# JJVA Network Protocol Specification

John Robe, Joshua Christensen, Vincent Cheng, Atul Sharma

March 2, 2017

# Table of Contents:

# Rationale

This protocol is designed to be an easy to implement, robust communications protocol for a hosted spreadsheet application. It requires that the client submit spreadsheet data free of malformed formulas, invalid cell names and circular dependencies. The protocol is designed under the assumption that network errors, such as desync and dropped packets, are exceptional cases, and the performance of the protocol in the general case should not suffer because of these exceptional cases. However, the protocol is also designed to be resilient in high latency, and high packet loss environments.

## Overview

**It is not necessary read this document in its entirety in order to implement this protocol. Critical sections are: Key Concepts, Message Format, the Client and Server diagrams, Notes on Key and Sequence # Assignment, and Notes on Message Processing. All other information is clarification and extra, helpful information.**

## Key Concepts:

- This protocol assumes that the client checks for malformed formulas, circular dependencies, and that it sends proper cell names.
- Spreadsheet calculations are done client side, to reduce latency when editing.
- This protocol implicitly states that every edit made to the sheet is automatically saved.
- Clients operate under the assumption that they are correct until proven otherwise.
- The server reserves the right to reject requested edits until it pushes them to other clients.
- If the server rejects an edit from a client, all following edits with the same key are implicitly rejected.
- The server at no point requires input from the client in order to function properly.
- If either the client or server receives a message with a tag it doesn't recognize, it can safely ignore the message. This allows users to define custom messages that don't break other functionality.

## Message Format:

Messages are curly bracketed, ordered, comma separated values formatted in the following fashion:

- The first character is an open curly bracket ('{')
- Immediately following the first character, there is a Message Tag that specifies the type of message, terminated with a comma. Message Tags are uppercase, alphanumeric words (not strings, as we define them below), with no whitespace and terminated with a comma.
- Following the Message Tag, there are comma separated parameters that differ depending on the Message Tag. The type and number of parameters differ, but they will always be comma separated, with no whitespace between the comma and the

start of the parameter, and between the end of the parameter and the terminating comma. These parameters are either Strings, or Ints:
    - **Strings**: A series of characters, formatted as a c++ string literal. In other words, all quotation marks must be preceded with an escape character, barring the starting and ending quotation marks. For further information, reference c++ documentation.
    - **Ints**: 32 bit integer values with no whitespace. Formatted such that, if the value were passed to atoi(), the function it would return the value of the integer.
- Following the last of the parameters, there is a closing curly bracket (‘}’)

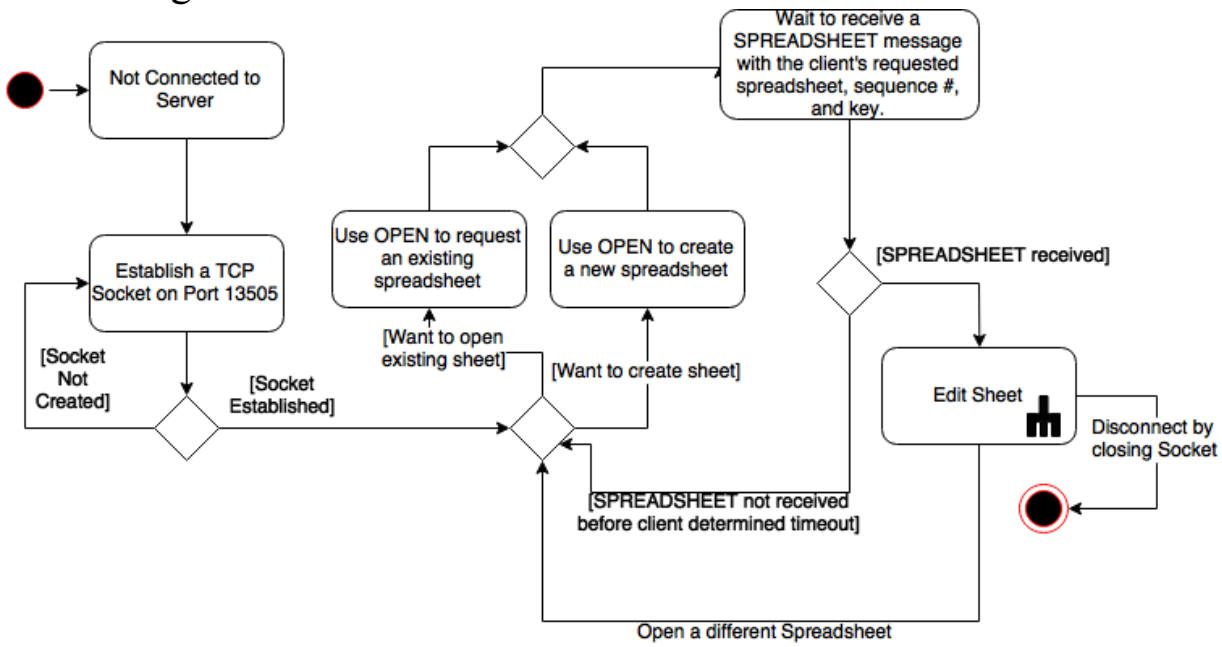
### Actions possible with this Protocol:

Following is a list of actions possible with this protocol. Following each action, the relevant message tag is included in parenthesis.

- Create a document on the server (OPEN)
- List all existing documents on the server (LISTSHEETS & SHEETLIST)
- Open some document, to begin editing (OPEN)
- Delete documents no longer in use (DELETE)
- Make a change to the server’s spreadsheet (PUSH)
- Receive updates when the server’s sheet is updated (UPDATE & SPREADSHEET)
- Undo the last performed action on the spreadsheet. (UNDO)

# Client Operations

## Connecting to a Server:

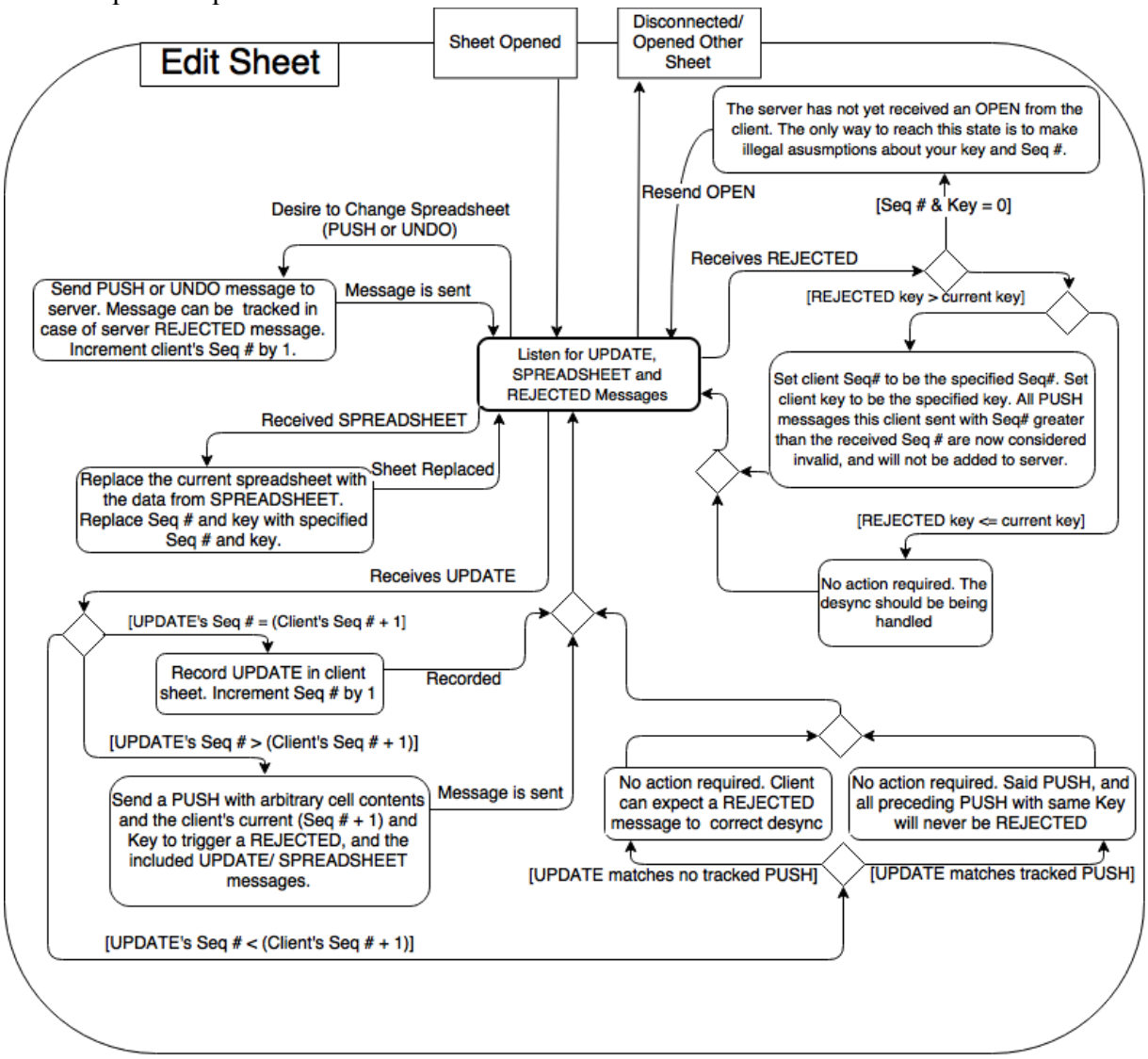


## Connecting to a Server: Detailed Steps:

- Establish a TCP Socket with the server on Port 13505.
- Once the client has a Socket, it should use the message OPEN to either open an existing spreadsheet, or create a new one. The OPEN command has a single string as a parameter. If the string corresponds to the name of an existing spreadsheet, that spreadsheet is “opened”, and all PUSH and UNDO messages are assumed by the server to be referring to said spreadsheet, until the server receives another OPEN command. The OPEN command should be responded to by the server with a SPREADSHEET command, which includes all the information a client needs to edit the Spreadsheet. See the detailed information for SPREADSHEET for more detail.
- If the client doesn’t receive a SPREADSHEET before some amount of time greater than one second, it should send another OPEN request, just in case either the OPEN or SPREADSHEET messages were dropped. Also, if this is the case, it’d be advisable to check the socket status, and ensure you are connected.
- Once the client have received a SPREADSHEET message, it is free to edit the spreadsheet. Refer to the Editing section for details. If the client wishes to edit a different spreadsheet, it can at any time simply open a different sheet with the OPEN command.
- In order to disconnect from the server, simply close the socket. No other action is needed, but beware that, unless the client has received confirmation that its edits have succeeded, the client cannot be sure that they will be saved.

# Client: Editing a Spreadsheet: Diagram:

Note: Seq # = Sequence #.



## Client: Editing a Spreadsheet: Detailed Steps:

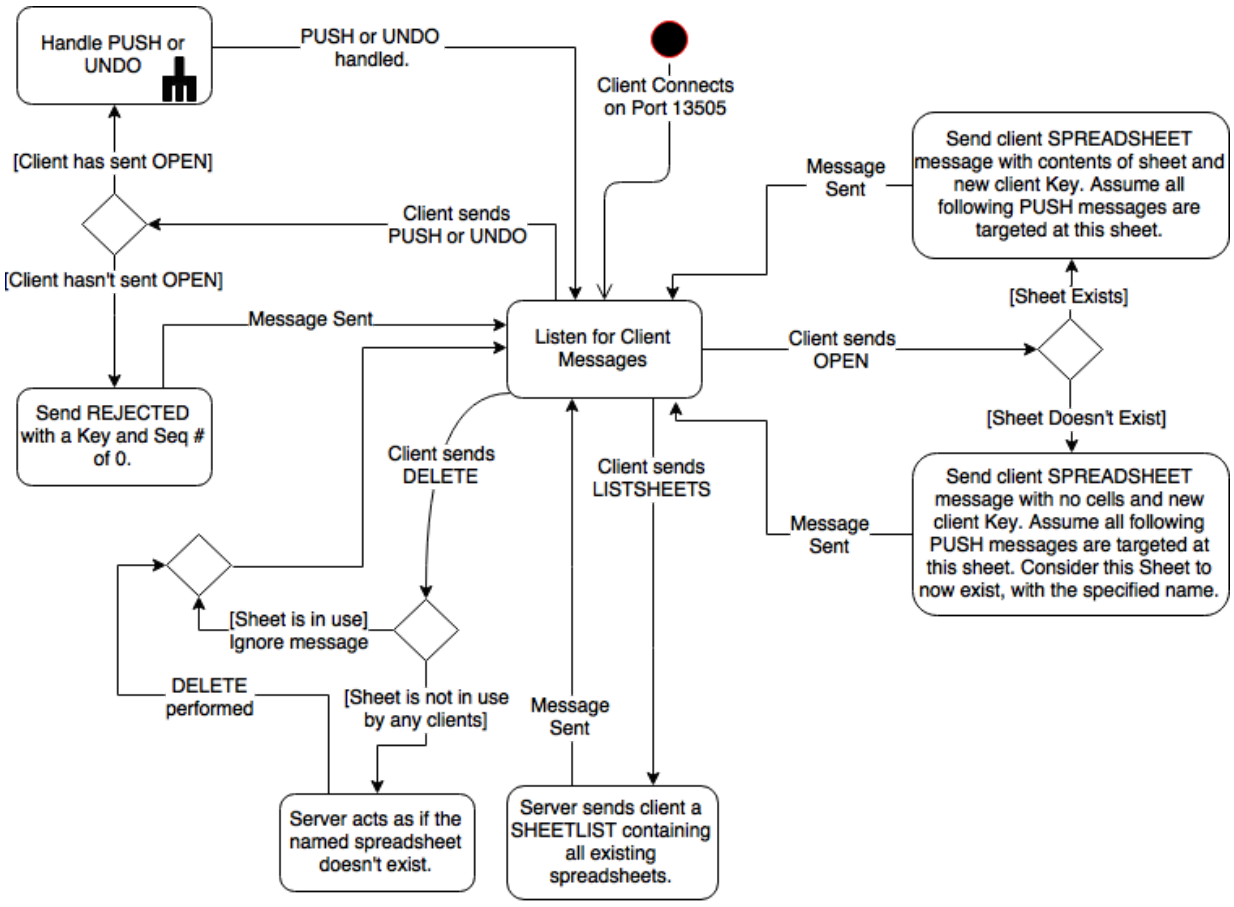
Editing a spreadsheet is event based. The default state should be listening for server messages, specifically UPDATE, SPREADSHEET and REJECTED messages. The following are cases detailing actions that the client should take if each of these messages are received, as well as a case for when the client wants to edit the spreadsheet.

- **TO UNDO:** In order to undo the most recently executed operation, send the server an UNDO message with the client's current sequence # + 1, and the client's current key. The client should then increment its sequence # by 1.
- **TO EDIT THE SHEET:** In the event the client desires to edit the sheet, it should ensure that its edit doesn't contain circular dependencies, its edit doesn't contain invalid formulas, and that its edit has a valid cell name (refer to the PUSH message details for further information). Then it should send a PUSH message to the server containing the cell name to edit, the desired contents of the cell, the client's current sequence # plus one, and the client's current key. Once it sends the PUSH, it should increment its sequence # by one. WARNING: The server reserves the right to respond to any PUSH with a REJECTED message, until such time that the server sends out an UPDATE containing the data from that PUSH. It is advisable to keep track of unconfirmed PUSH messages until such a time as they are confirmed added, in the case that the server rejects them and they need to be re-added. The client can keep track of what is wrong in the rejected message, and work through the edits to see if they can immediately resend a PUSH once updated, or not.
- **SPREADSHEET**: In the event the client receives a SPREADSHEET message, it should replace its displayed spreadsheet with that sheet, and update its Sequence # and client key to be the keys specified in the SPREADSHEET message.
- **REJECTED**: In the event the client receives a REJECTED message, it should do one of two things (checked in the order in which they appear here):
    - **If both the Key and Sequence # are equal to zero:** This is an error state that should never be received in a properly implemented client. It indicates that the server has not received an OPEN message from this client, but the client is attempting to send PUSH or UNDO operations to the server. The client should re-send an OPEN message to open a spreadsheet. Can also serve as a sort of server ping to measure latency for clients not yet editing a spreadsheet.
    - **If the key in the REJECTED message is greater than the current client key**: The client should consider all PUSH messages that it sent with that key, including and after the sequence # included in the message, to be invalid. These will never be added to the server sheet, unless resent under a different key and correct sequence #. The client should set its sequence # to be the one included in the REJECTED message, minus one. The client should set its key to be the one specified in the REJECTED message.
    - **If the key in the REJECTED message is equal to or less than the current client key:** These messages can be safely ignored. This REJECTED was sent as a response to a PUSH that should already be considered invalid.
- **UPDATE:** In the event the client receives an UPDATE message, it should do one of three things:

- **If the Sequence # in the UPDATE message is equal to the client's current sequence #, plus one:** The UPDATE should be recorded in the client's spreadsheet, and the client's current Sequence # should be incremented by 1. This is normal operation.
- **If the Sequence # in the UPDATE message is greater than the client's current sequence #, plus one:** The client should send a PUSH message with arbitrary cell contents, the client's current sequence # plus one, and the current client key. This will trigger a REJECTED message from the server, and cause the server to resend the messages the client missed.
- **If the Sequence # in the UPDATE message is less than the client's current sequence #, plus one:** In this case there are two possibilities. Neither requires client action, but they provide useful information the developers may find useful.
    - **If the UPDATE message matches a PUSH message that the client sent to the server:** From this, we can be assured that the server has added our PUSH message to its spreadsheet, and we no longer have to account for the fact that it may be rejected in the future. Additionally, if we receive confirmation that our PUSH message was added to the spreadsheet, we can be assured that all previous PUSH messages with the same key have also been added.
    - **If the UPDATE message does not match a PUSH message the client sent to the server:** From this, we can deduce that the client attempted to send a message when it wasn't up to date. This will be corrected with a REJECTED message.

# Server Operations:

## Server: Managing Clients: Diagram:



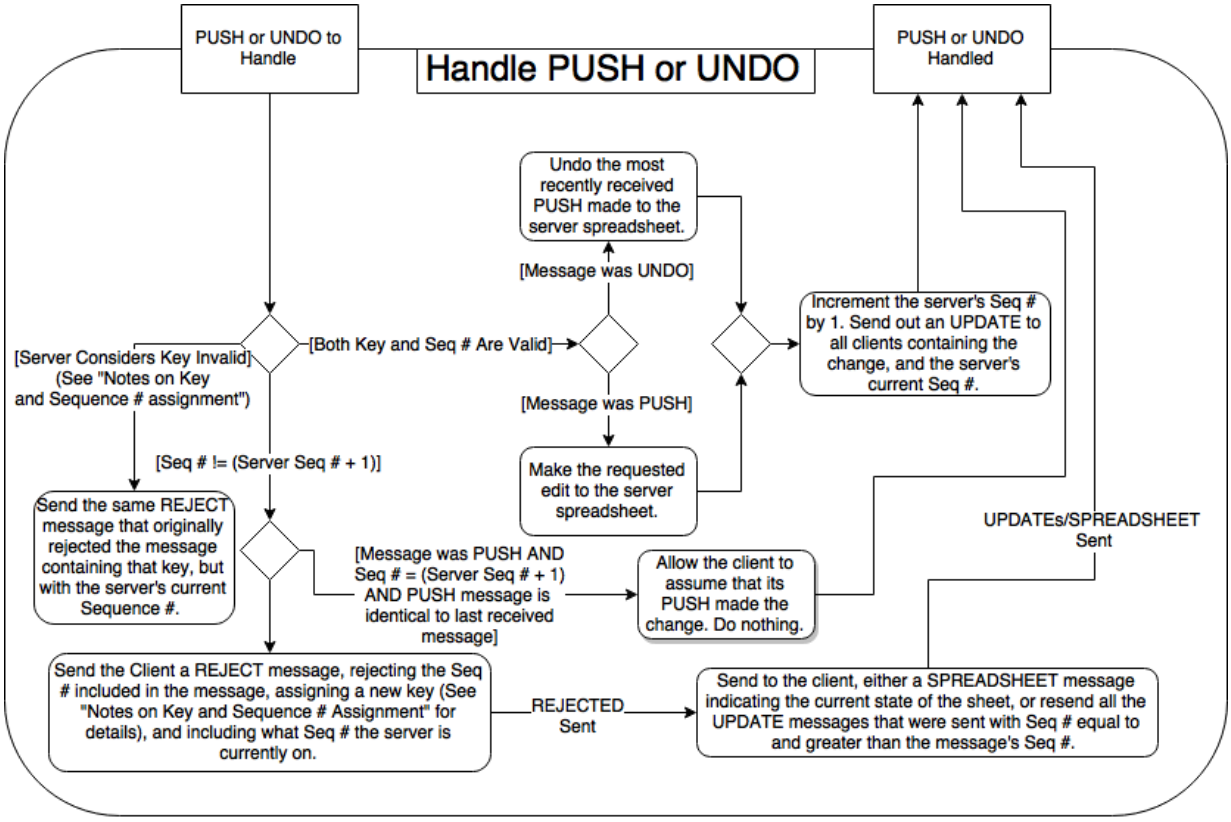

## Server: Managing Clients: Detailed Steps:

The server should accept TCP sockets on port 13505. When a client connects to the server, the server's default state for that client should be to listen for client messages. The following are actions that should be taken, given some message is received.

- **OPEN:** If an OPEN message is received, then one of two things could happen:
    - **If the name specified in the OPEN message refers to a spreadsheet that exists:** The server should respond with a SPREADSHEET message with the all the non-empty cells from the spreadsheet named, a client key, different from the client's current key, and the current sequence # for the spreadsheet.
    - **If the name specified in the OPEN message refers to a spreadsheet that does not yet exist:** The server should respond with a SPREADSHEET message with no cells, a client key different from the client's current key, and a sequence # of 1.

- **LISTSHEETS:** If a LISTSHEETS message is received, then the server should respond with a SHEETLIST message containing all existing spreadsheets.
- **DELETE:** If a DELETE message is received, then one of two things could happen.
    - **If the name specified by the DELETE corresponds to a spreadsheet in use by any clients:** Do nothing. We don't allow clients to DELETE spreadsheets that are in use by themselves, or other users.
    - **If the name specified by the DELETE does not correspond to a spreadsheet in use by any clients:** The server should now act as if the spreadsheet specified by the DELETE does not exist.
- **PUSH/UNDO:** If the client sends a PUSH or UNDO message, one of two things could happen.
    - If the client has not yet sent an OPEN command, the server should respond with a REJECTED message, with a client key and sequence # of 0.
    - If the client has sent an OPEN command, move to Server: Handle PUSH or UNDO.

## Server: Handle PUSH or UNDO: Diagram:



## Server: Handle PUSH or UNDO: Detailed Steps:

When the server receives a PUSH or an UNDO, it needs to perform the following steps:

- **If the Key of the message matches some previously rejected message:** The server should re-send the same REJECTED message that originally rejected that key, but with the server's current sequence # instead of the server's sequence # at the time of the original message.
- **If the Seq # of the message is not equal to the server's current sequence # plus 1:** First, send the client a REJECTED message rejecting the Seq # of the message the client sent. The REJECTED message should also include a newly assigned key, and the current Seq # of the server. Once the REJECTED is sent, the server should follow it with either:
    - All of the UPDATE messages previously sent with a Seq # equal to, or greater than the Seq # of the received message.
    - A SPREADSHEET message detailing the current state of the spreadsheet.
- **Otherwise, the Key and Seq # are valid:** Two things can happen here, depending on if the message received was an UNDO, or a push.
    - **Message was an UNDO:** Undo the most recently accepted PUSH (from any client, not just the client that sent the UNDO).
    - **Message was a PUSH:** Make the requested edit to the server spreadsheet.
    - **For both cases:** Once we perform either of the preceding steps, we should increment the server's Sequence # by 1. We should then send out an UPDATE to all clients containing the change, and the server's current Sequence #.

# Detailed Descriptions:

## Server to Client

### SHEETLIST

A SHEETLIST message is a message that contains all the spreadsheets that currently exist on the server. It's sent in response to a LISTSHEETS message from the client.

**Format:**

After the message tag, there's an Int, referring to the number of sheets that currently exist on the server. After the int, there's a number of string parameters, each containing the name of a sheet that exists on the server. The number of string parameters is equal to the number of sheets that currently exist on the server.

- Example: There are 5 sheets on the server, named "Sheet One", "Sheet Two", "Sheet Three", and so on. The following would be a SHEETLIST message sent from the server:

Ex: {SHEETLIST,5,"Sheet One","Sheet Two","Sheet Three","Sheet Four","Sheet Five"}

## UPDATE

An UPDATE message is a message that contains a single edit that the server wishes to send to a client. It's sent in response to the server accepting an edit from some client, in the form of either a PUSH or an UNDO.

**Format:**

Following the message tag, there's the sequence # of the UPDATE, in the form of an Int. Following the sequence #, there's a string containing which cell the edit is going in. Following the cell name, there's a string containing the new contents of the named cell.

**Example:** The server wishes to tell a client that the formula "=A1+B2" has been entered into cell A4, and that the sequence # of that edit was 15. Following would be the message that the server would send to the client:

Ex: {UPDATE,15,"A4","=A1+B2"}

## REJECTED

A REJECTED message is a message that indicates to a client that its edit (which could be either a PUSH or UNDO message) was rejected. REJECTED messages are sent in response to either the server not receiving an OPEN from a client before it attempts to make an edit, the client sending a message with a valid key but an invalid Sequence #, or the client sending a message with an invalid key.

Case 1: When a server is sending a REJECTED in response to a client not having sent an OPEN prior to edits, it should send a REJECTED with a Sequence # and key of 0.

Case 2: When a server is sending a REJECTED in response to an UNDO or PUSH with an invalid Sequence #, it should send a REJECTED with the sequence # of the UNDO or PUSH, a newly assigned key (See Notes on Key and Sequence # Assignment), and the current sequence # of the server (to indicate to the client when it should resend the REJECTED messages if it desires).

Case 3: When the server is sending a REJECTED in response to a PUSH or UNDO with an invalid key (see Notes on Key and Sequence # Assignment), it should re-send the REJECTED message that originally made that key invalid, except with the server's current sequence #.

**Format:**

Following the REJECTED message is the sequence # of the message that the server wishes to reject, in the form of an Int. For Case 1, this is 0, For Case 2, this is equal to the Sequence # in the received PUSH or UNDO. Following the sequence # rejected by the message is the client's new key, in the form of an Int. For Case 1, this is 0. For Case 2, this is equal to whichever key the server assigns to the client. Finally, the last parameter is the current sequence # of the server. For all cases, this is the same.

**Example (Case 1):** A client is sending a PUSH to the server, but the server never received an OPEN message from that client. The server's current seq # is 15.

Ex. {REJECTED,0,0,15}

**Example (Case 2):** A client is sending a PUSH to the server with an old seq # (13), but a valid key (3). The server's current seq # is 22.

Ex. {REJECTED,13,4,22}

**Example (Case 3):** A client is sending an UNDO to the server with an invalid key. The key was invalidated with the previous example. The server's current seq # is 25.

Ex. {REJECTED,13,4,25}

**Server to Client:**

After the server sends a REJECTED message, the server should will also send the corresponding UPDATE messages to get that specific client up to date to the current sequence number; however, if the client is more than 15 (can be changed as desired) sequence numbers out of date, then the server will instead send a single spreadsheet message. This will ensure that the client will get back up to date, and will stop receiving REJECTED messages. The client, if they hadn't received UPDATE or SPREADSHEET messages within a timeout could send an OPEN message to proactively get back up to date as well.

## SPREADSHEET

A SPREADSHEET message is a message sent from server to client that contains information about every non-empty cell, as well as the key and sequence # that the client should start at. It can be sent for a number of reasons, including when the client opens a sheet, when the server decides to send it in lieu of an UPDATE, or when the server wants to change something about a client's key and sequence # information.

**Format:**

Following the SPREADSHEET message is an Int corresponding to the number of non-empty cells in the spreadsheet. Following that Int is a number of string parameters equal to 2*(number of non-empty cells). The parameters are ordered such that a cell name is immediately followed by the contents of that cell. The first parameter following the Int is a cell name. After all of the cell name and cell contents parameters, there is an Int corresponding to the current sequence # of the spreadsheet. Following the sequence #, there is an Int that corresponds to the key assigned to the client.

**Example:** A server has a spreadsheet with 3 non-empty cells: A1, which contains "3", B2, which contains "=A1+2", and C3, which contains "Some String". The server's current sequence # is 15, and the server assigns a client key of 2 to the client. The SPREADSHEET message sent to some client opening the spreadsheet would be:

{SPREADSHEET,3,"A1","3","B2","=A1+2","C3","Some String",15,2}

## Notes on Key and Sequence # Assignment

Key and Sequence # assignment has been left loosely defined in this protocol, to allow the server to be implemented in a number of ways. However, certain rules must be followed when it comes to assigning keys and sequence #s.

**Reserved Key and Sequence #'s**: zero is a reserved value for both Key and Sequence #, to indicate some error state. They are reserved to allow for further modification to the protocol.

**Sequence #:** The server's sequence # is the global authority on what PUSH and UNDO messages have been accepted.

- Unless dealing with desync, clients expect the sequence # of UPDATE messages to be sequential, unless a SPREADSHEET message is received.
- If a client receives a SPREADSHEET message, it considers the sequence # from that message to be its new sequence #, regardless of previous state.
- Clients can make no assumptions about the status of the server's Sequence # unless told in the form of a REJECTED, UPDATE or SPREADSHEET message.
- More explicitly, clients can make no assumptions about the status of the server's Sequence # while not connected to the server.

**Key:** The key is the client's way of indicating to the server that it has received a REJECTED message. If the server rejects a message, then it should consider that key invalid until such a time that it can be assured there are no outstanding messages sent by the client with the same key. To simplify implementation, the client makes a number of assumptions about the key:

- The client assumes that a REJECTED message rejecting PUSH messages not already rejected will assign a key greater than its current key.
- When the client receives a SPREADSHEET message, the key included in that message is the client's new key, regardless of previous state.

Using these guidelines, it's easy to implement a trivial key and sequence # assignment system, but the rules also allow for more robust, but more complicated implementations.

### Notes on Message Processing

This protocol assumes that the server processes all messages received will be processed, and that they'll be processed in the order in which they were received by the socket. Any violation of this restriction may cause undefined behavior.

## Client to Server

### OPEN

Sent to the server when the client wants to open a spreadsheet for editing. Can also be sent if the client wants a SPREADSHEET message of the sheet for some reason. Responded to with a SPREADSHEET message.

**Format:**

Following the message key is a single string parameter, containing the name of the spreadsheet the client wishes to open.

**Example:** If the client wishes to open a spreadsheet "My Sheet", it should send the following message:

{OPEN,"My Sheet"}

## LISTSHEETS

Sent to the server when the client wants to know which spreadsheets exist on the server. Responded to with a SHEETLIST message.

**Format:**

This message has no parameters.

**Example:**

{LISTSHEETS}

## DELETE

Sent to the server when the client wants to delete a spreadsheet. If the sheet does not exist, does nothing. If the sheet is in use by one or more clients, does nothing. Otherwise, the sheet no longer considers the sheet to exist.

**Format:**

This message has one parameter, for the name of the sheet.

**Example:** If the client wishes to delete a spreadsheet called “NeedsToBeGone”, it should send the following message

{DELETE,”NeedsToBeGone”}

## PUSH

Sent to the server when the client wants to make a change to the spreadsheet. This protocol requires that clients check their PUSH messages for invalid cell names, circular dependencies and malformed formulas (these three definitions are defined in Glossary/Quick Reference) before sending them to the server. If the PUSH message meets this requirement, it can be sent to the server. The client should beware that the server reserves the right to respond to any PUSH message with a REJECTED message. If a PUSH message is rejected, that message, and all PUSH/UNDO messages sent after it with the same key, are considered invalid, and will never be added to the server without resubmission.

**Format:**

Following the Message Key, the first parameter in a PUSH message is the sequence # of the PUSH, in the form of an Int. Directly following the sequence # is another Int parameter, containing the client’s key. Following the key is the cell name of the cell the PUSH is editing, in the form of a String. Following the cell name is the new contents of the cell, in the form of a string.

**Example:** If the client wants to set cell A4 to be “Some String”, the client’s current sequence # is 13, and the client’s current key is 3, the push message the client would send to the server would be:

{PUSH,14,3,”A4”,”Some String”}

## UNDO

Sent to the server when the client wants to undo the most recently made change to the spreadsheet. The UNDO commands the most recent PUSH message that the server accepted from any client, not necessarily the most recent PUSH message sent by the client. UNDOs

can be rejected in the same manner as a PUSH, and when rejected, invalidate the key that they were sent with, in the same manner as a PUSH does.

**Format:** Following the Message key, the first parameter in an UNDO message is the sequence # of the UNDO, in the form of an Int. Directly following the sequence # is another Int parameter, containing the client's key.

**Example:** If the client wants to undo the most recently committed PUSH, where the client has a sequence # of 10, and a key of 4, it would send the following command

{UNDO,10,4}

# Changes and Improvements:

Any further improvement to the protocol past its initial state will be listed here, as well as in its relevant section.

# Glossary/Quick Reference:

## General Information & Definitions:

**Network Protocol Used:** TCP

**Port Used:** 13505

**Circular Dependencies:** A circular dependency is when a cell contains a formula which, either directly or indirectly, references itself. Examples

- Cell A1 has contents "=A1+1": Circular Dependency, as the cell depends on itself.
- Cell A1 has contents "=B2 + 1", Cell B2 has contents "=A1+1": Circular Dependency, as A1 refers to B2, which in turn refers to A1.
- Cell A1 has contents "=B2 + 1", Cell B2 has contents "=C3+1", Cell C3 has contents "=A1+1": Circular dependency, as A1 refers to B2, which refers to C3, which refers back to A1.

**Malformed Formulas:** A malformed formula is a formula that could not evaluate to a numerical value, even when provided with proper input. All parentheses must be matched, and there is no implicit multiplication between parameters leading open parenthesis or following closing parenthesis. Examples:

- "=A$+1": Invalid, as A$ is not a valid variable name.
- "=A1+": Invalid, as we have a binary operator that's missing a parameter.
- "=(A1+ 2": Invalid, as we don't have a closing parenthesis
- "=A1+2)": Invalid, as we don't have an opening parenthesis.
- "=2(A1+2)":Invalid, as there's no operator between the 2 and the open parenthesis.
- "=(A1+2)2": Invalid, as there's no operator between the 2 and the close parenthesis.
- =(A1+2)": Valid. Doesn't break any established rules.

**Valid Variable Names:** A valid variable name is some letter A-Z, followed by a number that can range from 1 to 99 inclusive, with no other characters. Examples:

- A1: Valid. A is an uppercase letter A-Z, and 1 is in the range 1-99, inclusive.
- a1: Invalid. "a" is not an uppercase letter.
- A100: Invalid. 100 is not in the range 1-99, inclusive.
- $1: Invalid. $ is not an uppercase letter, A-Z.
- A1$: Invalid. $ is an additional character not explicitly allowed by the above definition.

**PUSH/UPDATE Equality:**

- A PUSH is equal to another push if it has the same parameters.
- A PUSH is equal to an UPDATE if the UPDATE and PUSH make the same edit, with the same sequence #.

Server to Client:

| **Message Tag** | **Parameters** | **Use** |
|---|---|---|
| SHEETLIST | Int:<br># of sheets<br>String:<br>A number of strings containing the name of each existing spreadsheet. | Used to send to the clients a list of all existing server spreadsheets. |
| UPDATE | String:<br>Name of the edited cell<br>Contents of edited cell.<br>Int:<br>Sequence # | Sent to the client to indicate a change in the server's spreadsheet model. |
| REJECTED | Int:<br>Sequence of rejected message #<br>New client key<br>Current server sequence # | Sent in response to an invalid client PUSH. Indicates that the message denoted by the included sequence #, and all following messages, are invalid and must be discarded. Also includes a new client key that the client should use going forward. |
| SPREADSHEET | Int:<br># of cell name-contents pairs<br>String:<br>"cellName", "cellContents" for each cell pair<br>Int:<br>Sequence #<br>Client key | Represents the current state of the spreadsheet in its entirety, including every non-empty cell value. When received, the client sequence # and key is set to the specified # and key. |

Client to Server:

| **Message Tag** | **Parameters** | **Use** |
|---|---|---|
| OPEN | String:<br>Name of the sheet the client wants to open | If a spreadsheet corresponding to sheetName exists, the client will send a SPREADSHEET with that sheet's data. Otherwise, server creates that spreadsheet, and sends an empty SPREADSHEET. |
| LISTSHEETS | (none) | Server will respond with a SHEETLIST message, with the names of all created spreadsheets. |
| DELETE | String:<br>Name of the sheet that the client wants to delete. | If the sheet is not currently in use by any client, removes the specified sheet from the server. Otherwise, does nothing. If the sheet doesn't exist on the server, does nothing. |
| PUSH | String:<br>Name of cell the client wants to edit.<br>Contents that the client wants to set the cell to.<br>Int:<br>Sequence # of the PUSH<br>Client's current key. | Indicates a change in the client spreadsheet model.<br>If the sequence # or key is not correct, the push may be REJECTED. If we receive one of our PUSH messages back as an UPDATE, that message, and all preceding messages were confirmed by the server. |
| UNDO | Int:<br>Sequence # of the UNDO<br>Client's current key. | Requests the server to "undo" the most recently made PUSH committed to the sheet, by any client. |