Spreadsheet Communications Protocol

Version 2

Thursday, March 29th, 2018

Prepared by: **#include "not_a_virus.h"**
Ella Moskun
Matthew Christensen
Josh Butner
Jennifer Nelson

Prepared for: **Software Practice II**
CS 3505-001
Spring 2018
The University of Utah
Prof. Peter Jensen

Changes from Version 1

In light of peer feedback and new requirements being announced in class, this version of the protocol makes the following changes and clarification:

- **Addition of 'focus' and 'unfocus' messages to reflect new UI requirements announced in lecture**
- Clarification that control characters are not allowed in cell contents nor spreadsheet names
- Clarification that cell names are case insensitive

Background

This protocol is designed to specify client-server communication for networked, multi-user spreadsheet applications extended from those previously created in Software Practice I (CS 3500-001, Fall 2017). The protocol is intended to balance simplicity with completeness and user-friendliness. For familiarity, some inspiration was taken from the networking of CS 3500's SpaceWars project.

Spreadsheet applications implementing this protocol should retain the behavior specified in CS 3500's assignment descriptions (PS2 through PS6) unless otherwise stated in this document.

Changes from CS 3500

Spreadsheet applications that implement this protocol will have following changes from those created in CS 3500:

- Creation of a separate server application
- Division of responsibilities between client and server applications
- Addition of networking facilities to the client
- Removal of any file handling from the client
- Addition of user interface mechanisms to the client for connecting to a server, selecting a spreadsheet/creating a new spreadsheet, performing undo and revert operations
- **Updated behavior of client to allow for cell errors (circular dependencies and invalid formulas) in the spreadsheet**
- Creation of some mechanism (such as displaying error messages in cells or highlighting cells) to communicate cell errors to the user
- Removal of any bonus features that may conflict with this protocol
- **Changed behavior of client to reflect new UI requirements announced in lecture**

Allowing cell errors (including circular dependencies) is the most radical change required by this protocol. Allowing and gracefully presenting them, however, removes the enormous complexity that would be required to prevent them in a multi-client scenario. This is the method that most spreadsheets use. Excel, for example, displays circular dependencies as #REF.

Division of Responsibility

Responsibilities are divided between clients and servers as follows:

Client	Server
● Connect to a server indicated by the user ● Process all messages received from the server ● Send messages to the server as indicated by this protocol ● Provide a mechanism for the user to request a particular spreadsheet from the server ● Evaluate formulas ● Display cell values ● Display cell errors ● Display when another user is in the process of editing a given cell ● Provide mechanisms for the user to request edit, undo, and revert operations	● Listen for and connect clients ● Process all messages received from clients ● Send messages to clients as indicated by this protocol ● Maintain persistent storage of spreadsheets' contents as strings ● Maintain edit history to facilitate undo and revert operations ● Allow for multiple clients to interact with a given spreadsheet concurrently ● Allow for multiple spreadsheets to be interacted with concurrently

General Networking Considerations

All network communication under this protocol will be through the Transmission Control Protocol (TCP) on port **2112**. All messages are transmitted as NULL-terminated strings encoded in UTF-8.

Servers and clients will process all messages **in the order that they were received**. Servers and clients will ignore invalid messages impossible requests (such as attempting to undo or revert when there are no changes to undo or revert).

Message Format

All network messages in this protocol are strings containing, in order:

- A key indicating the message type
- A space character
- The contents of the message, if applicable for that message type
- A End of Text (ETX) character (U+03, represented in this document as \3)

Note that there is **always** a space following the key, even if the contents of the message are omitted.

Messages that refer to certain cells use a <cell_id> in the form of a single letter A-Z (upper or lowercase) followed by a number 1-99, **without** any leading zeros. For consistency this document uses capital letters, but lowercase and uppercase letters should be treated the same (for instance, A1 and a1 are the same cell).

To assist with parsing, line feed and other control characters are **not** allowed in cell contents, user IDs, or spreadsheet names.

Certain server messages (connect_accepted and full_state) use line feed characters (U+0A, represented in this document as \n) to separate elements within their contents.

Table of Client Messages

Key	Content	Examples	Meaning
register	(none)	"register \3"	Initiate the handshake with the server
disconnect	(none)	"disconnect \3"	Disconnect the client from the server
load	<spreadsheet_name>	"load sales\3" "load other_sheet\3"	Load the spreadsheet with the name <spreadsheet_name> if it already exists on the server, otherwise create it
ping	(none)	"ping \3"	Client ping to the server
ping_response	(none)	"ping_response \3"	Client response to ping from the server
edit	<cell_id>:<cell_contents>	"edit A1:=2*3+A2\3" "edit Z99:price\3" "edit C30:\3"	Request the contents of <cell_id>be set to <cell_contents>
focus	<cell_id>	“focus A9\3” “focus Z32\3”	Notifies the server that the sending client is in the process of editing <cell_id>.
unfocus	(none)	“unfocus \3”	Notifies server that sending client is no longer in the process of editing.
undo	(none)	"undo \3"	Request the most recent edit or revert be undone
revert	<cell_id>	"revert A1\3"	Request to revert the most recent change to <cell_id>

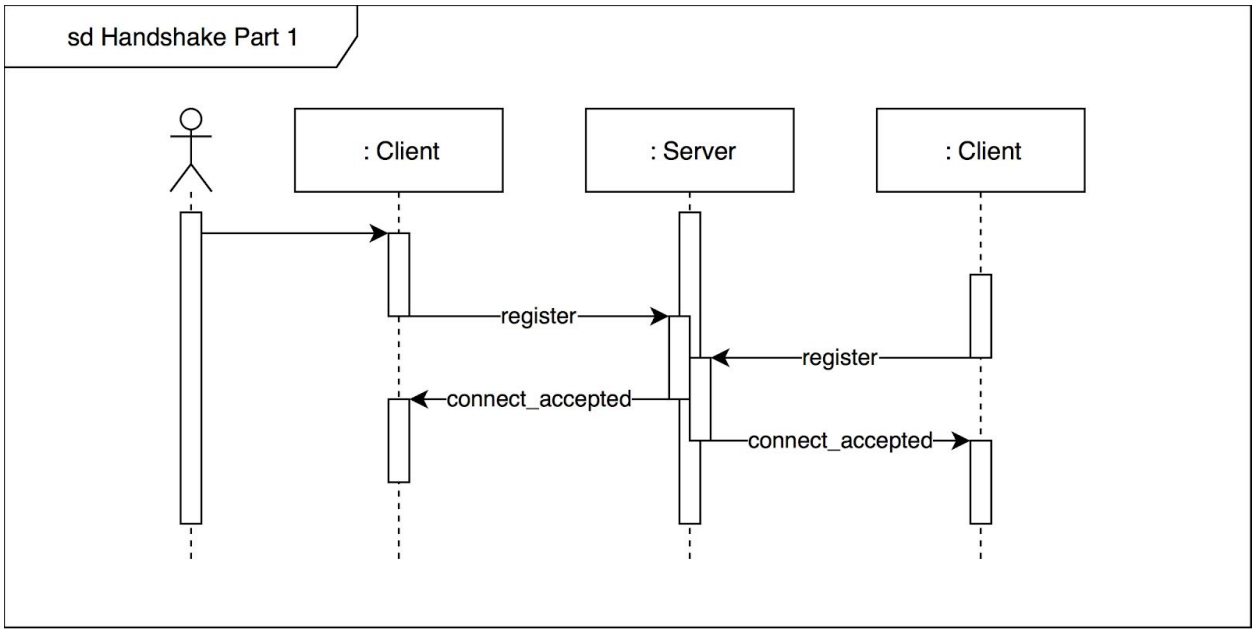
Table of Server Messages

Key	Content	Examples	Meaning
connect_accepted	<spreadsheet_name>\n … <spreadsheet_name>	"connect_accepted sales\n marketing ideas\n another_sheet\3" "connect_accepted \3"	Acknowledge new client and give list of existing spreadsheets. **The list may be empty.**
file_load_error	(none)	"file_load_error \3"	Unable to open or create the requested spreadsheet
disconnect	(none)	"disconnect \3"	Disconnect the client receiving the message
ping	(none)	"ping \3"	Server ping to the client
ping_response	(none)	"ping_response \3"	Server response to ping from the client
full_state	<cell_id>:<cell_contents>\n … <cell_id>:<cell_contents>	"full_state A6:3\n A9:=A6/2\n\3" "full_state \3"	The full state of the spreadsheet, with the id and contents of every non-empty cell
change	<cell_id>:<cell_contents>	"change A4:=A1+A3\3" “change E8:3\3” "change D3:\3"	Notification that the contents of <cell_id> is now <cell_contents>, following an edit, undo, or revert
focus	<cell_id>:<user_id>	“focus A9:unique_1\3” “focus A9:2nique4U\3” “focus A9:3nique5U\3”	Notify clients that a user, represented by a unique string <user_id> is in the process of editing the specified cell.
unfocus	<user_id>	“unfocus unique1\3” “unfocus 2nique4U\3” “unfocus 3nique5U\3”	Notify clients that a user, represented by a unique string <user_id> is no longer in the process of

editing.

Handshake

When a server first starts up, after it initializes itself, it should start a thread that constantly listens for any clients trying to connect to it. When a client starts up and the user indicates it wants to connect to a particular server, it will initialize contact by sending a register message to the server. This is the first step of the handshake across the connection. Upon receiving the register message, the server should respond to the client with a connect_accepted message, prompting the user to request a load for a specific spreadsheet from a list of names.



The first half of the handshake, showing two clients connecting simultaneously

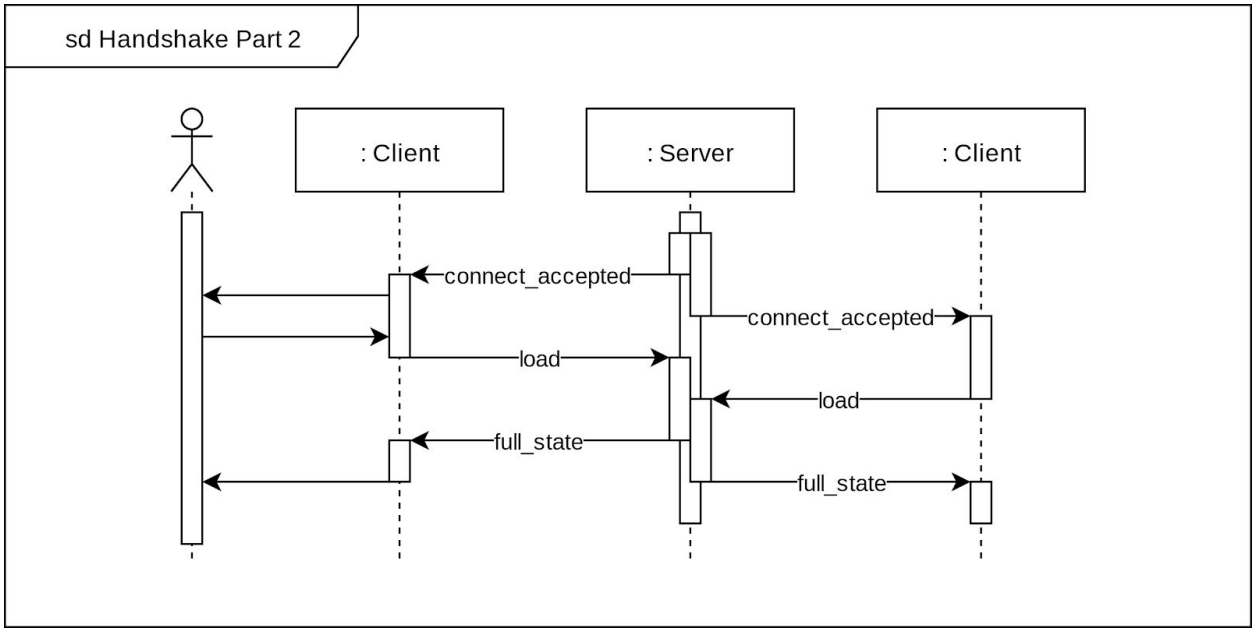
When the user chooses their spreadsheet, the client must send a load message to the server, with the name of the spreadsheet that they want to edit (e.g. load best_spreadsheet_ever). Provided that the name offered is determined to be valid, the server will do one of two things:

- If the named spreadsheet currently exists on the server, then it will send the current contents of every non-empty cell to the client, using a full_state message. This message will the name of each non-empty cell of the spreadsheet, followed by its contents. The different cells will be delineated with a newline character.
- If no file with the name exists, then the server will create a new empty spreadsheet. The server will then send a full_state message with no cell information.

If the server is for some reason unable to open/load/create the spreadsheet with the given name, it will instead reply with a file_load_error message. The server will continue listening

for a load request, and the client may make a new load request until it receives no error received.

Until a user receives the full state of their requested spreadsheet, they should not be able to send cell edit requests to the server. Once the entire spreadsheet is sent, the handshake is complete, and the user may now request edits. In addition, after the spreadsheet info comes through, a ping loop will be started by both the client and server.



The second half of the handshake, after two clients connected.

After the full_state message is sent, a server may **optionally** send ‘focus’ messages to the newly connect client for every other user/client that is currently “focused” on a cell.

Ping Loop

The ping loop is used to determine if a client or server has disconnected without notifying the other end (e.g. crashed server or forcibly closed client). Every 10 seconds the server will ping each connected client and listen for a response. If the server fails to receive a response from a client 6 times (a full minute) the server will initiate the protocol to disconnect a client. Every 10 seconds the client will ping the connected server and listen for a response. If the client fails to receive a response from a client 6 times (a full minute) the client will initiate the protocol to disconnect from the server.

Server Ping Protocol

- Clients are pinged (using the “ping” message) at 10 second intervals.
- Always listening for a “ping_response” message from clients.
- On receiving a response, refresh the the timeout to 0 from the client it came from.
- On receiving a “ping” message from a client, immediately sends a “ping_response” to the same client.
- If the timeout for a client reaches 60 seconds, initiate the protocol to disconnect a client.

Client Ping Protocol

- Connected server is pinged (using the “ping” message) at 10 second intervals.
- Always listening for a “ping_response” message from the server.
- On receiving a response, refresh the the timeout of the server to 0.
- On receiving a “ping” message from a server, immediately sends a “ping_response” back to the server.
- If the timeout for the server reaches 60 seconds, initiate the protocol to disconnect from the server.

Disconnect

A client will disconnect from a server in 3 conditions:

- The user closes the client.
- The server disconnects the client.
- The server fails to reply to a ping for 60 seconds.

A server will disconnect a client under 3 conditions:

- The server is closed.
- A client disconnects from the server.
- A client fails to reply to a ping for 60 seconds.

When a client disconnects from a server, it sends the server the disconnect message only if this disconnection was not initiated by the server. The client is responsible for updating the user interface to reflect the disconnection in some reasonable way.

When a server disconnects a client, it sends the client the disconnect message only if this disconnection was not initiated by the client and remove the client from the list of active clients. The server is responsible for handling the situation where there are no longer any active clients in some reasonable way.

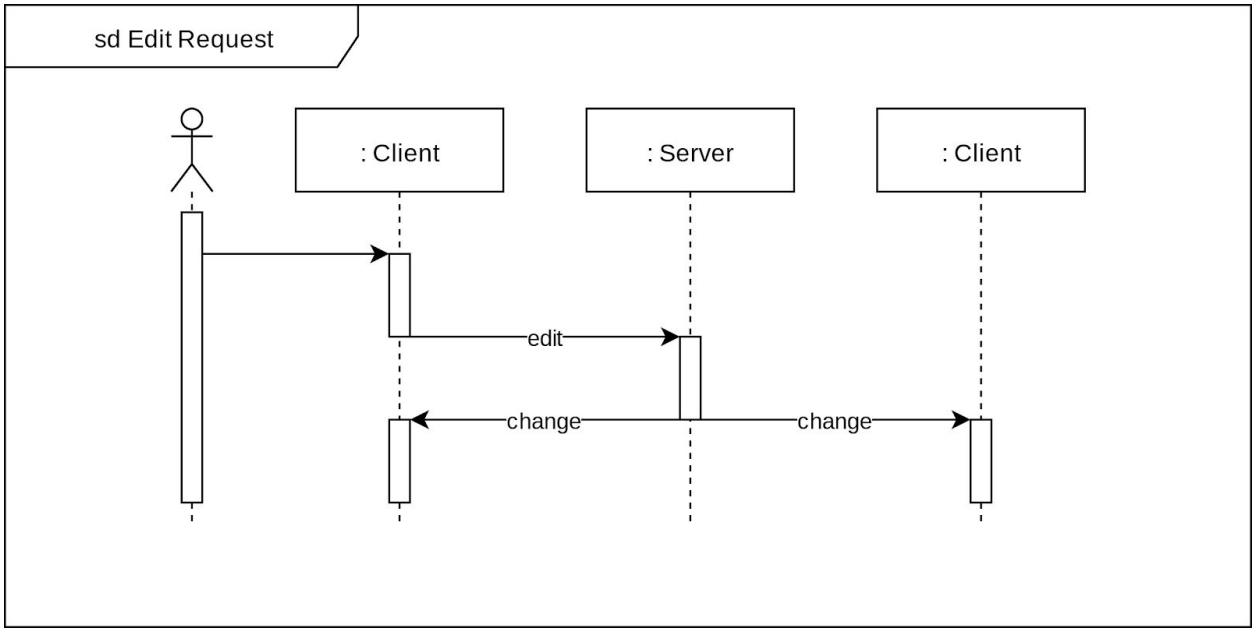
Edit

Clients request a change to the spreadsheet by sending an edit message. When an edit request is received, the server will process it and change the value of the cell to the requested value, then send the contents of the changed cell to all clients (including the client who requested the edit) as a change message. Clients are **not** to make any changes to their copy of the spreadsheet data unless they receive a command from the server.

Cells can be made empty by simply giving the representing their contents as the empty string. An edit message to set a cell to be empty would look like "edit A1:\3". Similarly, a change message to set a cell to be empty would look like "change A1:\3".

Upon receiving an edit request, the server can refuse the message. (This should only happen if a user requests an undo or revert when there is no prior state to revert to.) After changing its internal state, the server sends this change out to **all** connected clients.

Clients are expected to incorporate **every** server-commanded change, regardless of the change's contents. (The client should display error values as appropriate.) There is no special confirmation to the client who requested the edit--from its perspective, this could be *anyone's* change request coming through.



A simple example of an approved edit request.

User Focusing (Knowing Who is Editing)

Clients notify the server of when they are “focusing” on a cell (in the process of editing that cell) by sending *focus* and *unfocus* messages. The focus message specifies which cell is being focused on, though the unfocus message does not specify a cell.

The server notifies clients of a client ‘focusing’ on a cell by sending a *focus* message with the cell name and unique ID, or an *unfocus* message with just the unique ID. The unique ID is a string *unique to the client that is focusing* and *persistent throughout a client’s connection.*

This protocol specifies that *focus* messages must have a *unfocus* message before another *focus* message. (I.e. if “client_1” sends “focus a3\3”, it must send “unfocus \3” before it can send “focus a4\3”. Likewise, the client must send “unfocus client_1” between “focus a3:client_1\3” and “focus a4:client_1\3”.)

Multiple users/clients may “focus” on the same cell simultaneously.

Undo and Revert

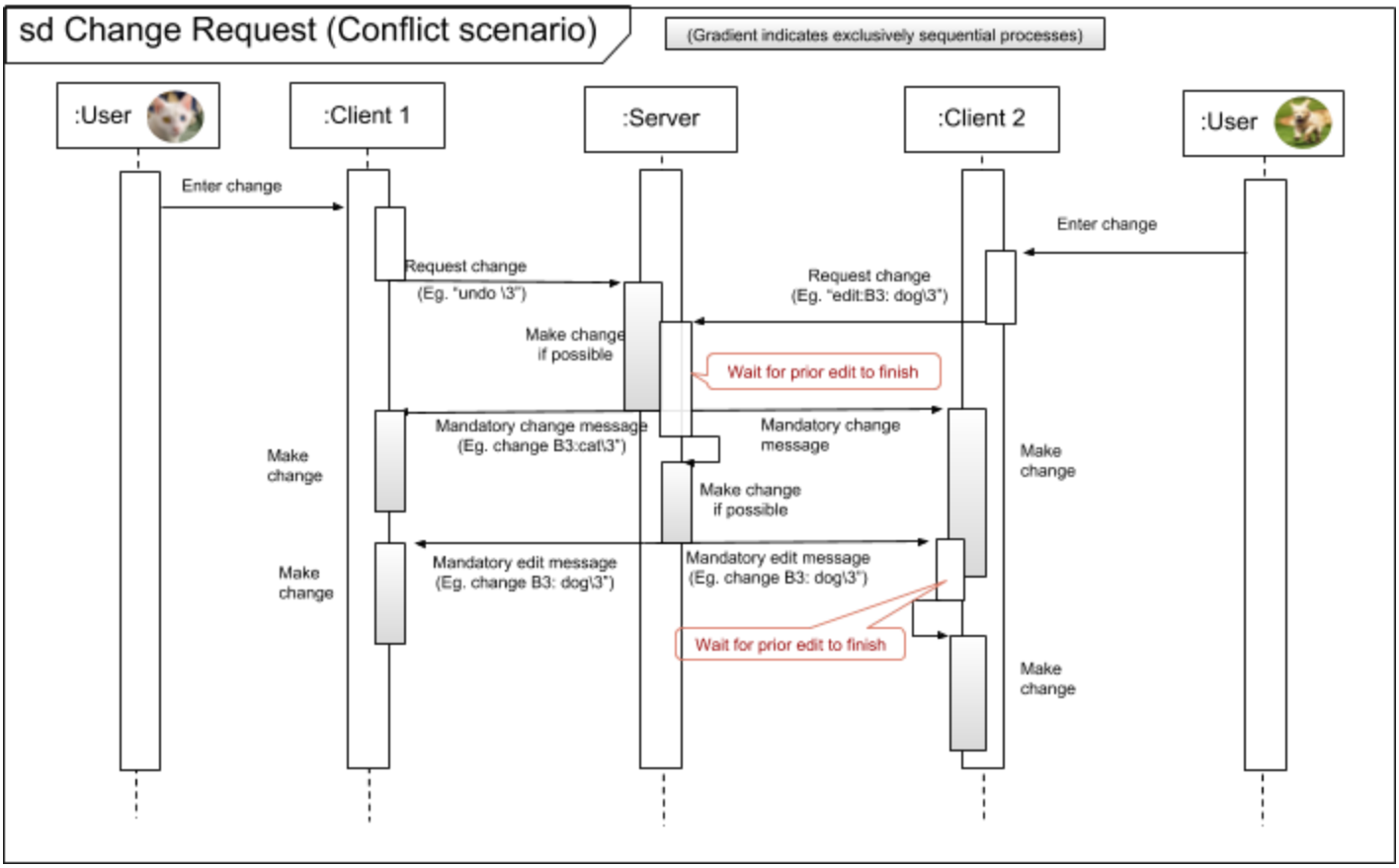
The undo and revert operations count as edit requests but use different messages. The communication sequences are identical except for the initial message.

- When an undo request is received, the server will process it and undo the most recent change, then send the contents of the changed cell to all clients (including the client who requested the undo) as a change message.
- When a revert is requested the server will process it and revert the most recent change to the target cell, then send the contents of the changed cell to all clients (including the client who requested the revert) as a change message.
- An edit can be undone by the undo and revert action. A revert can be undone by an undo, but not a revert. An undo is a destructive action, and cannot be undone.
- Care must be taken that reverting a cell multiple times steps back through history rather than toggles the cell between two states.
- Care should be taken to ensure that undoing a revert command will return a cell to its value before the revert occured.
- Undo and revert must maintain a complete history back to the point when the spreadsheet was loaded this server session.

Example Interaction of Multiple (Edit, Undo, or Revert) Requests

In the more complicated case of two clients requesting changes to the same cell nearly simultaneously, the server should be careful of race conditions. (Likewise, the client should be careful about accepting nearly simultaneous server commands.) This protocol specifies

that the server must consider change requests sequentially in the order they are received. The diagram below shows a conflict scenario with exclusively sequential changes for both the server and clients.



Example of two clients simultaneously requesting edits to the same cell. Gradients indicate exclusively sequential processes.