Communications Protocol

Vijay Bajracharya
Nick Hayes
Rohun Kaddu
Mateo Lopez
Carson Storm
Adian Taylor

Abstract

This document defines the communication protocol used by the clients and servers in the collaborative spreadsheet. The protocol is designed to enable multiple users (clients) to edit a spreadsheet simultaneously in such a way that all changes are reflected in real-time for all clients. This protocol defines the behavior of the server and clients and the messages that are exchanged.

Contents

1 Introduction

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

2 Common Client-Server interactions

2.1 Connecting to the Server

The client will open a TCP connection to the server on port 2112 after the connection has been opened, the server will start accepting messages formatted as JSON strings. This connection may be closed and reopened at the clients' discretion, but any open spreadsheet will need to be reopened.

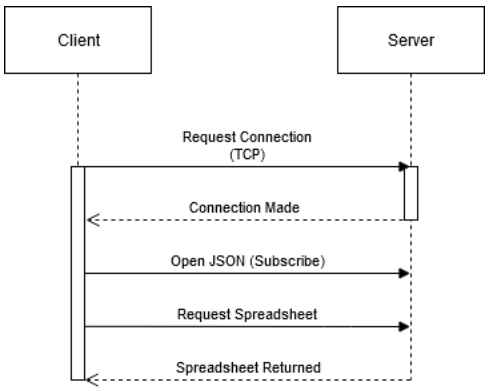

Figure 1: Connecting to the server sequence

2.2 Managing Spreadsheets

The protocol defines four basic commands that manipulate spreadsheets as a unit: create, delete, rename, and list_spreadsheet. Every spreadsheet is identified by an *id* and associated with a *name* that need not be unique. The *name* is primarily for display purposes, while the spreadsheet *id* is used to identify it in all operations pertaining to a specific spreadsheet.

2.2.1 Creating a new spreadsheet

To create a new spreadsheet the client will use the create command, which accepts the name of the spreadsheet to create, and responds with the *id* of the newly created spreadsheet. The newly created spreadsheet will persist on the server until it is deleted.

2.2.2 Deleting a spreadsheet

To delete a spreadsheet the client will use the delete command, which accepts the *id* of the spreadsheet that is to be deleted. This will delete the spreadsheet from the spreadsheet list and all of its history; it will also trigger an update for all clients who currently have the spreadsheet open, notifying them that the spreadsheet has been deleted.

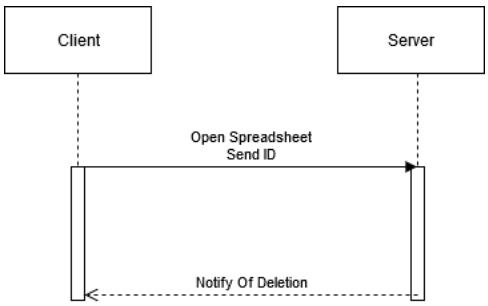

Figure 2: Deleting a spreadsheet sequence

2.2.3 Renaming a spreadsheet

To rename a spreadsheet the client will use the rename command, which accepts the *id* of the spreadsheet to rename and the *name* to rename the spreadsheet to. This command will trigger an update for all clients who currently have the spreadsheet open, notifying them that the name of the spreadsheet has changed. See rename for a description of the command.

2.2.4 Listing all spreadsheets

To list all of the spreadsheets stored on the server the client will use the list_spreadsheets command, to which the server will respond with a list of all of the spreadsheets stored on the server.

2.3 Accessing a spreadsheet

One of the clients' primary responsibilities is to display the contents of open spreadsheets in real-time; this protocol defines four commands used to access the contents of the spreadsheet: open, close, get_history, and get_spreadsheet. These commands do not alter the spreadsheets in any way, they only access the information stored in the spreadsheets.

2.3.1 Opening a spreadsheet

The client will need to stay updated with the current contents of the spreadsheet, so this protocol provides an open command, which will accept the *id* of the spreadsheet that the client would like to open. The server will respond to this command with the current contents of the spreadsheet, as well as adding the client as a subscriber to all future updates regarding the spreadsheets. These updates will contain the current state of the spreadsheet and delete and rename notifications for the spreadsheet.

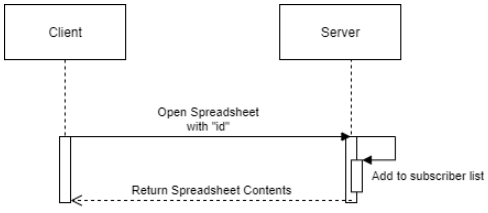

Figure 3: Opening a spreadsheet sequence

2.3.2 Closing a spreadsheet

In the event that a client no longer wishes to receive updates for a spreadsheet, for example, if the user closes the client or the spreadsheet, the client can unsubscribe from spreadsheet

updates by using the close command, which accepts the *id* of the spreadsheet the client wishes to close. This will prevent the client from receiving further updates about the spreadsheet, and the client will need to reopen the spreadsheet to begin receiving updates again.

2.3.3 Getting the edit history of a spreadsheet

There are a few reasons the client would want to get a complete history of a spreadsheet, for example, if the client wanted to be able to revert the spreadsheet to the state it was an hour ago. The client can access the complete history of a spreadsheet using the get_history command, which accepts the *id* of the spreadsheet whose history to get. The server will respond with a list of all the edits that have been made to the spreadsheet. The response will include edits that have been undone and will indicate which edit is the most recent.

2.3.4 Getting the current state of the spreadsheet

In the event that the client wishes to retrieve the current state of the spreadsheet regardless of whether or not they've opened the spreadsheet, they can use the get_spreadsheet command, which accepts the *id* of the spreadsheet to get. The server will then respond with the contents of the spreadsheet.

2.4 Editing a spreadsheet

The client's other main responsibility is to update the spreadsheet stored on the server. This protocol defines two commands to be used for editing a spreadsheet: push and undo. All other required edits can be composed using these two commands. These commands do alter the contents of the spreadsheet.

2.4.1 Applying an edit to the spreadsheet

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

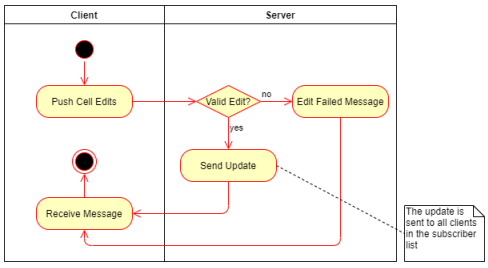

Figure 4: Update propagation after application of edit

2.4.2 Undoing an edit

Undo operations are not considered edits, but simply a rollback of the history of the spreadsheet. This protocol provides an undo command which will put the spreadsheet in the state it was before the most recent edit was applied. *undo* accepts the *id* of the spreadsheet to perform the undo on. This command will trigger an update for all clients that currently have the spreadsheet open.

2.4.3 Reverting an edit

The client may want to revert the spreadsheet to a previous state, that is, reapplying an earlier edit to the spreadsheet. This can be accomplished through the get_history and push commands. The client must simply get the history of the desired spreadsheet, select the edit to revert to, and then push that edit to the server as if it were a new edit. Since, get_history returns edits that have been undone, it is possible to revert an undo.

2.4.4 Editing while offline

In order for the client to have a seamless editing experience while offline, the client will have to cache portions of the history of the spreadsheet, so that undo and reverts can still be supported. Undoes past the point where the client went offline will not be possible, and so will likely have to be replaced with revert operations. The client will also need to keep track of the edit id of the last edit it received from the server before going offline, it will be necessary for syncing with the server when the client comes back online.

2.4.5 Synchroizing with the server after being offline

Synchronizing with the server after being offline is fairly simple for the client provided that it kept some sort of queue for the edits it made while offline, and the edit id of the last edit it received from the server. The client will use the push command with all of it's queued edits and the cached edit id to synchronize itself with the server. The next update it receives will be the merged version of the spreadsheet.

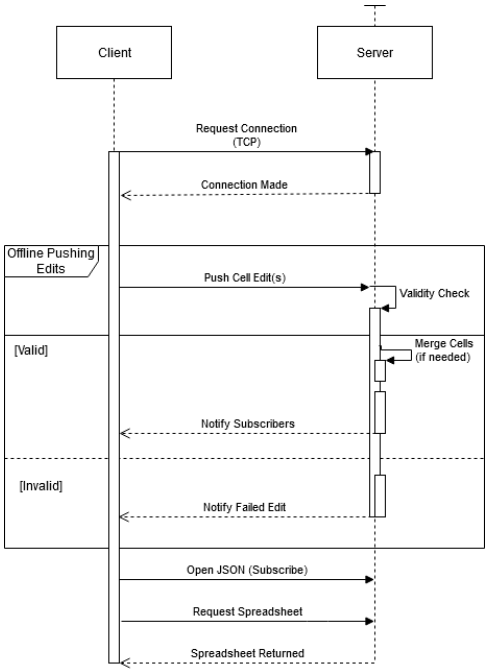

Figure 5: Synchronizing sequence

3 Commands

Command	Description
create	Creates a new spreadsheet
delete	Deletes a spreadsheet
rename	Renames a spreadsheet
list_spreadsheets	Lists all spreadsheets
open	Subscribes the client to updates
close	Unsubscribes the client to updates
get_history	Gets the edit history of a spreadsheet
get_spreadsheet	Gets the contents of a spreadsheet
push	Pushes edits to be applied to the spreadsheet
undo	Undoes the most recent edit of a spreadsheet

3.1 Common Data Types

3.1.1 Command

All *Command*s contain the *command* field which is used to identify the command being executed. All *Command*s will contain the following fields:

Field	Type	Description
command	string	The command to execute
...	any	Command parameters

3.1.2 Result

There server will respond to each *Command* with a *Result*. *Result*s have two variants: *ok* and *error*. All *Result*s will contain the following fields:

Field	Type	Description
result	string	Identifies the command this result is for
ok	boolean	Whether or not the command was successful

If the *Result* is ok ($ok = true$) then the *Result* may contain addition fields that contain the results of the command. Otherwise the *Result* will contain the following fields:

Field	Type	Description
error	string	A description of the error that occurred.

3.1.3 Edits and Spreadsheet

Various commands will accept as parameters or return as a result the contents of a spreadsheet, both *Edit*s and *Spreadsheet*s will contain spreadsheet contents. They differ in that a *Spreadsheet* object must contain the contents of a whole spreadsheet while a *Edit* object may contain a subset of the contents of a spreadsheet. Both *Edit*s and *Spreadsheet*s will contain the following fields:

Field	Type	Description
$CellName	string	The contents of $CellName.

These data types are simply a mapping between cell names and their contents, valid cell names are a capital letter from "A" to "Z" followed by a number from 1 to 99.

3.2 Create

The *create* command is used to create a new spreadsheet. Create expects a *name* field to be provided to serve as the name of the new spreadsheet. Any errors resulting from this command will be purely due to an error occurring on the server, for example, if the server does not have the capacity for a new spreadsheet.

3.2.1 Create Command

In addition to the *command* field which will have a value of "create", this command expects the following fields:

Field	Type	Description
name	string	The name of the newly created spreadsheet.

The following is an example of a *create* command:

```
{
    "command": "create",
    "name": "sheet1"
}
```

Listing 1: create command message

3.2.2 Create Results

The result of this command will contain the standard result fields, in addition to the following fields to be included for ok results:

Field	Type	Description
id	number	The id of the newly created spreadsheet.

The following are examples of results of the *create* command:

```
{
    "result": "create",
    "ok": true,
    "id": 1
}
```

Listing 2: create result (ok) message

```
{
    "result": "create",
    "ok": false,
    "error": "failed to create a new spreadsheet"
}
```

Listing 3: create result (error) message

3.3 Delete

The *delete* command is used to delete an existing spreadsheet. Delete will erase all the spreadsheet's history and remove it from the spreadsheet list. This command can fail if there is no spreadsheet that matches the provided *id*, or if an error occurs on the server. Deleting a spreadsheet will trigger an update to be sent to all clients who have the spreadsheet open informing them that the spreadsheet has been deleted.

3.3.1 Delete Command

In addition to the *command* field which will have a value of "delete", this command expects the following fields:

Field	Type	Description
id	number	The *id* of the spreadsheet to delete.

The following is an example of a *delete* command:

```
{
    "command": "delete",
    "id": 1
}
```

Listing 4: delete command message

3.3.2 Delete Results

The result of this command will contain the standard result fields. The following are examples of results of the *delete* command:

```
{
    "result": "delete",
    "ok": true,
}
```

Listing 5: delete result (ok) message

```
{
    "result": "delete",
    "ok": false,
    "error": "failed to delete the spreadsheet"
}
```

Listing 6: delete result (error) message

3.4 Rename

The *rename* command is used to rename an existing spreadsheet. Rename will change the *name* that is associated with the spreadsheet. This command can fail if there is no spreadsheet that matches the provided *id*, or if an error occurs on the server. Renaming a spreadsheet will trigger an update to be sent to all clients who have the spreadsheet open, informing them that the name of the spreadsheet has changed.

3.4.1 Rename Command

In addition to the *command* field which will have a value of "rename", this command expects the following fields:

Field	Type	Description
id	number	The *id* of the spreadsheet to rename.
name	string	The new *name* of the spreadsheet.

The following is an example of a *rename* command:

```
{
    "command": "rename",
    "id": 1,
    "name": "sheet1-old"
}
```

Listing 7: rename command message

3.4.2 Rename Results

The result of this command will contain the standard result fields. The following are examples of results of the *rename* command:

```
{
    "result": "rename",
    "ok": true
}
```

Listing 8: rename result (ok) message

```
{
    "result": "rename",
    "ok": false,
    "error": "failed to rename the spreadsheet"
}
```

Listing 9: rename result (error) message

3.5 List Spreadsheets

The *list_spreadsheets* command is used to retrieve the ids of all the spreadsheets stored on the server. This command will only fail if an error occurs on the server.

3.5.1 List Spreadsheets Command

The *command* field which will have a value of "list_spreadsheets", and will the *Command* will not need to contain any other fields.

The following is an example of a *list_spreadsheets* command:

```
{
    "command": "list_spreadsheets"
}
```

Listing 10: list command message

3.5.2 List Spreadsheets Results

The result of this command will contain the standard result fields, in addition to the following fields to be included for ok results:

Field	Type	Description
spreadsheets	string[]	The *names* of the spreadsheets, where the index is their id.

The following are examples of results of the *list_spreadsheets* command:

```
{
    "result": "list_spreadsheets",
    "ok": true,
    "spreadsheets": [
        "sheet1", "sheet2", "sheet3", "sheet4"
    ]
}
```

Listing 11: list result (ok) message

```
{
    "result": "list_spreadsheets",
    "ok": false,
    "error": "failed to read spreadsheets"
}
```

Listing 12: list result (error) message

3.6 Open

The *open* command is used to subscribe to updates for a specific spreadsheet. This command will cause the server to add the current TCP connection with the client to a list of subscribers that will receive updates associated with a specific sheet. If a client fails to receive any update, it will be removed from the subscribers list and will need to reopen the spreadsheet to begin receiving updates again. The server will respond to the open command with the current contents of the spreadsheet in the same way it does for the get_spreadsheet command. This command can fail if there is no spreadsheet that matches the provided *id*, or if an error occurs on the server.

3.6.1 Open Command

In addition to the *command* field which will have a value of "open", this command expects the following fields:

Field	Type	Description
id	number	The *id* of spreadsheet to open.

The following is an example of a *open* command:

```
{
    "command": "open",
    "id": 1
}
```

Listing 13: open command message

3.6.2 Open Results

The result of this command will contain the standard result fields, in addition to the following fields to be included for ok results:

Field	Type	Description
spreadsheet	Spreadsheet	The contents of the opened spreadsheet.

The following are examples of results of the *open* command:

```
{
    "result": "open",
    "ok": true,
    "spreadsheet": {
        "A1": "foo",
        "A2": "bar"
    }
}
```

Listing 14: open result (ok) message

```
{
    "result": "open",
    "ok": false,
    "error": "Spreadsheet does not exist"
}
```

Listing 15: open result (error) message

3.6.3 Updates

There are three variants of the updates that clients will receive for the spreadsheets they have open: *delete*, *rename* and *edits*. Each update message will contain an *update* field which will identify the type of update and an *id* field which will identify the spreadsheet the update is for. The *delete* update will contain only the *update* and *id* fields which indicate that the spreadsheet has been deleted. The *rename* update will contain the standard fields, as well as a *name* field which will contain the new name of the spreadsheet. The *edits* update will contain a *edits* field which will contain an array of Edit objects in order from oldest to newest edits. The update will contain all edits that have been made since the last update. The update will also contain a *start_id* field which will contain the edit id, of the first edit, each of the following edit ids can be computed by incrementing the edit id. The following are examples of the three variants of the update messages:

```
{
    "update": "delete",
    "id": 1
}
```

Listing 16: delete update message

```
{
    "update": "rename",
    "name": "sheet1-old"
}
```

Listing 17: rename update message

```
{
    "update": "edits",
    "id": 1,
    "start_id": 4,
    "edits": [
        {
            "A1": "foo"
        }
    ]
```

```
}
```

Listing 18: edits update message

3.7 Close

The *close* command is used to unsubscribe from updates for a specific spreadsheet. This command will cause the server to remove the current TCP connection with the client from its subscribers list. This command can fail if there is no spreadsheet that matches the provided id, or if an error occurs on the server.

3.7.1 Close Command

In addition to the *command* field which will have a value of "close", this command expects the following fields:

Field	Type	Description
id	number	The *id* of spreadsheet to close.

The following is an example of a *close* command:

```
{
    "command": "close",
    "id": 1
}
```

Listing 19: close command message

3.7.2 Close Results

The result of this command will contain the standard result fields. The following are examples of results of the *close* command:

```
{
    "result": "close",
    "ok": true
}
```

Listing 20: close result (ok) message

```
{
    "result": "close",
    "ok": false,
    "error": "Spreadsheet is not open"
}
```

Listing 21: close result (error) message

3.8 Get History

The *get_history* command is used to retrieve the history of a spreadsheet. This command will return the complete history of the spreadsheet separated into edits ordered from oldest to newest. This returned history may contain edits that have been undone, so any edit that comes after the current edit has been undone. This command can fail if there is no spreadsheet that matches the provided *id*, or if an error occurs on the server.

3.8.1 Get History Command

In addition to the *command* field which will have a value of "get_history", this command expects the following fields:

Field	Type	Description
id	number	The *id* of the spreadsheet whose history to get

The following is an example of a *get_history* command:

```
{
    "command": "get_history",
    "id": 1
}
```

Listing 22: get_history command message

3.8.2 Get History Results

The result of this command will contain the standard result fields, in addition to the following fields to be included for ok results:

Field	Type	Description
edits	Edit[]	The edit history in order from oldest to newest.
current	number	The zeroth based index of the current state of the spreadsheet.

The following are examples of results of the *get_history* command:

```
{
    "command": "get_history",
    "ok": true,
    "edits": [
        {
            "A1": 1,
            "B1": "=A1"
        },
        {
            "A2": "bar"
        }
    ],
    "current": 1
}
```

Listing 23: get_history result (ok) message

```
{
    "result": "get_history",
    "ok": false,
    "error": "Spreadsheet does not exist"
}
```

Listing 24: get_history result (error) message

3.9 Get Spreadsheet

The *get_spreadsheet* command is used to retrieve the contents of a spreadsheet. This command will return the complete contents of the spreadsheet in its current state. This command can fail if there is no spreadsheet that matches the provided *id*, or if an error occurs on the server.

3.9.1 Get Spreadsheet Command

In addition to the *command* field which will have a value of "get_spreadsheet", this command expects the following fields:

Field	Type	Description
id	number	The *id* of the spreadsheet whose contents to get.

The following is an example of a *get_spreadsheet* command:

```
{
    "command": "get_spreadsheet",
    "id": 1
}
```

Listing 25: get_spreadsheet command message

3.9.2 Get Spreadsheet Results

The result of this command will contain the standard result fields, in addition to the following fields to be included for ok results:

Field	Type	Description
spreadsheet	Spreadsheet	The contents of the requested spreadsheet.
current_edit	number	The id of the most recent edit.

The following are examples of results of the *get_spreadsheet* command:

```
{
    "result": "get_spreadsheet",
    "ok": true,
    "current_edit": 4,
    "spreadsheet": {
        "A1": "foo",
        "A2": "bar"
    }
}
```

Listing 26: get_spreadsheet result (ok) message

```
{
    "result": "get_spreadsheet",
    "ok": false,
    "error": "Spreadsheet does not exist"
}
```

Listing 27: get_spreadsheet result (error) message

3.10 Undo

The *undo* command is used to undo the most recent edit. This command will rollback the history of the spreadsheet, but will not erase the edit that was rolled-back so that it can still be accessed by get_history. Undo will trigger an update to be sent to all clients who have the spreadsheet open informing them that an edit has been made. While an undo is not actually an edit, for the purpose of updating the clients, undoes will be treated as if the reapply the previous edit. This command can fail if there is no spreadsheet that matches the provided *id*, or if an error occurs on the server.

3.10.1 Undo Command

In addition to the *command* field which will have a value of "undo", this command expects the following fields:

Field	Type	Description
id	number	The *id* of the spreadsheet to perform the undo on.

The following is an example of a *undo* command:

```
{
    "command": "undo",
    "id": 1
}
```

Listing 28: undo command message

3.10.2 Undo Results

The result of this command will contain the standard result fields. The following are examples of results of the *undo* command:

```
{
    "result": "undo",
    "ok": true
}
```

Listing 29: undo result (ok) message

```
{
    "result": "undo",
    "ok": false,
    "error": "Spreadsheet does not exist"
}
```

Listing 30: undo result (error) message

3.11 Push

The *push* command is used to push edits to the server so that they can be propagated to all other clients. The pushed edits will be merged into the spreadsheet according to the provided *from_id*, which specifies the point in spreadsheet's history these edits where made. The server will handle the merging of histories, to allow for synchronizing after going offline. Pushed edits must be valid in that they do not cause a circular dependency or formula error. Push will trigger an update to be sent to all clients who have the spreadsheet open informing them that an edit has been made. This command can fail if there is no spreadsheet that matches the provided *id*, the pushed edits are not valid or if an error occurs on the server.

3.11.1 Push Command

In addition to the *command* field which will have a value of "push", this command expects the following fields:

Field	Type	Description
id	number	The *id* of the spreadsheet to perform the push on.
from_id	number	The *id* of the edit preceding the provided edits.
edits	Edit[]	The edits to apply to the spreadsheet.

The *from_id* specifies the id of the most recent edit received from the server, this allows clients to push changes based on an earlier version of the spreadsheet. This edit id will most likely come from spreadsheet updates, a call to get_spreadsheet

The following is an example of a *push* command:

```
{
    "command": "push",
    "id": 1,
    "from_id": 4,
    "edits": [
        {
            "A1": "=A2",
            "A2": 1
        },
        {
            "A2": "=A1",
            "A1": 1
        }
    ]
}
```

Listing 31: push command message

3.11.2 Push Results

The result of this command will contain the standard result fields. The following are examples of results of the *push* command:

```
{
    "result": "push",
    "ok": true
}
```

Listing 32: push result (ok) message

```
{
    "result": "push",
    "ok": false,
    "error": "Edits cause a circular dependency"
}
```

Listing 33: push result (error) message