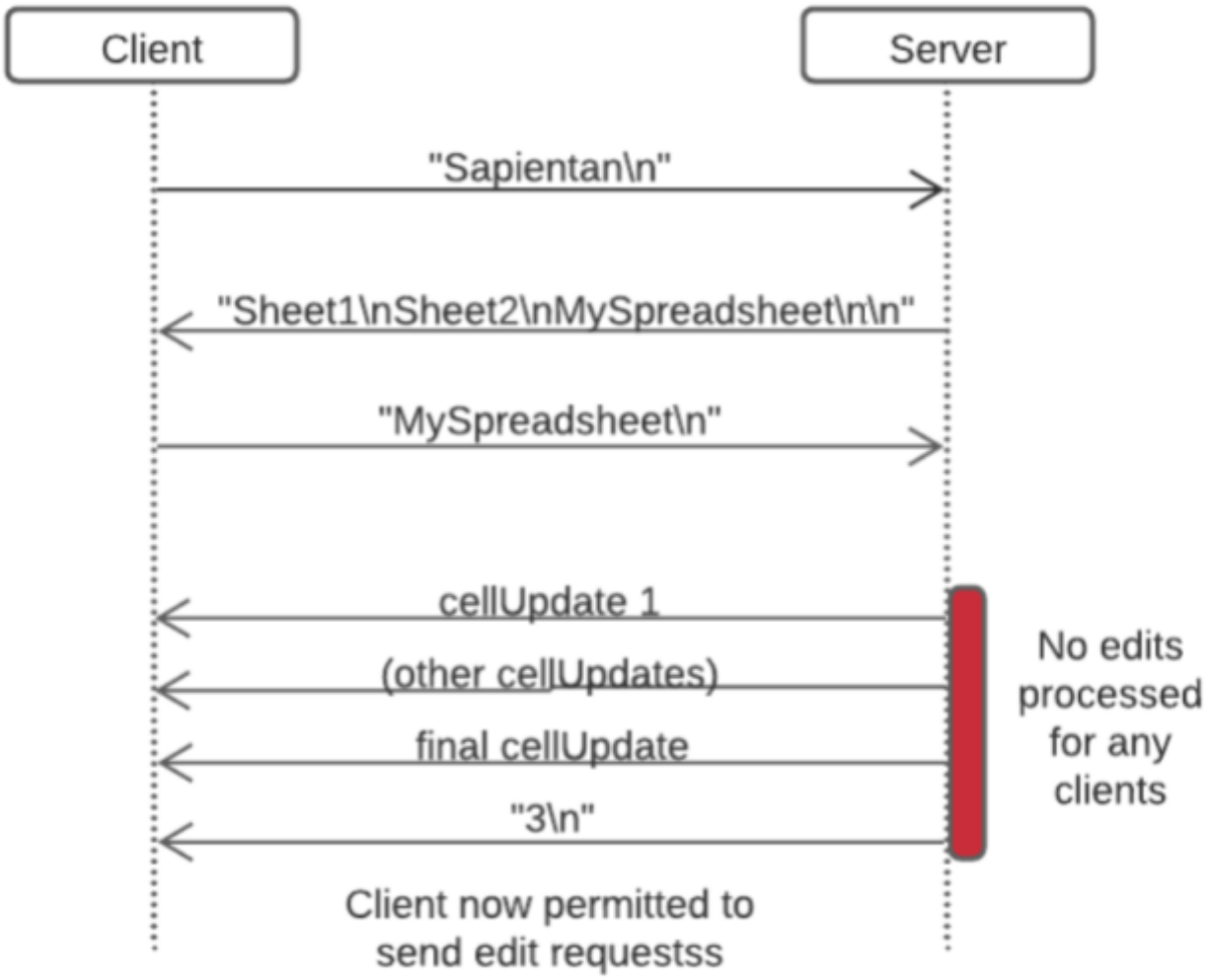

Figure 2: Example Handshake

2 Request and Server Message JSON

Requests are the way the client sends requests to alter cells, etc., while server messages are the way the server informs the clients about changes to the spreadsheet. Both directions of communication are in JSON format and may utilize any of the fields below. All JSON messages between client and server are terminated with a newline character.

Possible request fields:

- *requestType
- cellName
- Contents

Possible server message fields:

- *messageType
- cellName
- contents
- message
- selector
- selectorName

*required

Based on the type of request/message, additional fields may be required, but any non-required fields (including fields that are not part of any object request) must be ignored by the server. Additional clarification on which fields are conditionally required is detailed in the information for the relevant request type.

3 Editing Cells

Once a client has received all the cells in a spreadsheet (indicated by receiving their client ID), it can begin to select and edit cells in that spreadsheet. When a client selects a cell, it sends a message to notify the server of the selection. The server then sends a message to all other clients notifying them of the selection. When a client desires to edit a cell, it sends a message to the server requesting that the edit be made. The server checks the change for validity according to the rules of the spreadsheet of CS3500 (no circular dependencies, no invalid formulas, etc.). If the change is valid, it sends a message to each client connected (including the client which sent the edit message) containing the cell's new contents; otherwise, it sends an Invalid Request Error to the client that requested the edit including the name of the cell that will not be updated.

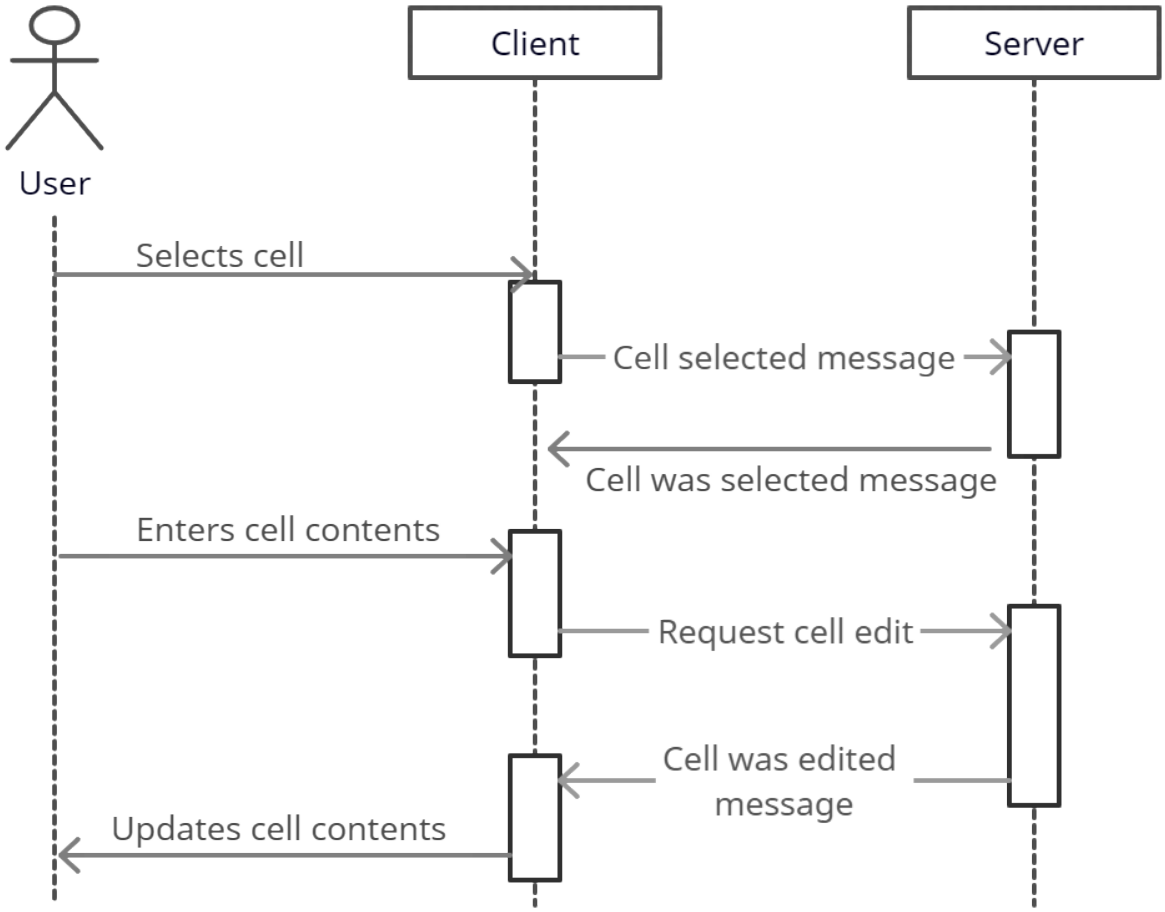

Figure 3. Basic communications between clients and the server

3.1 Client to server communication:

While editing a spreadsheet, clients use two basic requests: "select" and "edit". A client must first send a "select" request for a given cell, before requesting an edit of the same cell. Both of these requests are shown below.

Request	Format (JSON)
Select cell	`{ requestType: "selectCell", cellName: <cell name> }`
Request cell edit	`{ requestType: "editCell", cellName: <cell name>, contents: <desired contents> }`

Table 3.1. Client requests for editing

3.1.1 Select

When a client selects a cell, it needs to send a message to the server notifying the server that the cell has been selected. The client does not need permission from the server to select a cell and can select any cell at any time. The message format for selecting a cell is shown in Table 3.1. The cellName parameter should be set to the cell's address. Thus, a client wishing to select cell A1 would send the server the request `{requestType: "selectCell", cellName: "A1"}`.

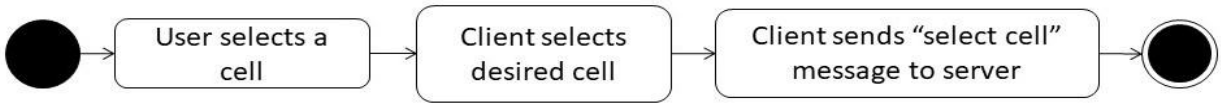

Figure 3.1.1 Procedure for a client selecting a cell

3.1.2 Edit

A client sends an "edit" message to the server when it wants to change a cell's contents. The message format for requesting an edit to a cell is shown in Table 3.1. The cellName parameter should be set to the cell's address, and the "contents" parameter should be set to the desired new contents for the cell. For instance, a client wishing to set the contents of cell A1 to 42 would send the server the request `{requestType: "editCell", cellName: "A1", contents: "42"}`.

The client must first send a "select" message and then an edit message for any given cell (see the section on the server's messages below). After sending a request for a cell's contents to be changed, a client must wait to change the contents of its local copy of the cell until the server sends it a "cellUpdated" message (see below). Clients can make requests to change a cell's contents but cannot directly change it themselves.

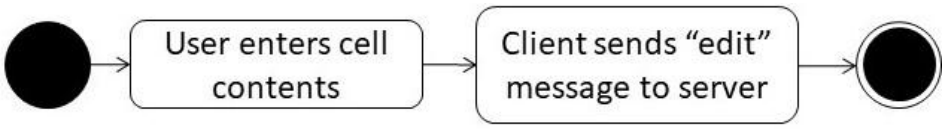

Figure 3.1.2 Procedure for a client to request an edit.

3.2 Server to client communication:

In response to the requests that a client sends when editing a spreadsheet, the server must send two basic messages of its own: a "selectCell" message and a "cellUpdated" message. The messages are shown below.

Message	Format (JSON)
Cell was selected	`{ messageType: "cellSelected", cellName: <cell name> selector: <ID # of selector>, selectorName: <name of selector> }`
Cell's contents changed	`{ messageType: "cellUpdated", cellName: <cell name>, contents: <contents> }`

Table 3.2. Server messages for basic editing

3.2.1 Selected

When the server receives a "selectCell" request from a client, the server notes which cell the client has selected and which spreadsheet the client is editing and then sends a message to all other clients using that spreadsheet to notify them of the selection change. The message's format is that of a JSON-serialized object with "messageType", "cellName", "selector", and "selectorName" parameters. The "messageType" parameter is set to "cellSelected", the "cellName" parameter is set to the cell's address, the "selectorName" parameter is set to the string name of the selector, and the "selector" parameter is set to the ID number of the client that made the selection (see section 1 for more information).

For example, if the server received the message `{requestType: "select", cellName: "A1"}` from client `0, with username Jerry` editing a spreadsheet called "Test", the spreadsheet would send the message `{messageType: "selected", cellName: "A1", selector: 0, selectorName: "Jerry"}` to all clients except client `0` using the spreadsheet "Test".

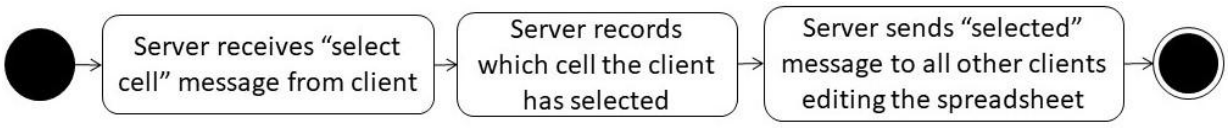

Figure 3. 2.1 Procedure for server when a selectCell request is made

3.2.2 Cell Update

When the server receives an edit message from a client whose cell name field does not match the cell that the client currently has selected, the server will ignore the edit request. Otherwise, the server will check whether the proposed edit is valid (according to the rules of the CS 3500 spreadsheet). If this check fails, the server does not implement the edit and instead sends the client that made the request an Invalid Request Error (see section 5 for more information).

If the edit passes the above check, then the server will update the cell's contents as requested in its copy of the spreadsheet and send a message to all clients (including the client that placed the request) informing them of the cell's new contents. The format of this message is shown in Table 3.2. The "cellName" parameter should be set to the cell's address, and the "contents" parameter should be set to the cell's new

contents. For instance, if the server received the message `{ requestType: "edit", cellName: "A1", contents: "42" }` from a client editing the spreadsheet "Test", then the server would send the message `{ messageType: "cell", cellName: "A1", contents: "42" }` to all clients using the "Test" spreadsheet.

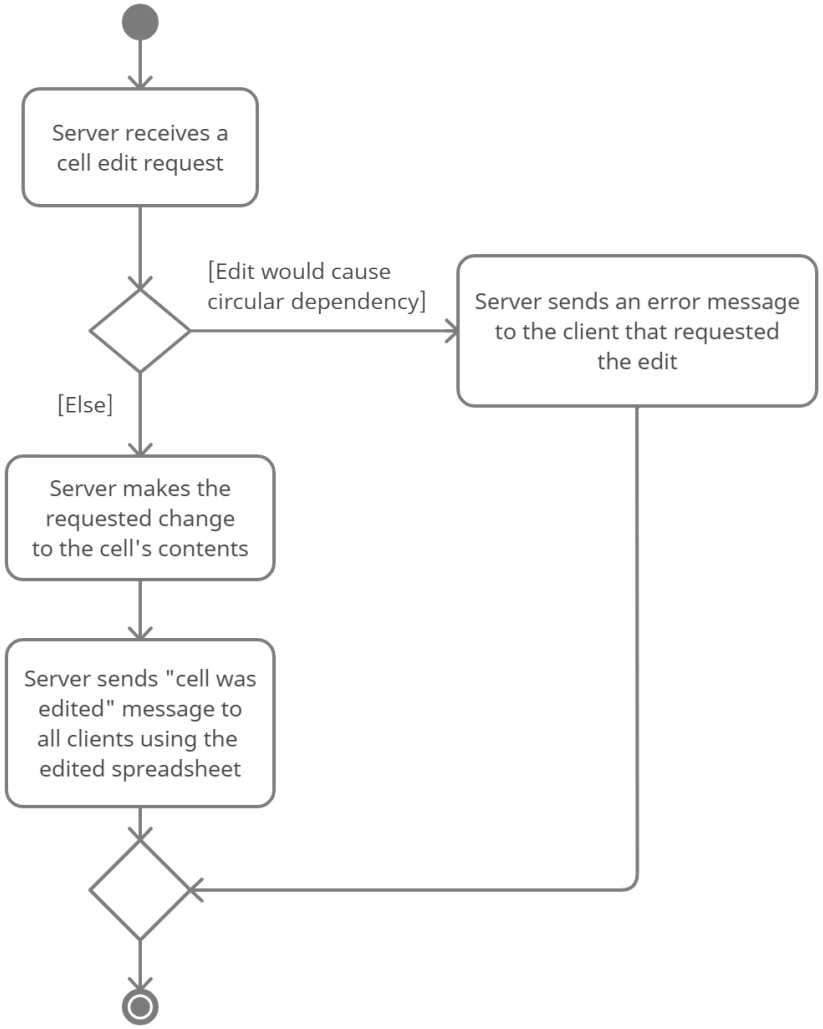

Figure 3.2.2 Procedure for when the server receives an edit request

3.2.3 Request Order

The server processes edit requests in the order that it receives them. Thus, if two clients both submit requests to edit the same cell, the server will process the first edit received and notify all clients of the change and then process the second edit received and notify all clients of the change, overwriting the first edit that was received. In addition, note that the spreadsheet persists all changes. Any change that it processes and does not reject as creating a circular dependency will be saved. Additionally, all changes will be remembered and can be undone. For more information about undoing, see section 4.1.

4 Undo and Revert

4.1 Undo

An undo request is sent to the server from the client in the format shown below. When the server receives an undo request it will reverse the latest edit made to the spreadsheet and, notably, this will not count as an additional edit to the spreadsheet (contrasting with revert). The client will then receive a normal update for the undone cell as by any other edit. An additional undo will have the effect of moving the spreadsheet back an additional edit. If no undos are possible the server will send the client an Invalid Request Error.

Message	Format (JSON)
Undo request:	`{"requestType": "undo"}`

4.2 Revert

The server will also support requests to revert cells. This includes the client sending a request to revert a cell back to the most recent edit to the cell. Notably, reverts will count as additional edits to the spreadsheet and thus can be undone. This type of request is known as a "revertCell" request. Examples for these requests are shown below.

Message	Format (JSON)
Revert request:	`{ "requestType": "revertCell", "cellName": "A1" }`

If the user wants to revert a change to cell A1, the client sends the following request: { "requestType" : "revertCell", "cellName" : "A1" }. As long as it is a valid edit according to the server, the server will update the contents of A1 and send a cellUpdate message to all clients as it would for a normal edit to a cell. If the client sends the request again, assuming it is valid, the server would then move the cell back one more time into the history of edits made for the cell.

For example, assume a cell's contents have been edited (in order of earliest to latest) to: 5, "Hello", and 3, with 3 being the current contents of the cell. The first revert request will revert the cell's contents back to "Hello", and the second revert request will revert the cell's contents back further to 5. Similarly, a third revert request will make the cell's contents blank again (as it was when the spreadsheet was made) and at that point no further reverts to the cell can be made and any attempt to do so will result in the server sending an Invalid Request Error. Note that because these count as additional edits to the spreadsheet (contrasting with undo), such that they can be undone.

4.3 Interactions between Revert and Undo

Again, reverts are considered edits, and thus can be undone. The following table is meant to give an example of how the undo and revert requests interact. Note that reverts are being undone and as there is an equal number of undo requests the final result of the table is the same as the original table and there can be no more undos or reverts.

Note that : $A_2 \leftarrow$ "Text" indicates sending a request to change the contents of A_2 to "Text"

Commands	Contents of A_2	Contents of A_3
Open sheet	blank	blank
$A_2 \leftarrow$ "Table"	"Table"	blank
$A_3 \leftarrow$ "=A_2"	"Table"	"=A_2"
$A_2 \leftarrow$ "Text"	"Text"	"=A_2"
Revert A_3	"Text"	blank
$A_2 \leftarrow$ "Data"	"Data"	blank
undo	"Text"	blank
undo	"Text"	"=A_2"
undo	"Table"	"=A_2"
Revert A_2	blank	"=A_2"
undo	"Table"	"=A_2"
Revert A_2	blank	"=A_2"
undo	"Table"	"=A_2"
Revert A_2	blank	"=A_2"
undo	"Table"	"=A_2"
undo	"Table"	blank
undo	blank	blank

5 Errors

Message	Format (JSON)
Invalid Request Error	`{ messageType: "requestError", cellName: <cell name>, message: <message> }`
Server Shutdown Error	`{ messageType: "serverError", message: <message> }`

When clients attempt to set cell contents, the server may need to send error messages to clients (see the sections 3 on editing and 4 on reverting for details on when the server may need to send error messages). Similar to server messages, these error messages will be in JSON format, with parameters "messageType" and "cellName". The "messageType" parameter will be "requestError", and "cellName" will be set to the cell's address.

Additionally, if an error forces the server to shutdown while clients are still connected, then before shutting down, the server will send a Server Shutdown Error message to all clients to notify them that the server is about to shutdown.

All errors also have a "message" parameter that can be used to provide additional information. This message should be suitable for display to a user.

Glossary of Requests/Messages

Request Types (Client → Server)	
Type	**Details**
(Handshake) Username	`"<username>\n"`
(Handshake) Filename	`"<filename>\n"`
editCell	`{requestType: "editCell", cellName: <cell name>, contents: <desired contents>}`
revertCell	`{requestType : "revertCell", cellName : <cellName}`
selectCell	`{requestType: "selectCell", cellName: <cell name>}`
undo	`{"requestType": "undo"}`

Message Types (Server → Client)	
Type	**Details**

FileName List
(Handshake)
	`"<file1>\n"`
	`"<file2>\n"`
	`"<file3>\n\n"`

cellUpdate	`{messageType: "cellUpdated", cellName: <cell name>, contents: <contents>}`
cellSelected	`{messageType: "cellSelected", cellName: <cell name> selector: <ID # of selector>, selectorName: <name of selector>}`
disconnected	`{messageType: "disconnected", user: "<ID#ofUser>"}`
error - Invalid Request	`{messageType: "requestError", message: <message>}`
error - Server Shutdown	`{messageType: "serverError", message: <message>}`

Changes made to the document

Where	What
On page 4 and in the glossary	“Disconnected” json changed as follows “requestType” → “messageType”
On page 4	Clarified that the server does not apply any received changes to the spreadsheet
On page 5	Fixed error in diagram where the list of spreadsheets to edit was not properly terminated
On page 13	Changed “name” to “cellName”
On page 13	Specified that error messages should be suitable for display to the user