

ADAMS

Advanced **D**ata mining **A**nd **M**achine learning **S**ystem

Module: adams-spreadsheet



Peter Reutemann

December 21, 2016

©2012-2016



THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato



Except where otherwise noted, this work is licensed under
<http://creativecommons.org/licenses/by-sa/4.0/>

Contents

1	Introduction	7
2	Flow	9
3	Tools	15
3.1	Filtering and processing	16
3.2	Plug-ins	16
3.2.1	View plug-ins	16
3.2.2	Data plug-ins	17
4	Formulas	19
5	Troubleshooting	23
5.1	Fractional times	23
	Bibliography	25

List of Figures

3.1	Viewer for spreadsheet files.	15
3.2	Row finder setup.	16
3.3	The filtered spreadsheet.	16

Chapter 1

Introduction

Tabular data is a very common data format, not only for machine learning. The *spreadsheet* module offers some basic spreadsheet support for reading/writing and some generic actors. Other modules, like the *odf* or *excel* one, offer other native readers and writers. The data read by these readers can be processed with the same actors.

Chapter 2

Flow

A lot of the following actors and conversion schemes offer either an index of a column or a range of columns as a parameter. In addition to the usual *first/last/...* placeholders, you can use the actual column names (case-insensitive). This makes an actor less error-prone, in case the order of the columns may change.

The following standalones are available:

- *LookupInit* – Initializes an empty lookup table in storage, which needs populating using the *LookupAdd* transformer.¹

The following sources are available:

- *DatabaseMetaData* – outputs spreadsheets with information obtained from the meta-data of a database connection.
- *NewSpreadSheet* – for creating an empty spreadsheet with pre-defined columns.
- *SpreadSheetDbReader* – turns results from SQL queries into spreadsheet objects.²
- *Lookup* – Outputs a value from a stored lookup table, identified by a user-supplied key.³
- *LookupTable* – Outputs a stored lookup table as spreadsheet.⁴

The following transformers are available:

- *Lookup* – obtains the value associated with the string received as input from an internally stored lookup table.⁵
- *LookupAdd* – adds a key/value pair to a lookup table.⁶
- *LookupInit* – initializes a lookup table by using two columns from a spreadsheet, one acting as key, the other as value.⁷
- *LookupRemove* – removes a key/value pair from a lookup table.⁸

¹adams-spreadsheet-lookup2.flow

²adams-spreadsheet-database.access.flow

³adams-spreadsheet-lookup.flow

⁴adams-spreadsheet-lookup.flow

⁵adams-spreadsheet-lookup.flow

⁶adams-spreadsheet-lookup.flow

⁷adams-spreadsheet-lookup.flow

⁸adams-spreadsheet-lookup.flow

- *SpreadSheetAggregate* – aggregates rows in a spreadsheet (min, max, average, standard deviation, etc).⁹
- *SpreadSheetAnonymize* – for anonymizing columns.¹⁰
- *SpreadSheetCellFinder* – outputs row/column pairs of cells that matched the specified criteria of the cell finder algorithm.
- *SpreadSheetCollapse* – collapses cell values of rows with the same key into single cell.
- *SpreadSheetColumnFilter* – filters columns using a column finder scheme.¹¹
- *SpreadSheetColumnIterator* – iterates over all the columns in the spreadsheet and outputs the names.¹²
- *SpreadSheetColumnStatistic* – generates statistics for a spreadsheet column.¹³
- *SpreadSheetColumnsByName* – generates a new spreadsheet with only the columns that match a regular expression (inverting is possible as well).
- *SpreadSheetCommonIDs* – determines common (or not in common) IDs from two or more spreadsheets.
- *SpreadSheetConvertCells* – applies arbitrary conversion schemes to individual cells.
- *SpreadSheetConvertHeaderCells* – applies arbitrary conversion schemes to the header cells.
- *SpreadSheetCopyColumns* – copies the content of a range of columns into new columns.
- *SpreadSheetCopyRows* – duplicates the content of a range of rows at another location in the spreadsheet.¹⁴
- *SpreadSheetDifference* – computes the difference between two spreadsheets.
- *SpreadSheetExtractArray* – allows the extraction of a row or column from a spreadsheet.
- *SpreadSheetFileReader* – for reading spreadsheet files; depending on the reader, multiple sheets can get read at once.¹⁵
- *SpreadSheetGetCell* – retrieves the value of a specific cell in the spreadsheet.¹⁶
- *SpreadSheetGetColumnIndex* – retrieves the 1-based index columns which name matches a regular expression.¹⁷
- *SpreadSheetGetHeaderCell* – retrieves the value of a specific cell in the header row of a spreadsheet.
- *SpreadSheetInfo* – generates basic information on the spreadsheet object.
- *SpreadSheetInsertColumn* – inserts a column in the spreadsheet, initializes the cells with a user-defined value.
- *SpreadSheetInsertRow* – inserts a row in the spreadsheet, initializes the cells with a user-defined value.

⁹adams-spreadsheet-aggregate.flow

¹⁰adams-spreadsheet-anonymize_columns.flow

¹¹adams-spreadsheet-filter_columns.flow

¹²adams-spreadsheet-iterate_cols.flow

¹³adams-spreadsheet-column_statistics.flow

¹⁴adams-spreadsheet-copy_rows.flow

¹⁵adams-spreadsheet-output_cells.flow

¹⁶adams-spreadsheet-output_cells.flow

¹⁷adams-spreadsheet-get_column_index.flow

- *SpreadSheetInsertRowScore* – inserts a column in the spreadsheet, containing a score calculate for the corresponding row.
- *SpreadSheetMatrixStatistic* – calculates statistics from the whole spreadsheet (or a defined subset).
- *SpreadSheetMerge* – merges multiple spreadsheets into a single one.¹⁸
- *SpreadSheetPlotGenerator* – turns a spreadsheet into plot containers to be displayed in the *SequencePlotter* sink.¹⁹
- *SpreadSheetQuery* – allows to run an SQL-like query on a spreadsheet to select a subset of rows/columns, delete rows, rename columns, update cells.²⁰
- *SpreadSheetRemoveColumn* – removes columns from a spreadsheet.
- *SpreadSheetRemoveRow* – removes rows from a spreadsheet.
- *SpreadSheetReorderColumns* – reorders columns in a spreadsheet, also allows duplicating/dropping of columns.
- *SpreadSheetReplaceCellValue* – replaces cell values that match a regular expression.
- *SpreadSheetRowBuffer* – buffers incoming row objects and outputs spreadsheets or outputs single rows when receiving spreadsheets.
- *SpreadSheetRowFilter* – filters rows using a row finder scheme.²¹
- *SpreadSheetRowStatistic* – generates statistics for a spreadsheet row.²²
- *SpreadSheetSetCell* – sets the value of a specific cell in the spreadsheet.²³
- *SpreadSheetSetHeaderCell* – sets the value of a specific cell in the header row of a spreadsheet.
- *SpreadSheetSort* – sorts a spreadsheet using an arbitrary number of columns (ascending or descending).
- *SpreadSheetStatistic* – calculates statistics using the data stored in the spreadsheet.²⁴
- *SpreadSheetSubset* – for obtaining a subset of the spreadsheet object (subset of columns and/or rows).
- *SpreadSheetSubsetByValue* – splits a spreadsheet into subsets using the unique (string) values in a column for grouping.
- *SpreadSheetStorageRowIterator* – iterates over a range of rows/columns and stores the cell values in internal storage.
- *SpreadSheetTransformCells* – applies an arbitrary callable transformer to individual cells.
- *SpreadSheetTransformHeaderCells* – applies an arbitrary callable transformer to the header cells.
- *SpreadSheetVariableRowIterator* – iterates over a range of rows/columns and stores the cell values in variables.²⁵

¹⁸adams-spreadsheet-simple_merge.flow, adams-spreadsheet-merge_using_id.flow

¹⁹adams-spreadsheet-spreadsheet_plot1.flow, adams-spreadsheet-spreadsheet_plot2.flow, adams-spreadsheet-statistic.flow

²⁰adams-spreadsheet-query.flow, adams-spreadsheet-date_queries.flow, adams-spreadsheet-celltype_queries.flow

²¹adams-spreadsheet-filter_rows.flow

²²adams-spreadsheet-row_statistics.flow

²³adams-spreadsheet-set_cells.flow

²⁴adams-spreadsheet-statistic.flow

²⁵adams-spreadsheet-variable_row_iterator.flow

The following sinks are available:

- *JFreeChartFileWriter* – generates plots from spreadsheet columns using JFreeChart[2] and writes them to disk.
- *JFreeChartPlot* – allows plotting of columns from spreadsheets using JFreeChart[2].
- *SpreadSheetDbWriter* – for storing a spreadsheet in a database.²⁶
- *SpreadSheetDisplay* – for displaying spreadsheet objects in tabular form.²⁷
- *SpreadSheetFileWriter* – writes spreadsheet objects to a file with the chosen writer class; depending on the writer either a single or multiple sheets can get written at once.
- *SpreadSheetRowViewer* – displays the numeric values of rows from a spreadsheet as line plots.

The following boolean conditions are available:

- *HasColumn* – checks whether the spreadsheet passing through has a column with that name.

The following conversion schemes are available:

- *ContainerToSpreadSheet* – for converting any flow container (e.g., a prediction container) into a spreadsheet for better visualization.
- *ConvertSpreadSheetRows* – converts the data rows in a spreadsheet into a different format, e.g., into sparse representation.
- *DoubleMatrixToSpreadSheet* – converts a two-dimensional double array (i.e., matrix) into a spreadsheet object.
- *MapToSpreadSheet* – turns a *java.util.Map* object into a spreadsheet.
- *NotesToSpreadSheet* – generates a spreadsheet from a Notes object.
- *PropertiesToSpreadSheet* – generates a spreadsheet from a Properties object.
- *RenameSpreadSheet* – renames a spreadsheet.
- *RenameSpreadSheetColumn* – renames a single column in a spreadsheet.
- *ReportToSpreadSheet* – generates a spreadsheet from a Report object.
- *SpreadSheetAddFormulaColumn* – adds a column with a row-wise, user-supplied formula.
- *SpreadSheetAddFormulaRow* – adds a row with a column-wise, user-supplied formula.
- *SpreadSheetAddRowID* – adds an ID column to the spreadsheet that contains the row index as value
- *SpreadSheetAddSumColumn* – adds a column with a row-wise sum formula.
- *SpreadSheetAddSumRow* – adds a row with a column-wise sum formula.
- *SpreadSheetAnyColumnToString* – converts any data types in a column to strings.
- *SpreadSheetCellLocationToCoordinates* – turns a cell location into an integer array of 1-based coordinates
- *SpreadSheetCellLocationToPosition* – turns a cell location obtained from a cell finder into a position string (eg 'A2').

²⁶adams-spreadsheet-database_access.flow

²⁷adams-spreadsheet-display.flow

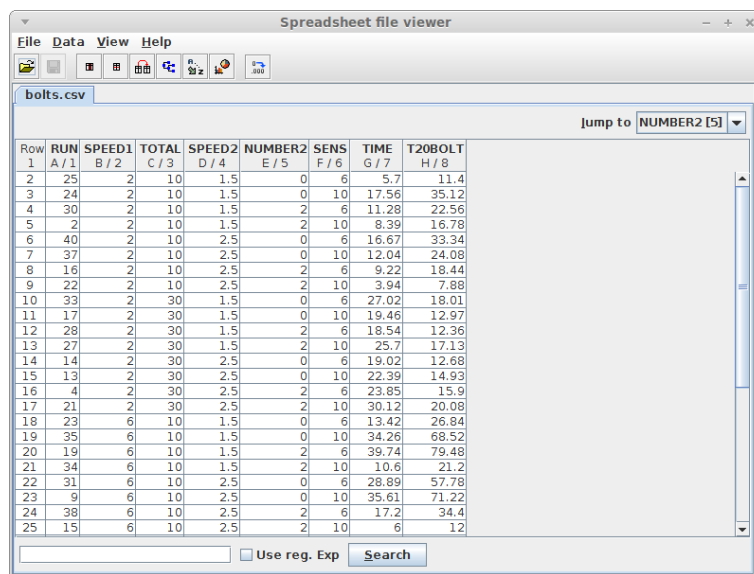
- *SpreadSheetColumnFinderToRange* – outputs a range string of the columns that the finder located (eg '1-6,8').
- *SpreadSheetDoubleColumnToLong* – converts any floating values to integer (= long) values.
- *SpreadSheetDoubleColumnToString* – converts any floating values to strings.
- *SpreadSheetEscapeColumnName* – ensures that a column is escaped correctly in order to be used in column range expressions.
- *SpreadSheetInsertCellLocation* – replaces a placeholder in a string with a cell location (e.g., "A1").
- *SpreadSheetJoinColumns* – joins two or more columns in a spreadsheet into a single one.
- *SpreadSheetLongColumnToDouble* – converts long values in a column to double objects.
- *SpreadSheetMaterializeFormulas* – replaces formulas in cells with the current value of the formula.
- *SpreadSheetRowFinderToRange* – outputs a range string of the rows that the finder located (eg '1-6,8').
- *SpreadSheetStringSplitColumn* – splits the string representation of a column into multiple columns using a regular expression.
- *SpreadSheetStringColumnToBoolean* – converts strings values in a column to boolean objects.
- *SpreadSheetStringColumnToDate* – converts strings values in a column to date objects.
- *SpreadSheetStringColumnToDateTime* – converts strings values in a column to date/time objects.
- *SpreadSheetStringColumnToDouble* – converts strings values in a column to floating values.
- *SpreadSheetStringColumnToLong* – converts strings values in a column to integer (= long) values.
- *SpreadSheetStringColumnToTime* – converts strings values in a column to time objects.
- *SpreadSheetToCreateTableStatement* – turns a spreadsheet with SQL column names and types into a SQL 'CREATE TABLE' statement.
- *SpreadSheetToDoubleMatrix* – turns all the numeric columns of a spreadsheet into a two-dimensional double matrix.
- *SpreadSheetToJson* – turns a spreadsheet into a JSON array object.
- *SpreadSheetToMap* – uses two columns from a spreadsheet (key and value) to populate a *java.util.Map* object.
- *SpreadSheetToStringMatrix* – turns all columns of a spreadsheet into a two-dimensional string matrix.
- *SpreadSheetUnescapeColumnName* – reverses the escaping for a column name.
- *SpreadSheetUniqueColumnNames* – ensures that the column names uniquely identify a column.
- *StringToSpreadSheet* – parses a string in CSV format and turns it into a spreadsheet object.
- *StringMatrixToSpreadSheet* – turns a (two-dimensional) string matrix into a spreadsheet.

- *TransposeSpreadSheet* – swaps columns with rows.

Chapter 3

Tools

The *Spreadsheet file viewer* is a simple tool for loading all spreadsheet file formats that ADAMS supports. Despite its name, the tool also allows you to modify cell values and save them back to a file. Figure 3.1 shows a dataset that was loaded from a CSV (comma-separated values) file.



The screenshot shows a window titled "Spreadsheet file viewer" with a menu bar (File, Data, View, Help) and a toolbar. The file "bolts.csv" is open. A "Jump to" dropdown menu is set to "NUMBER2 [5]". The main area displays a table with 25 rows and 10 columns. The columns are labeled: Row, RUN, SPEED1, TOTAL, SPEED2, NUMBER2, SENS, TIME, T20BOLT, and an unlabeled column. The data is as follows:

Row	RUN	SPEED1	TOTAL	SPEED2	NUMBER2	SENS	TIME	T20BOLT	
1	A / 1	B / 2	C / 3	D / 4	E / 5	F / 6	G / 7	H / 8	
2	25	2	10	1.5	0	6	5.7	11.4	
3	24	2	10	1.5	0	10	17.56	35.12	
4	30	2	10	1.5	2	6	11.28	22.56	
5	2	2	10	1.5	2	10	8.39	16.78	
6	40	2	10	2.5	0	6	16.67	33.34	
7	37	2	10	2.5	0	10	12.04	24.08	
8	16	2	10	2.5	2	6	9.22	18.44	
9	22	2	10	2.5	2	10	3.94	7.88	
10	33	2	30	1.5	0	6	27.02	18.01	
11	17	2	30	1.5	0	10	19.46	12.97	
12	28	2	30	1.5	2	6	18.54	12.36	
13	27	2	30	1.5	2	10	25.7	17.13	
14	14	2	30	2.5	0	6	19.02	12.68	
15	13	2	30	2.5	0	10	22.39	14.93	
16	4	2	30	2.5	2	6	23.85	15.9	
17	21	2	30	2.5	2	10	30.12	20.08	
18	23	6	10	1.5	0	6	13.42	26.84	
19	35	6	10	1.5	0	10	34.26	68.52	
20	19	6	10	1.5	2	6	39.74	79.48	
21	34	6	10	1.5	2	10	10.6	21.2	
22	31	6	10	2.5	0	6	28.89	57.78	
23	9	6	10	2.5	0	10	35.61	71.22	
24	38	6	10	2.5	2	6	17.2	34.4	
25	15	6	10	2.5	2	10	6	12	

At the bottom, there is a search bar with a "Use reg. Exp" checkbox and a "Search" button.

Figure 3.1: Viewer for spreadsheet files.

If there are more spreadsheet file formats registered, you can save the currently displayed spreadsheet in another format. Printing, of course, is available through the *Send to* menu. By default, the viewer displays each cell with as many digits after the decimal point as necessary. But you can also unify this and specify how many digits should be used for all floating point cells.

3.1 Filtering and processing

The viewer supports some basic filtering and processing:

- *columns* – creates a subset of the spreadsheet by selecting a subset of columns, e.g., all columns which name starts with a certain string.
- *rows* – creates a subset of the spreadsheet by selecting a subset of rows, e.g., rows with a certain value in a column.
- *convert* – applies a conversion scheme specific to spreadsheets, e.g., transposing a spreadsheet.
- *transform* – applies a flow transformer specific to spreadsheets, e.g., inserting a column or creating a subset.

Figures 3.2 and 3.3 show the setup for a row finder filter and the resulting new spreadsheet.

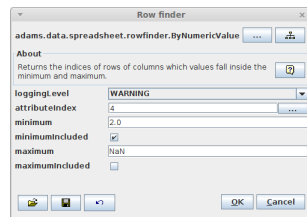


Figure 3.2: Row finder setup.

Row	RUN	SPEED1	TOTAL	SPEED2	NUMBER2	SENS	TIME	T200BOLT
1	40	2	10	2.5	0	6	16.67	33.34
2	37	2	10	2.5	0	10	12.04	24.08
3	16	2	10	2.5	2	6	9.22	18.44
4	22	2	10	2.5	2	10	3.94	7.88
5	14	2	30	2.5	0	6	19.02	12.68
6	13	2	30	2.5	0	10	22.39	14.93
7	4	2	30	2.5	2	6	23.85	15.9
8	21	2	30	2.5	2	10	30.12	20.08
9	31	6	10	2.5	0	6	28.89	57.78
10	9	6	10	2.5	0	10	35.61	71.22
11	15	6	10	2.5	2	6	17.2	34.4
12	15	6	10	2.5	2	10	6	12
13	6	6	30	2.5	0	6	100.43	66.95
14	20	6	30	2.5	0	0	109.28	72.85
15	10	6	30	2.5	2	6	106.46	70.97
16	32	6	30	2.5	2	0	134.01	89.34
17	1	4	20	2	1	8	10.78	10.78
18	3	4	20	2	1	8	9.39	9.39
19	5	4	20	2	1	8	9.84	9.84
20	7	4	20	2	1	8	13.94	13.94
21	12	4	20	2	1	8	12.33	12.33
22	18	4	20	2	1	8	7.32	7.32
23	29	4	20	2	1	8	7.91	7.91
24	36	4	20	2	1	8	15.58	15.58

Figure 3.3: The filtered spreadsheet.

3.2 Plug-ins

The viewer can be extended with two sorts of plug-ins:

- ones that generate a view based on the current sheet (“view”)
- ones that process the current sheet (“data”)

3.2.1 View plug-ins

A view plug-in is derived from the following super-class:

```
adams.gui.tools.spreadsheetviewer.AbstractViewPlugin
```

There are three methods that need implementing:

- *getMenuText()* – returns the text used for the menu item and the title of the dialog displaying the generated view.
- *getMenuIcon()* – returns the name of the icon (no path) that should be displayed in the menu (use null to display no icon).

- *doGenerate(SpreadSheet)* – this method generates the actual view in form of a *adams.gui.core.BasePanel*.

An example is the *Statistics* plug-in, which shows simple statistics for a spreadsheet, number of rows and columns and what types of columns are present:

```
adams.gui.tools.spreadsheetviewer.Statistics
```

Further superclasses:

- *AbstractSelectedSheetsViewPlugin* – for view plugins that operate on one or more spreadsheets that the user selects.
- *AbstractSelectedSheetsViewPluginWithGOE* – same as *AbstractSelectedSheetsViewPlugin* but offers the user to change the settings through a *GenericObjectEditor* view.

3.2.2 Data plug-ins

A view plug-in is derived from the following super-class:

```
adams.gui.tools.spreadsheetviewer.AbstractDataPlugin
```

There are four methods that need implementing:

- *getMenuText()* – returns the text used for the menu item and the title of the dialog displaying the generated view.
- *getMenuIcon()* – returns the name of the icon (no path) that should be displayed in the menu (use null to display no icon).
- *doProcess(SpreadSheet)* – this method processes the current spreadsheet and returns a new spreadsheet object.
- *isInPlace()* – returns whether the generated spreadsheet object should simply replace the current one (“in-place”) or added as new tab.

Further superclasses:

- *AbstractSelectedSheetsDataPlugin* – for data plugins that operate on one or more spreadsheets that the user selects (see *Append* plugin).
- *AbstractSelectedSheetsDataPluginWithGOE* – same as *AbstractSelectedSheetsDataPlugin* but offers the user to change the settings through a *GenericObjectEditor* view (see *Merge* plugin).

Chapter 4

Formulas

ADAMS supports formulas with a range of basic functions. The following lists describe briefly what functionality is available. A full list is available through the *Help -> Formulas* menu in the *Spreadsheet file viewer*.

Operands:

- $NUM + NUM$ – addition
- $NUM - NUM$ – subtraction
- $NUM * NUM$ – multiplication
- NUM / NUM – division
- $NUM \wedge NUM$ – exponential
- $NUM \% NUM$ – modulo

Boolean operations:

- $<$ – less than
- $<=$ – less than or equal
- $>$ – greater than
- $>=$ – greater than or equal
- $=$ – equals
- $!=$ – does not equal (alternative: $<>$)
- $!$ – negation
- $\&$ – and
- $|$ – or

Numeric functions:

- *abs* – absolute value of a cell/number.
- *average* – average computed from a range of cells.
- *ceil* – smallest value that is greater than or equal to the cell/number and is equal to a mathematical integer.
- *cos* – the trigonometric cosine of a number/cell.
- *countblank* – counts empty/missing value cells in a range of cells.
- *countif* – counts value only a condition is true.
- *exp* – Euler's number e raised to the power of a number/cell.

- *floor* – largest value that is less than or equal to the cell/number and is equal to a mathematical integer.
- *if[else]* – if-then-else construct.
- *intercept* – compute the intercept of linear regression between two cell ranges.
- *log* – natural logarithm (base e) of a number/cell.
- *max* – largest value from range of cells.
- *min* – smallest value from range of cells.
- *pow[er]* – the first number/cell raised to the power of the second number/cell.
- *rint* – returns number that is closest in value to the number/cell and is equal to a mathematical integer.
- *sin* – trigonometric sine of a number/cell.
- *slope* – compute the slope of linear regression between two cell ranges.
- *sqrt* – the correctly rounded positive square root of a number/cell.
- *stdev* – the sample standard deviation from a range of cells.
- *stdevp* – the population standard deviation from a range of cells.
- *sum* – the sum over a range of cells.
- *sumif* – the conditional sum over a range of cells.
- *tan* – trigonometric tangent of a number/cell.

String functions:

- *concatenate* – concatenates up to 5 strings.
- *find* – returns the location of a search string in a string.
- *left* – returns substring of specified length from the left of the string.
- *len[gh]* – the length of a string.
- *lower[case]* – converts string to a lowercase one.
- *matches* – matches a string against a regular expression.
- *mid* – returns substring of specified length from the specified position in the string.
- *right* – returns substring of specified length from the right of the string.
- *replace* – replaces a substring at a specified position with a new string.
- *rept* – returns a string made up of X copies of the supplied string.
- *trim* – removes all leading and trailing whitespaces.
- *substr* – creates a substring from a string, given start/end position.
- *substitute* – replaces occurrences of a search string in a string with a new string.
- *upper[case]* – converts string to an uppercase one.

Date/time functions:

- *year* – extracts the year from a date/time cell.
- *month* – extracts the month from a date/time cell.
- *day* – extracts the day from a date/time cell.
- *hour* – extracts the hour from a date/time cell.
- *minute* – extracts the minute from a date/time cell.
- *second* – extracts the second from a date/time cell.

- *weekday* – extracts the weekday from a date/time cell (Sunday=1, Saturday=7).
- *weeknum* – extracts the week number from a date/time cell.

Chapter 5

Troubleshooting

5.1 Fractional times

Despite database systems supporting fractional times, i.e., times with fractional seconds (= milliseconds), the JDBC interface does not support this (at this stage). This results in fractions always getting set to 0.

However, JDBC drivers, like the MySQL one starting with 5.1.37, may support in some cases the sending of fractional seconds to the server where they may be subject to rounding (for MySQL use `sendFractionalSeconds=true|false` in the JDBC URL).

In order to avoid the truncating of fractions, you can do two things:

- Create your own queries and have them executed using the *ExecSQL* standalone - this bypasses the JDBC driver's checks for fractions.
- Turn the columns in the spreadsheet, when using the *SpreadSheetDbWriter*, into strings. They will get converted automatically, at least in the case of MySQL, automatically into fractional times again before being inserted. Either use the *SpreadSheetAnyColumnToString* conversion or use `concat("", column_name)` to return the column *column_name* as string from the database in the first place.

Bibliography

- [1] *ADAMS* – Advanced Data mining and Machine learning System
<https://adams.cms.waikato.ac.nz/>
- [2] *JFreeChart* – is a free 100% Java chart library that makes it easy for developers to display professional quality charts in their applications.
<http://www.jfree.org/jfreechart/>