February 15, 2018

Spreadsheets with programmatically accessible version history

Tal Cohen

Spreadsheets with programmatically accessible version history

ABSTRACT

Spreadsheets enable users to store and track values of a variable. To graph changes in values of a variable over time, a user typically stores in the spreadsheet values of the variable at different time points. Retaining all values in individual, visible cells of a spreadsheet can make the spreadsheet difficult to read and/or maintain.

Modern spreadsheet programs include features to track changes made to the cells of a spreadsheet, e.g., as a version history. The version history enables a user to revert to a prior version. Techniques of this disclosure enable programmatic access to historical values stored in the cells of a spreadsheet. The techniques eliminate the need for a user to explicitly store multiple values for a variable over a time period. Spreadsheets that implement the techniques enable users to generate a graph by programmatically accessing historical values of the variable in the spreadsheet.

KEYWORDS

- Spreadsheet
- Track change
- Version history
- Revision history
- Graph
- Time series

BACKGROUND

The problem of accessing values of a variable as it changes with time is illustrated with an example, as follows. A project manager keeps track of a list of projects named “Alpha,”

"Bravo," "Charlie," etc. in a spreadsheet as shown in Fig. 1. The spreadsheet indicates the number of engineers and designers assigned to each project.

	A	B	C
1	Project	# of engineers	# of designers
2	Alpha	5	1
3	Bravo	3	3
4	Charlie	7	0
5	Delta	12	2

Fig. 1: An example spreadsheet

As engineers and designers are added to or removed from different projects, values in the spreadsheet illustrated in Fig. 1 are updated. If a user of the spreadsheet, e.g., the project manager wishes to chart these values with time, the historical values are not directly available in the spreadsheet. In order to generate the chart, the manager needs to use a different spreadsheet format that makes historical data readily available, e.g., as illustrated in Fig. 2. That includes a separate row for each date.

	A	B	C	D
1	Project	As of date...	# of engineers	# of designers
2	Alpha	2017-04-01	5	1
3	Alpha	2017-05-12	6	1
4	Alpha	2017-05-22	6	2
5	Alpha	2017-08-10	7	2
6	Bravo	2017-04-01	3	3
7	Bravo	2017-05-12	3	2
8	Bravo	2017-05-22	4	2
9	Bravo	2017-08-10	4	3
10	Charlie	2017-04-01	7	0
11	Charlie	2017-05-12	6	1
12	Charlie	2017-05-22	6	2
13	Charlie	2017-08-10	5	2
14	Delta	2017-04-01	12	2
15	Delta	2017-05-12	13	2
16	Delta	2017-05-22	13	3
17	Delta	2017-08-10	14	3

Fig. 2: Spreadsheet that stores historical data

The spreadsheet format of Fig. 2 is significantly more complicated than that of Fig. 1 due to the inclusion of historical values of variables and associated dates. While such a spreadsheet format can be used for the generation of a chart, e.g., as shown in Fig. 3, the spreadsheet is more difficult to read.

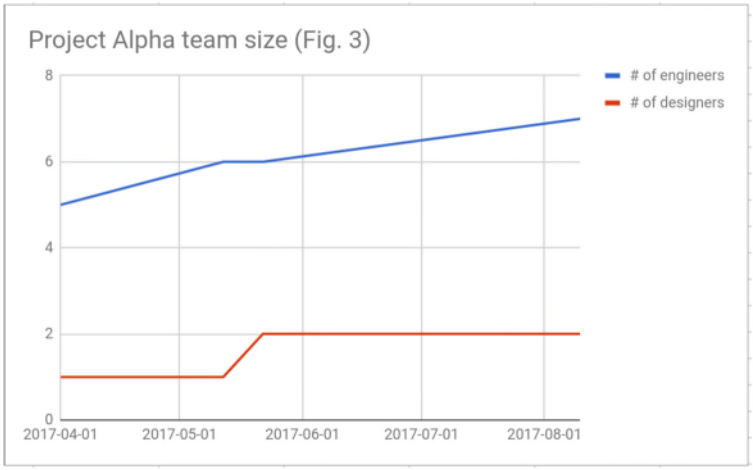

Fig. 3: Graph of project alpha team size for the data of Fig. 2

Further, explicit inclusion of historical data, e.g., as shown in Fig. 2, also makes the spreadsheet more difficult to maintain. For example, each time the number of designers or engineers in a project changes, rather than simply update a single cell in the spreadsheet, a new row needs to be added. The addition of a new row adds a maintenance burden and can also lead to errors, e.g., if the date-field is entered manually when a new row is added.

DESCRIPTION

Modern spreadsheet applications include features to maintain a history of changes made to a spreadsheet, if the user permits such storage. With specific user permission, the history of changes made to a spreadsheet is maintained. The history enables users to view one or more prior versions of the spreadsheet and/or to revert to a particular prior version.

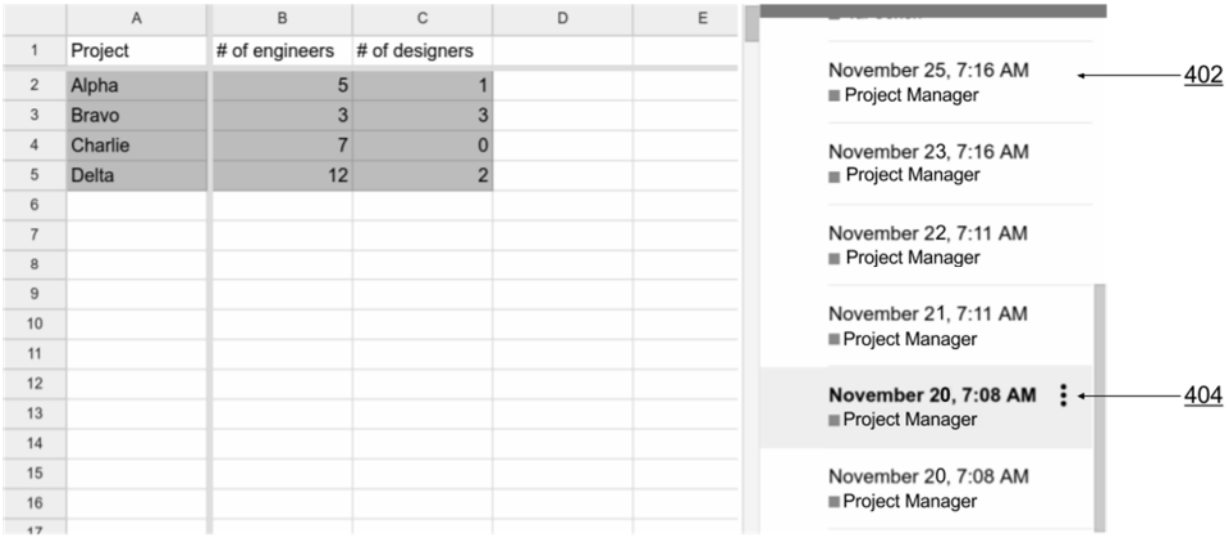

Fig. 4: Version history of a spreadsheet

Fig. 4 illustrates an example of a spreadsheet program that maintains a history of changes made to a spreadsheet, upon user permission. As seen in Fig. 4, the current version of the spreadsheet has time-stamp "November 25, 7:16 AM" (402). The version history enables the user, e.g., indicated by the user name "Project Manager," to access prior versions. As seen in Fig. 4, the user is currently viewing an earlier version associated with the timestamp "November 20, 7:08 AM" (404).

Techniques of this disclosure make historical data (stored as version history) accessible within the spreadsheet program itself, e.g., via spreadsheet functions or code within the spreadsheet. Such access to historical data enables, for example, the creation of charts of cell-values versus time.

Some examples of spreadsheet functions that enable access to the value of a cell at given points of time are as follows:

- `values = ValueAtTime(range, time)`, where
  `range` is an index of a cell or cells;

`time` is a time-point for which values of cells comprising `range` are to be retrieved;

`values` is a returned parameter comprising values of cells within `range` and `time`.

- `values = ValuesOverTime(range, start_time, end_time)`, where
  `range` is an index of a cell or cells;
  `start_time` is the start of a period for which values of cells comprising `range` are to be retrieved;
  `end_time` is the end of a period for which values of cells comprising `range` are to be retrieved; and
  `values` is a returned parameter comprising values of cells within `range`, `start_time` and `end_time`.

The size of `values`, the parameter returned by this function, varies based on the number of changes made. For example, if no changes were made, `values` contains a single cell; if five changes were made, `values` contains six cells (the original value, plus the five updated ones); etc.

- `timestamps = ChangeTimes(range, start_time, end_time)`, where
  `range` is an index of a cell or cells;
  `start_time` is the start of a period for which values of cells comprising `range` are sought;
  `end_time` is the end of a period for which values of cells comprising `range` are sought; and
  `timestamps` is a returned parameter that is a list comprising the times at which values within `range` changed.

The size of `timestamps` is identical to the size of the returned value for a call to `ValuesOverTime()` with identical arguments.

Examples of use

The examples below illustrate the use of the above functions.

Example 1: `ValueAtTime(B3, DATE(2017, 10, 01))` returns the value of cell B3 as of October 1st, 2017.

Example 2: `ValueAtTime(A2:C5, NOW()-2)` returns the values of cells A2 through C5 as of two days ago.

Example 3: `ValuesOverTime(A2, DATE(2017, 01, 01), NOW())` returns the values of cell A2 since January 1st, 2017 until current time.

Example 4: Cell A2 has been changed five times since inception on 1st January 2017, namely on 1st February 2017, 1st March 2017, 1st April 2017, 1st May 2017, and 1st June 2017. The present date is 2nd June 2017. A call to `ChangeTimes()` thus:

```
ChangeTimes(A2, DATE(2017, 01, 01), NOW())
```

returns a six-element list comprising the start date plus the dates of the five updates, e.g., [1-Jan-2017, 1-Feb-2017, 1-Mar-2017, 1-Apr-2017, 1-May-2017, 1-Jun-2017].

Example 5: Cell A2 has never been changed since inception on 1st January 2017. A call `ChangeTimes(A2, DATE(2017, 01, 01), NOW())` returns a single timestamp, 1-Jan-2017.

An alternative to spreadsheet functions includes spreadsheet syntax. For example, the functionality of `ValueAtTime(range, time)` can be made available via spreadsheet syntax as follows:

`range@time`, where

`range` is an index of a cell or cells; and

`time` is a time-point for which values of cells comprising `range` are sought.

Example 1: `B3@DATE(2017, 10, 01)` returns the values of cell B3 as of October 1st, 2017.

Example 2: `A2:C5@(NOW()-2)` returns the values of cells A2 through C5 as of two days ago.

Variations of spreadsheet syntax are similarly defined for other spreadsheet functions described above. In some situations, the spreadsheet function is a more reasonable approach to access historical values of the cells of a spreadsheet. A spreadsheet program that implements the techniques described herein can support programmatic access to historical data through spreadsheet functions, spreadsheet syntax, or both.

One application of the techniques described above is to chart historical data. For example, in the earlier example of a spreadsheet used to keep track of the number of engineers and designers assigned to various projects, the project manager can continue to use the relatively simple spreadsheet format of Fig. 1 even as the number of engineers/designers for a project changes over time.

A change in the number of engineers for project Alpha, is tracked by simply updating cell B2 of Fig. 1. With user permission, the spreadsheet program retains previous values of all cells, including B2, and respective timestamps associated with each change in value. To generate a graph of changes in the number of engineers/designers versus time (as in Fig. 3), the user can generate the X-axis (time-points) of Fig. 3 using the function below:

```
ChangeTimes(B2:C2, DATE(2017, 04, 01), NOW()).
```

Further, the user can generate the Y-axis (team size of project alpha) as follows:

`ValuesOverTime(B2, DATE(2017, 04, 01), NOW())` (for engineers, blue line of Fig. 3); and

`ValuesOverTime(C2, DATE(2017, 04, 01), NOW())` (for designers, red line of Fig. 3).

A spreadsheet that implements techniques of this disclosure supports such function calls and returns appropriate values based on stored version history. In the version history, cell B2 of Fig. 1 includes the number of engineers in the project, and cell C2 implicitly contains the number of designers, at different time points. A user no longer needs to list every change in value explicitly, since the version control feature of the spreadsheet application is leveraged to return historical values in response to a function call. The time-dimension of a cell of a spreadsheet is effectively retained, and is made programmatically available.

The functions and syntax disclosed herein can also be used for other purposes. For example, the following formula detects if the value in a cell A2 has changed in the past 24 hours:

`A2 <> ValueAtTime(A2, NOW() - 1)`, where

<> represents the is-different-from operator.

The above formula compares the value of the cell A2 to its value a day (24 hours) ago, and returns `TRUE` if the two values are different. Such a formula can be used for various purposes, e.g., conditional formatting or highlighting, other calculations, etc.

Further to the descriptions above, a user may be provided with controls allowing the user to make an election as to both if and when systems, programs or features described herein may enable collection of user information (e.g., information about a user's social network, social actions or activities, profession, a user's preferences, or a user's current location), and if the user is sent content or communications from a server. In addition, certain data may be treated in one or more ways before it is stored or used, so that personally identifiable information is removed. For example, a user's identity may be treated so that no personally identifiable information can be determined for the user, or a user's geographic location may be

generalized where location information is obtained (such as to a city, ZIP code, or state level), so that a particular location of a user cannot be determined. Thus, the user may have control over what information is collected about the user, how that information is used, and what information is provided to the user.

CONCLUSION

Techniques of this disclosure enable programmatic access to historical values stored in the cells of a spreadsheet, e.g., as part of a version history. The techniques eliminate the need for a user to explicitly store multiple values for a variable over a time period. Function-calls and syntax described herein enable programmatic access to historical values. The techniques enable a user to maintain relatively simple spreadsheets, with no burden for storing individual values of a variable at different time points. Functions and/or syntax are provided that enables users to retrieve historical values of cells or time-stamps of changes which can be used for various purposes, e.g., to chart cell-values versus time, for conditional formatting, etc.