TIBCO PSG

ActiveSpaces Tools

User’s Guide

Version 0.8.0

Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| 0.7.0 | September 11, 2020 | Pierre Ayel | First Version |
| 0.7.1 | October 7, 2020 | Pierre Ayel | Updated for version 0.7.1:   * New tool TableDelete * Changed datetime-filter default value to false. |
| 0.7.2 | October 23, 2020 | Pierre Ayel | Updated for version 0.7.2:   * TableExport support for datagrid checkpoints |
| 0.8.0 | July 08, 2022 | Pierre Ayel | Updated for version 0.8.0:   * TableImport support for double quotes in input CSV Files, * Changed default columns in TableExport |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Copyright Notice

COPYRIGHT© 2016 TIBCO Software Inc. This document is unpublished and the foregoing notice is affixed to protect TIBCO Software Inc. in the event of inadvertent publication. All rights reserved. No part of this document may be reproduced in any form, including photocopying or transmission electronically to any computer, without prior written consent of TIBCO Software Inc. The information contained in this document is confidential and proprietary to TIBCO Software Inc. and may not be used or disclosed except as expressly authorized in writing by TIBCO Software Inc. Copyright protection includes material generated from our software programs displayed on the screen, such as icons, screen displays, and the like.

Trademarks

All brand and product names are trademarks or registered trademarks of their respective holders and are hereby acknowledged. Technologies described herein are either covered by existing patents or patent applications are in progress.

Confidentiality

The information in this document is subject to change without notice. This document contains information that is confidential and proprietary to TIBCO Software Inc. and its affiliates and may not be copied, published, or disclosed to others, or used for any purposes other than review, without written authorization of an officer of TIBCO Software Inc. Submission of this document does not represent a commitment to implement any portion of this specification in the products of the submitters.

Content Warranty

The information in this document is subject to change without notice. THIS DOCUMENT IS PROVIDED "AS IS" AND TIBCO MAKES NO WARRANTY, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO ALL WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. TIBCO Software Inc. shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Export

This document and related technical data, are subject to U.S. export control laws, including without limitation the U.S. Export Administration Act and its associated regulations, and may be subject to export or import regulations of other countries. You agree not to export or re-export this document in any form in violation of the applicable export or import laws of the United States or any foreign jurisdiction.

For more information, please contact:

TIBCO Software Inc.  
3303 Hillview Avenue  
Palo Alto, CA 94304  
USA

**Table of Contents**

1 Introduction 6

1.1 Overview 6

1.2 Known Issues 6

2 Installation 9

2.1 Software Location 9

2.2 Installation on Windows 9

2.3 Installation on Unix 9

3 Common Options 10

3.1 <config-file-options> 10

3.2 <datagrid-options> 11

3.3 SSL Properties 13

3.4 Advanced Properties 13

4 TableExport 14

4.1 Description 14

4.2 Features 14

4.3 Usage 18

4.4 Advanced Properties 25

4.5 Usage Examples 26

5 TableImport 28

5.1 Description 28

5.2 Features 28

5.3 Usage 33

5.4 Usage Examples 42

6 TableDelete 43

6.1 Description 43

6.2 Features 43

6.3 Usage 44

6.4 Advanced Properties 47

6.5 Usage Examples 47

7 Appendix – Date and Time Format Syntax 49

8 Appendix – Double Format Syntax 53

9 Appendix – SQL Where Syntax 57

9.1 Where Clause 57

9.2 Order Clause 57

9.3 Limit Clause 59

10 Appendix – Usage for Performance Testing 60

10.1 How to check the performance of a BW Query-like activity? 60

10.2 How to check the performance of one single GET? 60

10.3 How to run multiple GETs? 60

10.4 How to simulate GETs and PUTs? 61

**Table of Figures**

Figure 1: TableImport.sh: udate logic with input file 30

**Table of Tables**

Table 1: Known Issues 7

Table 2: <config-file-options> 11

Table 3: <datagrid-options> 12

Table 4: SSL Properties 14

Table 5: Common advanced properties 14

Table 6: TableExport <options> 19

Table 7: TableExport <data options> 20

Table 8: TableExport <output formatting options> 22

Table 9: TableExport <output file options> 24

Table 10: TableExport advanced properties 26

Table 11: TableImport <options> 34

Table 12: TableImport <data options> 34

Table 13: TableImport <input format options> 35

Table 14: TableImport <source options> 36

Table 15: TableImport <querying options> 37

Table 16: TableImport <update options> 37

Table 17: TableImport <report options> 39

Table 18: TableImport <advanced options> 40

Table 19: TableImport <performance testing options> 41

Table 6: TableExport <options> 44

Table 7: TableDelete <data options> 44

Table 18: TableDelete <advanced options> 46

Table 10: TableDelete advanced properties 47

1. Introduction

This document is a user’s guide for the TIBCO PSG ActiveSpaces Tools, a set of command line tools to export data from one TIBCO ActiveSpaces datagrid table into csv-like files and import data from csv-like files. The tools are not a TIBCO product, but a project-ware provided with samples, source code and documentation.

The tools are written in Java, using the TIBCO ActiveSpaces Java API version 4.4. They can work on both Windows and Linux.

***Notes:***

* Some features of the tools come from customer requirements and may be removed in future releases if they are not generic enough.
* Some command line options or configuration properties may be removed in future releases for the same reasons.
* In case of question, please contact Pierre Ayel (payel@tibco.com).
  1. Overview
* **TableExport** can retrieve rows from one existing table in one TIBCO ActiveSpaces datagrid and write the column values into a set of CSV-like output files.
* **TableImport** can read rows from one CSV-like input file and put them into one existing table in one TIBCO ActiveSpaces datagrid. Alternatively, it can insert rows with random data as well.
* **TableDelete** can delete one or more rows from one existing table. The rows can be selected with an SQL expression (as with TableExport tool).
  1. Known Issues

Table 1: Known Issues

| **#** | **Version** | **Issue** |
| --- | --- | --- |
| 1 | All to 0.4.3 | MEDIUM: TableImport.sh: when importing rows, the name of the key column must be in lowercase in the input file. It should be case-insensitive.  **Fixed in version 0.4.4** |
| 2 | All to 0.4.3 | MEDIUM: TableImport.sh: when importing rows, the first column in the input file must be the key column. If the key column is in second place or more, the GET fails.  **Fixed in version 0.4.4** |
| 3 | All to 0.4.3 | MAJOR: To avoid a full table scan, the default datetime filtering in TableExport.sh must be changed into <column> BETWEEN <datetime> AND <datetime>  **Fixed in version 0.4.4** |
| 4 | All to 0.4.3 | MINOR: If the configuration file contains SSL parameters, they will be used even if the connection to the realm does not use SSL, which will make the connection fail.  **Workaround:**  Comment out SSL parameters in the configuration file if the connection to the real server does not use SSL.  **Fixed in version 0.4.4** |
| 5 | All to 0.4.3 | MINOR: TableExport.sh does not surround output column values with double-quotes if they contain the same characters as the output delimiter. This is not an issue for 3UK use-case as exported columns will not contain the output delimiter. |
| 6 | All to 0.4.3 | MEDIUM: TableImport.sh does not support double-quote around column values in the input CSV file. |
| 7 | All to 0.4.3 | MAJOR: TableImport.sh will fail to update a column if the input file contain an empty string as value and the column type is not string. For string columns, the column will be updated with an empty string. The next version will have an option to disable updating columns if the value is empty string in the input file.  **Fixed in version 0.4.4** |
| 8 | All to 0.4.6 | MEDIUM: TableExport.sh and TableImport.sh cannot write out or update columns of type OPAQUE.  **Fixed in version 0.5.0** |
| 9 | All to 0.5.0 | MINOR: TableImport.sh: when querying for existing rows, key columns of type datetime are ignored. |
| 10 | All to 0.5.2 | MAJOR: TableImport.sh will fail to put rows into grid with “ColumnName is null” error if –insert-new-primary-keys command line option (or corresponding property) is false or not specified (which defaults to false),  Fixed in version 0.5.3. |
| 11 | All to 0.5.3 | MEDIUM: TableImport.sh and TableExport.sh are not handling quoted command line parameters correctly. The work-around is to put those parameters as properties in the config file. For example: where=transactionstatus=’SUCCESS’ order by transactiondate ASC  **Fixed in version 0.5.4** |
| 12 | All | MEDIUM: Some features of the tools come from customer requirements and may be removed in future releases if they are not generic enough. |
| 13 | All | MEDIUM: Some of the command line options and corresponding properties default values are not accurate in this document (--columns default value is not \*) |

1. Installation
   1. Software Location

The tools are provided in a zip file called astools-<version>.zip.

* 1. Installation on Windows
     1. Pre-Requisites
* A Java runtime environment (JRE) or JDK (version 1.8) must be available.
* You DO NOT need to install TIBCO ActiveSpaces or TIBCO FTL client libraries. The tools installation archive contains its own set of TIBCO ActiveSpaces and TIBCO FTL client libraries.
  + 1. Installation
* Extract the archive into a folder of your choice.
  1. Installation on Unix
     1. Pre-Requisites
* A Java runtime environment (JRE) or JDK (version 1.8) must be available.
* You DO NOT need to install TIBCO ActiveSpaces or TIBCO FTL client libraries. The tools installation archive contains its own set of TIBCO ActiveSpaces and TIBCO FTL client libraries.
  + 1. Installation
* Extract the archive into a folder of your choice (named <install-folder>below and in the rest of this document).

***Example:***

cd /opt/tibco

mkdir tools

cd tools

unzip astools-0.7.1.zip

* Execute the following commands to complete the installation (that will unzip some of the provided client libraries for TIBCO ActiveSpaces and FTL for Linux):

cd <install-folder>/bin

sh ./install.sh

1. Common Options

The tools are a set of command line/shell scripts for Windows and Linux

This chapter describes the command line options which are common to all scripts.

* 1. <config-file-options>

All the command line options of all scripts are overwriting the values or properties that can be defined in a configuration file. The following command line parameters can be provided to specify where the configuration file is:

[--config <config-file>]

Options can be provided in any order. Options inside [ ] are optional.

Options are listed in this table:

Table 2: <config-file-options>

| Parameter | Description | Examples |
| --- | --- | --- |
| --config <config-file> | The location and name of a configuration file to use for some or all of the property to use. | --config /opt/tibco/astools/config/TableExport.cfg |

* 1. <datagrid-options>

The following command line parameters must be provided for the script to connect to one TIBCO ActiveSpaces datagrid:

[--realm <url>]

[--grid <name>]

[--user <username>]

[--password <password>]

[--dg-log-level tibdg:off|tibdg:severe|tibdg:warn|tibdg:info|tibdg:verbose|tibdg:debug|tibdgapi:debug3]

Options can be provided in any order. Options inside [ ] are optional.

Options are listed in this table:

Table 3: <datagrid-options>

| Parameter | Description | Examples |
| --- | --- | --- |
| --realm <url> | The URL of the datagrid FTL Realm Server (or a pipe-separated list of server URLs).  If missing from the command line, the tool will use the property “dg.url” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value http://localhost:8080 is used. | --realm http://10.0.0.2:18080|http://10.0.03:18080 |
| --grid <name> | The name of the datagrid to connect to.  If missing from the command line, the tool will use the property “dg.gridname” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value \_default is used. | --grid \_default |
| --user <username> | The username to use to connect to the datagrid.  If missing from the command line, the tool will use the property “dg.username” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, no credentials are used. | --user tibco |
| --password <password> | The password to use to connect to the datagrid.  If missing from the command line, the tool will use the property “dg.password” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, no password is used. | --password xyz |
| --dg-log-level tibdg:off|  tibdg:severe|  tibdg:warn|  tibdg:info|  tibdg:verbose|  tibdg:debug|  tibdgapi:debug3 | The level of tracing for internal ActiveSpaces API logs. Those logs are traced inside the tools own log files.  If missing from the command line, the tool will use the property “dg.log.level” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value tibdg:off is used. | --dg-log-level tibdg:debug |

* 1. SSL Properties

The following properties MUST be specified in the configuration file if the connection to the datagrid uses SSL:

Table 4: SSL Properties

| Property | Description | Examples |
| --- | --- | --- |
| com.tibco.tibdg.trust.type | If using SSL, this indicate if the tool should trust any certificate (trust\_everyone) or those from the file configured in com.tibco.tibdg.trust.file property (trust\_file) | trust\_file |
| com.tibco.tibdg.trust.file | If the property com.tibco.tibdg.trust.type equals trust\_file, this is the file containing the certificate to trust. | /tmp/certs.jks |

* 1. Advanced Properties

The following properties can be specified in the configuration file only, for advanced purposes:

Table 5: Common advanced properties

| Property | Description | Examples |
| --- | --- | --- |
| dg.timeout | The timeout in seconds for communications with the datagrid.  Default value is 5 seconds. | 10 |
| dg.connect.wait.time | The timeout in seconds to connect to the datagrid proxies (can be written as a double value).  It is recommended not to be greater than 10.  Default value is 0.1 seconds. | 1 |
| dg.realm.connect.retries | The number of connection attempts to the datagrid FTL realm server.  Default value is 5. | 10 |
| dg.client.label | The name associated with the connections the tool establish to the datagrid. This can be used for investigation purposes to help identify which client or application is connected to the grid.  If this property is missing, the tools create their own name. | MyClient |

1. TableExport

The tools are a set of command line scripts for Windows and Linux. Each script usage is documented in its own chapter.

* 1. Description

**TableExport** can retrieve rows from one existing table in one TIBCO ActiveSpaces datagrid and write the column values into a set of CSV-like output files, or on standard output or not at all (the tool will report how long it took to query the rows and how many were found).

To execute the tool, run the bin/TableExport.bat script on Windows or bin/TableExport.sh on Linux.

* 1. Features
     1. Datetime filtering

The tool can select rows which “transactiondate” column value is matching the current day by generating a WHERE clause equivalent to:

transactiondate BETWEEN ‘<yesterday> 00:00:00.000000000Z’ AND ‘<yesterday> 23:59:59.999999999Z’

If you want the filtering to happen:

* Add the --datefime-filter true option to the command line.
* Or set the property statement.datetime.filter=true

To specify another column to use for the filtering:

* Add the --datefime-column <column name> option to the command line.
* Or set the property statement.datetime.filter.column=<column name>

If you specify a where clause (with the –where command line option or where property) and the filtering is on, the where clause will be added the datetime filtering, e.g. the full WHERE clause or the rows to retrieve will be:

transactiondate BETWEEN ‘<yesterday> 00:00:00.000000000Z’ AND ‘<yesterday> 23:59:59.999999999Z’ AND <your clause>

***Notes:***

* For the datetime filtering to work, the selected columns (with –columns command line option or columns property) must equal to “\*” or contain the transactiondate column name (or the column name specified with the –datetime-column or statement.datetime.filter.column property).
  + 1. Output

To write the found rows on standard output:

* Add the --output stdout option to the command line.
* Or set the property output.file=stdout

To NOT write the found rows anywhere:

* Add the --output none option to the command line.
* Or set the property output.file=none

To write the rows into several files:

* If you want to specify the full location and name of the output file:
  + Add the --output <name> option to the command line.
  + Or set the property output.file=<name>

The tool will generate the following files:

* + <name>\_1\_<total number of files>.<extension>
  + <name>\_2-<total number of files>.<extension>
  + Etc…
* Otherwise, you must:
  + Add the –output-file-folder <folder> option to the command line.
  + Or set the property output.file.folder=<folder>

The tool will generate the following files:

* + <folder>/<prefix>\_<yyyyMMdd>\_v<version>\_1\_<total number of files>.<extension>
  + <folder>/<prefix>\_<yyyyMMdd>\_v<version>\_2-<total number of files>.<extension>
  + Etc…

<extension> is csv by default. To change it:

* Add the –-file-ext <extension> option to the command line.
* Or set the property output.file.extension=<extension>

<prefix> is transactions\_GB\_32f402 by default. To change it:

* Add the –-output-file-prefix <prefix> option to the command line.
* Or set the property output.file.prefix=<prefix>

<yyyyMMdd> is yesterday’s date in yyyyMMdd format (the same date as in the datetime filter, even if the filter is disabled).

<version> is 1 by default. The change it:

* Add the --output-file-version <version> option to the command line.
* Or set the property output.file.version=<version>

The number of files generated is bases on the maximum number of rows to write per file, which is 25000 by default.

To change it:

* Add the –max-rows <number> option to the command line.
* Or set the property output.file.max.rows=<number>
  + 1. Columns

By default, the tool extracts all the columns of the table.

To change the list of columns:

* Add the --columns <separated list of column names> option to the command line.
* Or set the property columns=<separated list of column names>

For example:

* + 1. Output Header Line

If the rows are written into files, the following line is written as header line in every file:

<yyyy-MM-dd> 00:00:00 – 23:59:59 UTC

<yyyy-MM-dd> is yesterday’sdate (the same date used in the datetime filter, even if the filter is disabled).

To not write any header line:

Add the --header false option to the command line.

Or set the property output.header=false

To write anything else:

Add the –output-header-value <anything else> option to the command line.

Or set the property output.header.value=<anything else>

***Notes:***

* If rows is written on standard output, the header is written with the name of the extracted columns.
* If the datetime filter is disabled, the header is written in the files with name of the extracted columns
  1. Usage

From any working directory, type the following:

<install-folder>/bin/TableExport.sh

<config-file-options>

<datagrid-options>

[--table <table>]

[--columns <c1,c2,c3…>]

[--where <where clause>]

..[--datetime-from “yyyy-MM-dd HH:mms:ss”]

..[--datetime-to “yyyy-MM-dd HH:mms:ss”]

[--datetime-filter-column <name>]

[--datetime-filter true|false]

[--datetime-day yesterday|today]

[--table-rowset true|false]

[--checkpoint <name>]

[--header true|false]

[--output-header-value <header>]

[--delimiter <delimiter>]

[--datetime-format <format>]

[--datetime-timezone <zone>]

..[--double-format <format>]

..[ --double-rounding ceiling|down|floor|half\_down|half\_even|half\_up|unnecessary|up]

..[--encoding <encoding>]

[--output stdout|none|<full pathname/ file>]

[--file-ext <ext>]

[--create-empty-file <true|false>]

[--max-rows <n>]

[--output-file-folder <folder>]

[--output-file-prefix <prefix>]

..[--output-file-version <version>]

Options can be provided in any order. Options inside [ ] are optional.

Options are listed in this table:

Table 6: TableExport <options>

| Parameter | Description | Examples |
| --- | --- | --- |
| <config-file-options> | The location and name of a configuration file to use for some or all of the property to use.  See section 3.1 <config-file-options> for more details. | |
| <datagrid-options> | The parameters to connect to TIBCO ActiveSpaces datagrid.  See section 3.2 <datagrid-options>for more details. | |
| Other options are described into the next paragraphs. | | |

* + 1. Data Options

Table 7: TableExport <data options>

| Parameter | Description | Examples |
| --- | --- | --- |
| --table <table> | The name of the table to export data from.  If missing from the command line, the tool will use the property “table” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default name “transactions” is used.  Table names are case insensitive. | --table transactions |
| --columns <c1,c2,c3…> | The comma-separated list of column names to export.  If missing from the command line, the tool will use the property “columns” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default name “\*” is used (all columns).  Names are case-insensitive? | -columns transactionid,MSISDN,status,Amount |
| --where <where clause> | The where clause to filter exported rows.  If missing from the command line, the tool will use the property “where” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, no clause is used. | -- where “opcoid = ‘xxx’ and description = ‘bla lab la’” |
| --datetime-from | The range of date and time to select rows in the format yyyy-MM-dd HH:mm:ss.  If missing from the command line, the tool will use the properties “statement.datetime.filter.from” and “statement.datetime.filter.to” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the value <current-day> 00:00:00 and <current-day> 23:59:59 are used. | --datetime-from 2020-05-20 10:30:00  --datetime-to 2020-05-21 01:00:30 |
| --datetime-to |
| --datetime-column <name> | The name of the column on which the date and time filter applies to.  If missing from the command line, the tool will use the property “statement.datetime.filter.column” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the value transactiondate is used. | --datetime-column transactiondate |
| --datetime-filter true|false | Indicates if rows should be filtered by date and time (with the options –datetime-from, --datetime-to and –datetime-column) or not.  If missing from the command line, the tool will use the property “statement.datetime.filter” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the value false is used. | --datetime-filter false |
| --datetime-day yesterday|today | Indicates which date to use in the daefault datetime filter.  If missing from the command line, the tool will use the property “statement.datetime.filter.day” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the value “yesterday” is used. | --datetime-day today |
| --table-rowset true|false | Indicates if the rows should be queried with the table.createRowSet API (true) or a normal SQL Query (false). The table.createRowSet is similar to a BW Palette Query activity.  If missing from the command line, the tool will use the property “statement.table.rowset” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the value false is used (SQL Query is used).  If true, the –columns option must contain the list of columns to write out and thus any SQL like transformation of column names or alias will fail. | --table-row-set true |
| --checkpoint <name> | Query data from one existing datagrid checkpoint instead of the current data.  If missing from the command line, the tool will use the property “checkpoint” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the tool queries the current data. | --checkpoint C1 |

* + 1. Output Formatting Options

Table 8: TableExport <output formatting options>

| Parameter | Description | Examples |
| --- | --- | --- |
| --header true|false | Indicates if a header line with column names should be written in the output files (or stdout).  If missing from the command line, the tool will use the property “output.header” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value true is used (header lines are printed out). | --header true |
| --output-header-value <header> | A custom value to use as header in output files (if –header is enabled).  If missing from the command line, the tool will use the property “output.header.value” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value “<yyyyMMdd> 00:00:00 – 23:59:59 UTC” is used if datetime filtering is enabled, otherwise a delimiter separated list of extracted column names.. | --output-header-value XXX |
| --delimiter <delimiter> | The delimiter to use between column values (and column names in the header lines).  If missing from the command line, the tool will use the property “output.field.delimiter” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value “,” is used. | --delimiter , |
| --datetime-format <format> | The date and time format for writing out date and time values, in Java SimpleDateFormat syntax (see appendix for more details).  If missing from the command line, the tool will use the property “output.datetime.format” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value “yyyy-MM-dd’T’HH:mm:ss” is used. | --datetime-format yyyy-MM-dd’T’HH:mm:ss |
| --datetime-timezone <zone> | The timezone to use when formatting date and time.  If missing from the command line, the tool will use the property “output.datetime.timezone” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value “UTC” is used. | --datetime-timezone BST |
| --double-format | The format for writing out double values, in Java DecimalFormat syntax (see appendix for more details).  If missing from the command line, the tool will use the property “output.double.format” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value “#.000” is used. | --double format #,###,###.000 |
| --double-rounding ceiling|  down|  floor|  half\_down|  half\_even|  half\_up|  unnecessary|  up | Rounding to apply when formatting double values. Values are the same as Java class RoundingMode enumerations.  If missing from the command line, the tool will use the property “output.double.rounding” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value “half\_down” is used. | --double-rounding floor |
| --encoding <encoding< | **Since version 0.5.0:**  The encoding to use to transform OPAQUE bytes from the grid into the output.  If missing from the command line, the tool will use the property “encoding” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value “UTF-8” is used. | --encoding UTF-16 |

* + 1. Output File Options

Table 9: TableExport <output file options>

| Parameter | Description | Examples |
| --- | --- | --- |
| --output stdout|none|<full pathname/ file> | The folder and name of the output file to write into. If only one file is written out, the name will be suffixed with \_1-1.<extension>. If multiple files are written out, their names will be suffixed with \_<index>-<filecount>.<extension>.  If missing from the command line, the tool will use the property “output.file” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the tools will used –output-file-folder, --output-file-prefix and –output-file-version to generate an output file name.  If this command line value (or property from the property file or default value) equals “none”, the tool only fetches the data from the datagrid but do not output any data. If the value is “stdout”, the tool prints out the rows into standard output. | --output /tmp/outputfile |
| --file-ext <ext> | The extension to add to output filenames.  If missing from the command line, the tool will use the property “output.file.extension” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value “csv” is used. | --file-ext csv |
| --create-empty-file <true|false> | Indicates if one empty file should be written out if no row are found.  If missing from the command line, the tool will use the property “output.create.empty.file” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the value “false” is used (no empty file). | --create-empty-file true |
| --max-rows <n> | The maximum number of rows to write in each output file.  If missing from the command line, the tool will use the property “output.file.max.rows” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the value 25000 is used. | --max-rows 25000 |
| --output-file-folder <folder> | Indicates the folder where output files are written in case the –output command line option or property is missing. In this case, the output files will be <folder>/<prefix>\_<yyyyMMdd>\_v<version>\_<index>-<filecount>.<extension>.  <prefix> is configured with the –output-file-prefix option.  <yyyyMMdd> is the current day.  <version> is configured with the –output-file-version option.  If missing from the command line, the tool will use the property “output.file.folder” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the tool will use –output value as file fullname. | --output-file-folder /shared/results/boku/xxxx |
| --output-file-prefix <prefix> | Indicates the <previx> value to use when –output-file-folder is used.  If missing from the command line, the tool will use the property “output.file.prefix” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value “transactions\_GB\_32f402” is used. | --output-file-prefix bar |
| --output-file-version <version> | Indicates the <version> value to use when –ouput-file-folder is used.  If missing from the command line, the tool will use the property “output.file.version” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value “1” is used. | --output-file-version 2 |

* 1. Advanced Properties

The following properties can be specified in the configuration file only for advanced purposes:

Table 10: TableExport advanced properties

| Property | Description | Examples |
| --- | --- | --- |
| statement.prefetch | Number of rows to fetch from the datagrid at a time. If not specified, the tool will use 256.  The bigger the number, the more memory the tool will require. This will improve performance but degrade performance of the TIBCO ActiveSpaces components and network interfaces.  The recommended value is 10000 (TBC). | 10000 |
| statement.fetch.timeout | Number of TBC to wait for each row fetching.  0 means no timeout.  The default value is too small in case the tool cannot find matching rows early enough.  The recommended value is 0 (TBC). | 0 |

* 1. Usage Examples

Write a config file, named /tmp/config.cfg, with the following content:

dg.url=http://10.0.0.2:18080

dg.datagrid=hadpl

To extract columns transactionid,opcoid,msisdn from the table transactions, you can execute this command:

<install-folder>/bin/TableExport.sh --config /tmp/config.cfg –columns transactionid,opcoid,msisdn --output /tmp/output --max-rows 10000

If 24000 rows have been found, the following output files will be written out:

* /tmp/output\_1-3.csv (for rows 1 to 10000)
* /tmp/output\_2-3.csv (for rows 10001 to 20000)
* /tmp/output\_3-3.csv (for rows 20001 to 24000)

The following commands will produce the same effects:

<install-folder>/bin/TableExport.sh --config /tmp/config.cfg –columns transactionid,opcoid,msisdn --output /tmp/output --max-rows 10000 –-realm http://10.0.0.2:18080 –grid hadpl

<install-folder>/bin/TableExport.sh -–columns transactionid,opcoid,msisdn --output /tmp/output --max-rows 10000 –-realm http://10.0.0.2:18080 –grid hadpl

<install-folder>/bin/TableExport.sh --config /tmp/config.cfg –columns transactionid,opcoid,msisdn --output /tmp/output --max-rows 10000 –-table transactions –file-ext csv –-header true

To extract the rows but write them only on standard output, you can execute:

<install-folder>/bin/TableExport.sh --config /tmp/config.cfg –columns transactionid,opcoid,msisdn --output stdout --max-rows 10000

To extract the rows between two dates:

<install-folder>/bin/TableExport.sh --config /tmp/config.cfg –columns transactionid,opcoid,msisdn --output stdout --max-rows 10000 –datetime-from “2020-05-20 00:00:00” –datetime-to “2020-05-20 23:59:59”

To extract all the rows (and not filter by date and time):

<install-folder>/bin/TableExport.sh --config /tmp/config.cfg –columns transactionid,opcoid,msisdn --output stdout --max-rows 10000 –datetime-filter false

1. TableImport

The tools are a set of command line scripts for Windows and Linux. Each script usage is documented in its own chapter.

* 1. Description

**TableImport** can read rows from one CSV-like input file and put them into one existing table in one TIBCO ActiveSpaces datagrid. Alternatively, it can insert rows with random data as well.

To execute the tool, run the bin/TableExport.bat script on Windows or bin/TableExport.sh on Linux.

Since version 0.8.0, CSV Files with double-quotes are now fully supported.

* 1. Features

Random Data

The tool can insert random data in ANY table for testing purposes. To do so:

* Comment out the input.file property in the configuration file.
* Do not pass –-input option in the command line.

***Examples:***

To create 25 rows of random data into transactions table:

./TableImport.sh –realm <realm> –-rows 25

To create 25 rows of random data into subscriber table:

./TableImport.sh –realm <realm> --table subscriber –rows 25

To create 25 rows of random data into subscription table:

./TableImport.sh –realm <realm> --table subscription –rows 25

***Notes:***

* All rows are upserted, they overwrite any value already present in the table.
* For the transactions table:
  + row names are hardcoded in the tool.
  + The first third of the upserted rows will have transactiondate column set as current time and yesterday date.
  + The second third of the upserted rows will have transactiondate column set as current time and current day.
  + The last third of the upserted rows will have transactiondate column as current time and tomorrow date.

Input File

To use a CSV-like file as input for the rows to create/update:

* Add the –-input <file> option to the command line.
* Or set the property input.file=<file>

The first line of the file MUST contains the line of columns that each row will contain. If not, use the –header false command line option (or corresponding property) and –column c1,c2,c3… command line option to specify the name of the columns in the input file (this 2 option usage has been fixed in version 0.5.2).

Querying Existing rows

When the tool is using an input file, If will search for existing rows in the table before updating them.

For each row line, the tool will first collect all the key columns and use those value to search for existing rows. If only one key column is configured, the tool use a GET to retrieve a row. If mulitple key columns are configured, the tool invokes the API table.createRowSet to retrieve the row.

If multiple existing rows are found for one input file line, the tool will skip the line and not update any row in the table.

If one row is found, the tool will update it, according to the update logic described in the next section.

If no row is found, the tool will not create any row.

To create a new row if no row is found:

* Add the --insert-new-rows true option to the command line.
* Or set the property insert.new.rows=true

By default the key columns are transactionid.

To change those:

* Add the –key-columns c1,c2,… option to the command line.
* Or set the property input.key.columns=c1,c2,…

***Notes:***

* When configuration multiple columns as key columns:
  + Those columns do NOT need to be the primary key of the table, however, they need to be indexed columns otherwise the search will be slow and may impact the performance of the datagrid.
  + Columns of type datetime are ignored and not used for the query.

Update Existing Rows

The logic to update existing rows is described in the following diagram:

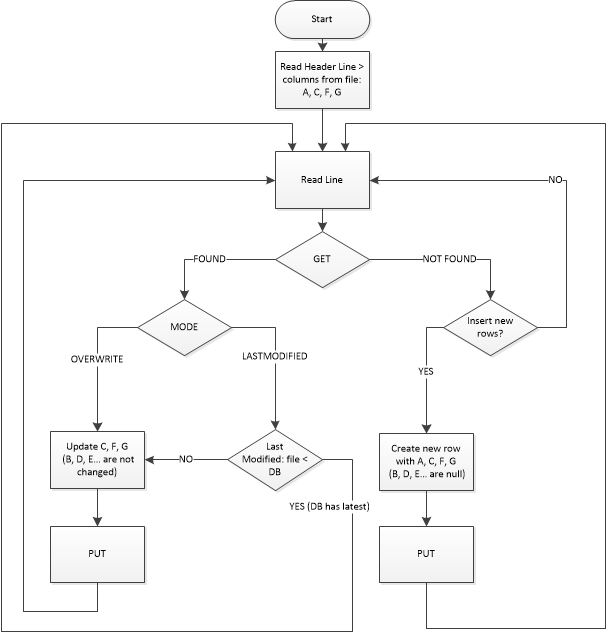


Figure 1: TableImport.sh: udate logic with input file

The column used to compare rows from the input file and from the table is by default transactiondate.

To change it:

* Add the –lastmodified-column <column> option to the command line.
* Or set the property input.lastmodified.column=<column>

To switch between “OVERWRITE” and “LASTMODIFIED” mode (default is “OVERWRITE”):

* Add the –update-mode overwrite|lastmodified option to the command line.
* Or set the property update.mode=<mode>

Reporting

When using an input file, the tool will report how many lines have been processed, skipped, in error, etc…

To ask the tool to write the lines and the report in output files:

* Add the –-report-folder <folder> option to the command line.
* Or set the property report.folder=<folder>

The report folder will be automatically create in case it does not exist. **Since version 0.5.2, yyyyMMdd-HHmmss will be added to the folder name automatically so each execution uses a separate report folder.**

***Examples:***

In the log file and standard output, the report will look like this:

…2705 Summary (number of lines):

…2705 lines.total=10

…2705 lines.errors=0

…2705 existing.updated.success=10

…2705 existing.updated.errors=0

…2705 notfound.inserted.success=0

…2705 notfound.inserted.errors=0

…2705 existing.skipped=0

…2705 notfound.skipped=0

…2705 existing.multiple=0

If a report folder is configured the following files are written out:

* <folder>/existing-skipped.csv: a file containing the header line and skipped lines from the input file (skipped because the row already exists and the date from the input file is lower than the date from the table)
* <folder>/existing-updated-errors.csv: a file containing the header line and updated lines from the input file which update failed.
* <folder>/existing-updated-success.csv: a file containing the header line and updated lines from the input file which update succeeded.
* <folder>/line-error.csv: a file containing the header line and lines from the input file which row querying or date comparison failed (for example if a datetime value format is incorrect)
* <folder>/notfound-inserted-errors.csv: a file containing the header line and new created lines from the input file, which insertion failed.
* <folder>/notfound-inserted-success.csv: a file containing the header line and new created lines from the input file, which insertion succeeded.
* <folder>/notfound-skipped.csv:: a file containing the header line and not found rows and not inserted rows.
* <folder>/existing-multiple.csv: a file containing the header line and line which correspond to multiple rows in the table (and were then skipped).
* <folder>/summary.txt: The report as trace into the log file.
  1. Usage

From any working directory, type the following:

<install-folder>/bin/TableImport.sh

<config-file-options>

<datagrid-options>

[--table <table>]

[--columns <c1,c2,c3…>]

[--header true|false]

[--delimiter <delimiter>]

[--datetime-format <format>]

[--datetime-timezone <zone>]

[--encoding <encoding>]

[--key-columns <c1,c2,c3…>]

..[--lastmodified-column <column>]

..[--update-mode overwrite|lastmodified]

..[--insert-new-rows true|false]

[--insert-new-primary-keys true|false]

[--upsert-empty-columns true|false]

[--intput <full pathname/ file>]

[--rows <rows>]

..[--report-folder <folder>]

..[--batch-row-count <n>]

[--threads <n>]

[--loop <seconds>]

[--line-delay-ms <ms>]

[--put-delay-ms <ms>]

Options can be provided in any order. Options inside [ ] are optional.

Options are listed in this table:

Table 11: TableImport <options>

| Parameter | Description | Examples |
| --- | --- | --- |
| <config-file-options> | The location and name of a configuration file to use for some or all of the property to use.  See section 3.1 <config-file-options> for more details. | |
| <datagrid-options> | The parameters to connect to TIBCO ActiveSpaces datagrid.  See section 3.2 <datagrid-options>for more details. | |
| Other options are described into the next paragraphs. | | |

* + 1. Data Options

Table 12: TableImport <data options>

| Parameter | Description | Examples |
| --- | --- | --- |
| --table <table> | The name of the table to put data into.  If missing from the command line, the tool will use the property “table” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default name “transactions” is used. | --table transactions |
| --columns <c1,c2,c3…> | The comma-separated list of column names to import if the input file does not start with a header line (and of course the option –header is set to false).  If missing from the command line, the tool will use the property “columns” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default name “\*” is used.  Column names are case-insensitive. | -columns transactionid,msisdn,status |

* + 1. Input Format Options

Table 13: TableImport <input format options>

| Parameter | Description | Examples |
| --- | --- | --- |
| --header true|false | Indicates if the input file first line should be parsed as a header line containing the list of column names to put into the table. If not, the column names from –column option will be used as columns in the input file.  If missing from the command line, the tool will use the property “input.header” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value true is used (the first line of the input file is treated as header line). | --header true |
| --delimiter <delimiter> | The delimiter to use between column values (and column names in the header lines).  If missing from the command line, the tool will use the property “input.field.delimiter” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value “,” is used. | --delimiter , |
| --datetime-format <format> | The date and time format for parsing date and time values from the input file, in Java SimpleDateFormat syntax (see appendix for more details).  If missing from the command line, the tool will use the property “input.datetime.format” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value “yyyy-MM-dd’T’HH:mm:ss” is used. | --datetime-format yyyy-MM-dd’T’HH:mm:ss |
| --datetime-timezone <zone> | **Since version 0.5.0:**  The timezone to use when parsing date and time values from the input file.  If missing from the command line, the tool will use the property “input.datetime.timezone” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value “UTC” is used. | --datetime-timezone BST |
| --encoding <encoding< | **Since version 0.5.0:**  The encoding to use to transform strings from the input file into OPAQUE bytes for the grid.  If missing from the command line, the tool will use the property “encoding” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value “UTF-8” is used. | --encoding UTF-16 |

* + 1. Source Options

Table 14: TableImport <source options>

| Parameter | Description | Examples |
| --- | --- | --- |
| --input <full pathname/ file> | The folder and name of the input file to read from.  If missing from the command line, the tool will use the property “input.file” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, random data is put into the table. | --input /tmp/data.csv |
| ---rows <n> | The maximum number of rows with random values to put into the table in case no input file has been specified.  If missing from the command line, the tool will use the property “random.row.count” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the value 1 is used. | ---rows 25000 |

* + 1. Querying Options

Table 15: TableImport <querying options>

| Parameter | Description | Examples |
| --- | --- | --- |
| --key-columns <c1,c2,c3…> | A comma-separated list of one or more columns to use for querying existing row before updating them.  If equal to “none”, no querying is done and input file line will be upserted.  If missing from the command line, the tool will use the property “input.key.columns” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the column transactionid is used. | --key-columns recorded,subscriberid |

* + 1. Update Options

Table 16: TableImport <update options>

| Parameter | Description | Examples |
| --- | --- | --- |
| --lastmodified-column <column> | The name of the datetime column to use for comparing rows from the input file and rows from the table.  If missing from the command line, the tool will use the property “input.lastmodified.column” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the column transactiondate is used. | --lastmodified-column createdate |
| --update-mode overwrite|lastmodified | Switch between “OVERWRITE” and “LASTMODIFIED” mode.  If missing from the command line, the tool will use the property “update.mode” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the mode “OVERWRITE” is used. | --update-mode lastmodified |
| --insert-new-rows true|false | If a row is not found, the tool will upsert it into the table  If missing from the command line, the tool will use the property “insert.new.rows from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, no new row is created. | --insert-new-rows true |
| --insert-new-primary-keys trua|false | Since version 0.5.2:  If a row is inserted and has no primary key value (or the value is empty string) and this option is true, a UUID will be put into the primary key column before a row is put into the grid.  If missing from the command line, the tool will use the property “insert.new.primary.keys from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value is false, which means no primary key is generated. | --insert-new-primary-keys true |
| --upsert-empty-columns true|false | If true:  If a column from the input file is empty (empty string), then the column is set to empty string or null (for datetime, opaque and string type columns) in the table.  If false  If a column from the input file is empty (empty string), then the column from the table is not changed for existing rows and not set for newly created rows (left to null).  Double and Long type columns cannot be changed to null.  If missing from the command line, the tool will use the property “upsert.empty.columns” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, empty columns are not upserted (false). | --upsert-empty-columns true |

* + 1. Report Options

Table 17: TableImport <report options>

| Parameter | Description | Examples |
| --- | --- | --- |
| --report-folder <folder> | The folder in which report file will be written.  If missing from the command line, the tool will use the property “report-folder” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, report files are not written out. | --report-folder c:/temp/report-20200619 |

* + 1. Advanced Options

Table 18: TableImport <advanced options>

| Parameter | Description | Examples |
| --- | --- | --- |
| --batch-row-count <n> | Number of rows to prepare in memory before sending them as a batch to put into the datagrid.  If missing from the command line, the tool will use the property “batch.row.count” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value is 10000.  The bigger the number, the more memory the tool will require. This will improve performance but also degrade performance of the TIBCO ActiveSpaces components and network interfaces.  The recommended value is 100 (TBC). | --batch-row-count 100 |
| --retry-count <n> | **Since version 0.5.5:**  Number of retries to execute in case a GET or PUT fails with a timeout.  If missing from the command line, the tool will use the property “retry.count” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value is 3. | --retry-count 10 |
| --retry-delay-ms <ms> | **Since version 0.5.5:**  Number of milliseconds to wait between GET or PUT retry attempts.  If missing from the command line, the tool will use the property “retry.count” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value is 1000 (1 second). | --retry-delay-ms 10000 |

* + 1. Performance Testing Options

Table 19: TableImport <performance testing options>

| Parameter | Description | Examples |
| --- | --- | --- |
| --threads <n> | Executes the tool on multiple threads. Each threads execute exactly the same whole import logic. Each thread has its own connection to the grid.  If missing from the command line, the tool will use the property “threads” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, 1 thread is used. | --threads 2 |
| --loop <seconds> | If <seconds> is greater than zero, each thread will execute the same import logic in a loop until <seconds> has elapsed (the last execution will complete before the tool exits so the total execution time may be greater than <seconds>).  If missing from the command line, the tool will use the property “loops” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value 0 is used, which means each thread executes the logic only once. | --loop 60000 |
| --line-delay-ms <ms> | Pauses for <ms> milliseconds before processing one line from the input file.  If missing from the command line, the tool will use the property “line.delay.ms” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value 0 is used (no delay). | --line-delay-ms 1000 |
| --put-delay-ms <ms> | Pauses for <ms> milliseconds before one row or the bacth of row is put into the grid.  If missing from the command line, the tool will use the property “put.delay.ms” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value 0 is used (no delay). | --put-delay-ms 1000 |

* 1. Usage Examples

To create 25 rows of random data into transactions table:

<install-folter>/bin/TableImport.sh –-realm <realm> –-rows 25

To create 25 rows of random data into subscriber table:

<install-folter>/bin/TableImport.sh –-realm <realm> --table subscriber –rows 25

To create 25 rows of random data into subscription table:

<install-folter>/bin/TableImport.sh –-realm <realm> --table subscription –rows 25

To import data from an input file into the subscription table and ignore lines with createdate column where file data is lower than the date in the table:

<install-folter>/bin/TableImport.sh –-realm <realm> --table subscription –-input <input.csv file> --key-columns recordid –-lastmodified-columns createddate –-update-mode lastmodified –report-folder /tmp/report-20200619-1324

To import data from an input file into the subscription table and overwrite the table data with the file data, ignoring dae and time:

<install-folter>/bin/TableImport.sh –-realm <realm> --table subscription –-input <input.csv file> --update-mode overwrite --report-folder /tmp/report-20200619-1324

1. TableDelete

The tools are a set of command line scripts for Windows and Linux. Each script usage is documented in its own chapter.

* 1. Description

**TableDelete** can delete one or more rows from one existing table in one TIBCO ActiveSpaces datagrid.

To execute the tool, run the bin/TableDelete.bat script on Windows or bin/TableDelete.sh on Linux.

* 1. Features
     1. Datetime filtering

The tool can select rows which “transactiondate” column value is matching the current day by generating a WHERE clause equivalent to:

transactiondate BETWEEN ‘<yesterday> 00:00:00.000000000Z’ AND ‘<yesterday> 23:59:59.999999999Z’

If you want the filtering to happen:

* Add the --datefime-filter true option to the command line.
* Or set the property statement.datetime.filter=true

To specify another column to use for the filtering:

* Add the --datefime-column <column name> option to the command line.
* Or set the property statement.datetime.filter.column=<column name>

If you specify a where clause (with the –where command line option or where property) and the filtering is on, the where clause will be added the datetime filtering, e.g. the full WHERE clause or the rows to retrieve will be:

transactiondate BETWEEN ‘<yesterday> 00:00:00.000000000Z’ AND ‘<yesterday> 23:59:59.999999999Z’ AND <your clause>

***Notes:***

* For the datetime filtering to work, the selected columns (with –columns command line option or columns property) must equal to “\*” or contain the transactiondate column name (or the column name specified with the –datetime-column or statement.datetime.filter.column property).
  1. Usage

From any working directory, type the following:

<install-folder>/bin/TableDelete.sh

<config-file-options>

<datagrid-options>

[--table <table>]

[--where <where clause>]

..[--datetime-from “yyyy-MM-dd HH:mms:ss”]

..[--datetime-to “yyyy-MM-dd HH:mms:ss”]

[--datetime-filter-column <name>]

[--datetime-filter true|false]

[--datetime-day yesterday|today]

[--batch-row\_count <n>]

[--retry-count <n>]

[--retry-delay-ms <ms>]

Options can be provided in any order. Options inside [ ] are optional.

Options are listed in this table:

Table 6: TableExport <options>

| Parameter | Description | Examples |
| --- | --- | --- |
| <config-file-options> | The location and name of a configuration file to use for some or all of the property to use.  See section 3.1 <config-file-options> for more details. | |
| <datagrid-options> | The parameters to connect to TIBCO ActiveSpaces datagrid.  See section 3.2 <datagrid-options>for more details. | |
| Other options are described into the next paragraphs. | | |

* + 1. Data Options

Table 7: TableDelete <data options>

| Parameter | Description | Examples |
| --- | --- | --- |
| --table <table> | The name of the table to delete data from.  If missing from the command line, the tool will use the property “table” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default name “transactions” is used.  Table names are case insensitive. | --table transactions |
| --where <where clause> | The where clause to filter deleted rows.  If missing from the command line, the tool will use the property “where” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, no clause is used (all rows from the table are deleted). | -- where “opcoid = ‘xxx’ and description = ‘bla bla bla” |
| --datetime-from | The range of date and time to select rows in the format yyyy-MM-dd HH:mm:ss.  If missing from the command line, the tool will use the properties “statement.datetime.filter.from” and “statement.datetime.filter.to” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the value <current-day> 00:00:00 and <current-day> 23:59:59 are used. | --datetime-from 2020-05-20 10:30:00  --datetime-to 2020-05-21 01:00:30 |
| --datetime-to |
| --datetime-column <name> | The name of the column on which the date and time filter applies to.  If missing from the command line, the tool will use the property “statement.datetime.filter.column” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the value transactiondate is used. | --datetime-column transactiondate |
| --datetime-filter true|false | Indicates if rows should be filtered by date and time (with the options –datetime-from, --datetime-to and –datetime-column) or not.  If missing from the command line, the tool will use the property “statement.datetime.filter” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the value false is used. | --datetime-filter false |
| --datetime-day yesterday|today | Indicates which date to use in the daefault datetime filter.  If missing from the command line, the tool will use the property “statement.datetime.filter.day” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the value “yesterday” is used. | --datetime-day today |

* + 1. Advanced Options

Table 18: TableDelete <advanced options>

| Parameter | Description | Examples |
| --- | --- | --- |
| --batch-row-count <n> | Number of rows to prepare in memory before sending them as a batch to delete from the datagrid.  If missing from the command line, the tool will use the property “batch.row.count” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value is 10000.  The bigger the number, the more memory the tool will require. This will improve performance but also degrade performance of the TIBCO ActiveSpaces components and network interfaces.  The recommended value is 100 (TBC). | --batch-row-count 100 |
| --retry-count <n> | Number of retries to execute in case a deletion fails with a timeout.  If missing from the command line, the tool will use the property “retry.count” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value is 3. | --retry-count 10 |
| --retry-delay-ms <ms> | Number of milliseconds to wait between deletions retry attempts.  If missing from the command line, the tool will use the property “retry.count” from the configuration file.  If the property is also missing from the configuration file or no configuration file has been specified, the default value is 1000 (1 second). | --retry-delay-ms 10000 |

* 1. Advanced Properties

The following properties can be specified in the configuration file only for advanced purposes:

Table 10: TableDelete advanced properties

| Property | Description | Examples |
| --- | --- | --- |
| statement.prefetch | Number of rows to fetch from the datagrid at a time. If not specified, the tool will use 256.  The bigger the number, the more memory the tool will require. This will improve performance but degrade performance of the TIBCO ActiveSpaces components and network interfaces.  The recommended value is 10000 (TBC). | 10000 |
| statement.fetch.timeout | Number of TBC to wait for each row fetching.  0 means no timeout.  The default value is too small in case the tool cannot find matching rows early enough.  The recommended value is 0 (TBC). | 0 |

* 1. Usage Examples

Write a config file, named /tmp/config.cfg, with the following content:

dg.url=http://10.0.0.2:18080

dg.datagrid=hadpl

To delete all rows from the table transactions, you can execute this command:

<install-folder>/bin/TableDelete.sh --config /tmp/config.cfg

The following commands will produce the same effects:

<install-folder>/bin/TableDelete.sh --config /tmp/config.cfg –-realm http://10.0.0.2:18080 –grid hadpl –-table transactions –-batch-row-count 10000

To extract the rows between two dates:

<install-folder>/bin/TableDelete.sh --config /tmp/config.cfg -–datetime-from “2020-05-20 00:00:00” –datetime-to “2020-05-20 23:59:59” –-datetime-filter true

To delete all rows that satisfy the SQL where clause “description – ‘bla bla bla’”:

<install-folder>/bin/TableDelete.sh --config /tmp/config.cfg –-where “description=’blab bla bla’”

1. Appendix – Date and Time Format Syntax

This chapter explains the syntax of the command line option (and corresponding property) –datetime-format.

This chapter is copied from the Oracle Java documentation of the java.text.SimpleDateFormat class from Java version 1.8.

It is also available online at <https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html>.

Date and time formats are specified by date and time pattern strings. Within date and time pattern strings, unquoted letters from 'A' to 'Z' and from 'a' to 'z' are interpreted as pattern letters representing the components of a date or time string. Text can be quoted using single quotes (') to avoid interpretation. "''" represents a single quote. All other characters are not interpreted; they're simply copied into the output string during formatting or matched against the input string during parsing.

The following pattern letters are defined (all other characters from 'A' to 'Z' and from 'a' to 'z' are reserved):

|  |  |  |  |
| --- | --- | --- | --- |
| **Letter** | **Date or Time Component** | **Presentation** | **Examples** |
| G | Era designator | [Text](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#text) | AD |
| y | Year | [Year](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#year) | 1996; 96 |
| Y | Week year | [Year](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#year) | 2009; 09 |
| M | Month in year (context sensitive) | [Month](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#month) | July; Jul; 07 |
| L | Month in year (standalone form) | [Month](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#month) | July; Jul; 07 |
| w | Week in year | [Number](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#number) | 27 |
| W | Week in month | [Number](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#number) | 2 |
| D | Day in year | [Number](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#number) | 189 |
| d | Day in month | [Number](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#number) | 10 |
| F | Day of week in month | [Number](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#number) | 2 |
| E | Day name in week | [Text](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#text) | Tuesday; Tue |
| u | Day number of week (1 = Monday, ..., 7 = Sunday) | [Number](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#number) | 1 |
| a | Am/pm marker | [Text](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#text) | PM |
| H | Hour in day (0-23) | [Number](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#number) | 0 |
| k | Hour in day (1-24) | [Number](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#number) | 24 |
| K | Hour in am/pm (0-11) | [Number](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#number) | 0 |
| h | Hour in am/pm (1-12) | [Number](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#number) | 12 |
| m | Minute in hour | [Number](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#number) | 30 |
| s | Second in minute | [Number](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#number) | 55 |
| S | Millisecond | [Number](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#number) | 978 |
| z | Time zone | [General time zone](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#timezone) | Pacific Standard Time; PST; GMT-08:00 |
| Z | Time zone | [RFC 822 time zone](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#rfc822timezone) | -0800 |
| X | Time zone | [ISO 8601 time zone](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#iso8601timezone) | -08; -0800; -08:00 |

Pattern letters are usually repeated, as their number determines the exact presentation:

* **Text:** For formatting, if the number of pattern letters is 4 or more, the full form is used; otherwise a short or abbreviated form is used if available. For parsing, both forms are accepted, independent of the number of pattern letters.
* **Number:** For formatting, the number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this amount. For parsing, the number of pattern letters is ignored unless it's needed to separate two adjacent fields.
* **Year:** If the formatter's [Calendar](file:///C:\docs\html\java\j2sdk180\api\java\text\DateFormat.html#getCalendar--) is the Gregorian calendar, the following rules are applied.
  + For formatting, if the number of pattern letters is 2, the year is truncated to 2 digits; otherwise it is interpreted as a [number](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#number).
  + For parsing, if the number of pattern letters is more than 2, the year is interpreted literally, regardless of the number of digits. So using the pattern "MM/dd/yyyy", "01/11/12" parses to Jan 11, 12 A.D.
  + For parsing with the abbreviated year pattern ("y" or "yy"), SimpleDateFormat must interpret the abbreviated year relative to some century. It does this by adjusting dates to be within 80 years before and 20 years after the time the SimpleDateFormat instance is created. For example, using a pattern of "MM/dd/yy" and a SimpleDateFormat instance created on Jan 1, 1997, the string "01/11/12" would be interpreted as Jan 11, 2012 while the string "05/04/64" would be interpreted as May 4, 1964. During parsing, only strings consisting of exactly two digits, as defined by [Character.isDigit(char)](file:///C:\docs\html\java\j2sdk180\api\java\lang\Character.html#isDigit-char-), will be parsed into the default century. Any other numeric string, such as a one digit string, a three or more digit string, or a two digit string that isn't all digits (for example, "-1"), is interpreted literally. So "01/02/3" or "01/02/003" are parsed, using the same pattern, as Jan 2, 3 AD. Likewise, "01/02/-3" is parsed as Jan 2, 4 BC.

Otherwise, calendar system specific forms are applied. For both formatting and parsing, if the number of pattern letters is 4 or more, a calendar specific [long form](file:///C:\docs\html\java\j2sdk180\api\java\util\Calendar.html#LONG) is used. Otherwise, a calendar specific [short or abbreviated form](file:///C:\docs\html\java\j2sdk180\api\java\util\Calendar.html#SHORT) is used.  
  
If week year 'Y' is specified and the [calendar](file:///C:\docs\html\java\j2sdk180\api\java\text\DateFormat.html#getCalendar--) doesn't support any [week years](file:///C:\docs\html\java\j2sdk180\api\java\util\GregorianCalendar.html#week_year), the calendar year ('y') is used instead. The support of week years can be tested with a call to [getCalendar()](file:///C:\docs\html\java\j2sdk180\api\java\text\DateFormat.html#getCalendar--).[isWeekDateSupported()](file:///C:\docs\html\java\j2sdk180\api\java\util\Calendar.html#isWeekDateSupported--).

* **Month:** If the number of pattern letters is 3 or more, the month is interpreted as [text](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#text); otherwise, it is interpreted as a [number](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#number).
  + Letter M produces context-sensitive month names, such as the embedded form of names. If a DateFormatSymbols has been set explicitly with constructor [SimpleDateFormat(String, DateFormatSymbols)](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#SimpleDateFormat-java.lang.String-java.text.DateFormatSymbols-) or method [setDateFormatSymbols(DateFormatSymbols)](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#setDateFormatSymbols-java.text.DateFormatSymbols-), the month names given by the DateFormatSymbols are used.
  + Letter L produces the standalone form of month names.
* **General time zone:** Time zones are interpreted as [text](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#text) if they have names. For time zones representing a GMT offset value, the following syntax is used:
* *GMTOffsetTimeZone:*
* GMT *Sign* *Hours* : *Minutes*
* *Sign:* one of
* + -
* *Hours:*
* *Digit*
* *Digit* *Digit*
* *Minutes:*
* *Digit* *Digit*
* *Digit:* one of

0 1 2 3 4 5 6 7 8 9

*Hours* must be between 0 and 23, and *Minutes* must be between 00 and 59. The format is locale independent and digits must be taken from the Basic Latin block of the Unicode standard.

For parsing, [RFC 822 time zones](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#rfc822timezone) are also accepted.

* **RFC 822 time zone:** For formatting, the RFC 822 4-digit time zone format is used:
* *RFC822TimeZone:*
* *Sign* *TwoDigitHours* *Minutes*
* *TwoDigitHours:*

*Digit Digit*

*TwoDigitHours* must be between 00 and 23. Other definitions are as for [general time zones](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#timezone).

For parsing, [general time zones](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#timezone) are also accepted.

* **ISO 8601 Time zone:** The number of pattern letters designates the format for both formatting and parsing as follows:
* *ISO8601TimeZone:*
* *OneLetterISO8601TimeZone*
* *TwoLetterISO8601TimeZone*
* *ThreeLetterISO8601TimeZone*
* *OneLetterISO8601TimeZone:*
* *Sign* *TwoDigitHours*
* Z
* *TwoLetterISO8601TimeZone:*
* *Sign* *TwoDigitHours* *Minutes*
* Z
* *ThreeLetterISO8601TimeZone:*
* *Sign* *TwoDigitHours* : *Minutes*

Z

Other definitions are as for [general time zones](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#timezone) or [RFC 822 time zones](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#rfc822timezone).

For formatting, if the offset value from GMT is 0, "Z" is produced. If the number of pattern letters is 1, any fraction of an hour is ignored. For example, if the pattern is "X" and the time zone is "GMT+05:30", "+05" is produced.

For parsing, "Z" is parsed as the UTC time zone designator. [General time zones](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#timezone) are not accepted.

If the number of pattern letters is 4 or more, [IllegalArgumentException](file:///C:\docs\html\java\j2sdk180\api\java\lang\IllegalArgumentException.html) is thrown when constructing a SimpleDateFormat or [applying a pattern](file:///C:\docs\html\java\j2sdk180\api\java\text\SimpleDateFormat.html#applyPattern-java.lang.String-).

SimpleDateFormat also supports localized date and time pattern strings. In these strings, the pattern letters described above may be replaced with other, locale dependent, pattern letters. SimpleDateFormat does not deal with the localization of text other than the pattern letters; that's up to the client of the class.

***Examples***

The following examples show how date and time patterns are interpreted in the U.S. locale. The given date and time are 2001-07-04 12:08:56 local time in the U.S. Pacific Time time zone.

|  |  |
| --- | --- |
| **Date and Time Pattern** | **Result** |
| "yyyy.MM.dd G 'at' HH:mm:ss z" | 2001.07.04 AD at 12:08:56 PDT |
| "EEE, MMM d, ''yy" | Wed, Jul 4, '01 |
| "h:mm a" | 12:08 PM |
| "hh 'o''clock' a, zzzz" | 12 o'clock PM, Pacific Daylight Time |
| "K:mm a, z" | 0:08 PM, PDT |
| "yyyyy.MMMMM.dd GGG hh:mm aaa" | 02001.July.04 AD 12:08 PM |
| "EEE, d MMM yyyy HH:mm:ss Z" | Wed, 4 Jul 2001 12:08:56 -0700 |
| "yyMMddHHmmssZ" | 010704120856-0700 |
| "yyyy-MM-dd'T'HH:mm:ss.SSSZ" | 2001-07-04T12:08:56.235-0700 |
| "yyyy-MM-dd'T'HH:mm:ss.SSSXXX" | 2001-07-04T12:08:56.235-07:00 |
| "YYYY-'W'ww-u" | 2001-W27-3 |

1. Appendix – Double Format Syntax

This chapter explains the syntax of the command line option (and corresponding property) –double-format.

This chapter is copied from the Oracle Java documentation of the java.text.DecimalFormat class from Java version 1.8.

It is also available online at <https://docs.oracle.com/javase/8/docs/api/java/text/DecimalFormat.html>.

Note that the tool set routing to RoundingMode.UNNECESSARY.

### *Patterns*

DecimalFormat patterns have the following syntax:

*Pattern:*

*PositivePattern*

*PositivePattern* ; *NegativePattern*

*PositivePattern:*

*Prefixopt* *Number* *Suffixopt*

*NegativePattern:*

*Prefixopt* *Number* *Suffixopt*

*Prefix:*

any Unicode characters except \uFFFE, \uFFFF, and special characters

*Suffix:*

any Unicode characters except \uFFFE, \uFFFF, and special characters

*Number:*

*Integer* *Exponentopt*

*Integer* . *Fraction* *Exponentopt*

*Integer:*

*MinimumInteger*

#

# *Integer*

# , *Integer*

*MinimumInteger:*

0

0 *MinimumInteger*

0 , *MinimumInteger*

*Fraction:*

*MinimumFractionopt* *OptionalFractionopt*

*MinimumFraction:*

0 *MinimumFractionopt*

*OptionalFraction:*

# *OptionalFractionopt*

*Exponent:*

E *MinimumExponent*

*MinimumExponent:*

0 *MinimumExponentopt*

A DecimalFormat pattern contains a positive and negative subpattern, for example, "#,##0.00;(#,##0.00)". Each subpattern has a prefix, numeric part, and suffix. The negative subpattern is optional; if absent, then the positive subpattern prefixed with the localized minus sign ('-' in most locales) is used as the negative subpattern. That is, "0.00" alone is equivalent to "0.00;-0.00". If there is an explicit negative subpattern, it serves only to specify the negative prefix and suffix; the number of digits, minimal digits, and other characteristics are all the same as the positive pattern. That means that "#,##0.0#;(#)" produces precisely the same behavior as "#,##0.0#;(#,##0.0#)".

The prefixes, suffixes, and various symbols used for infinity, digits, thousands separators, decimal separators, etc. may be set to arbitrary values, and they will appear properly during formatting. However, care must be taken that the symbols and strings do not conflict, or parsing will be unreliable. For example, either the positive and negative prefixes or the suffixes must be distinct for DecimalFormat.parse() to be able to distinguish positive from negative values. (If they are identical, then DecimalFormat will behave as if no negative subpattern was specified.) Another example is that the decimal separator and thousands separator should be distinct characters, or parsing will be impossible.

The grouping separator is commonly used for thousands, but in some countries it separates ten-thousands. The grouping size is a constant number of digits between the grouping characters, such as 3 for 100,000,000 or 4 for 1,0000,0000. If you supply a pattern with multiple grouping characters, the interval between the last one and the end of the integer is the one that is used. So "#,##,###,####" == "######,####" == "##,####,####".

#### Special Pattern Characters

Many characters in a pattern are taken literally; they are matched during parsing and output unchanged during formatting. Special characters, on the other hand, stand for other characters, strings, or classes of characters. They must be quoted, unless noted otherwise, if they are to appear in the prefix or suffix as literals.

The characters listed here are used in non-localized patterns. Localized patterns use the corresponding characters taken from this formatter's DecimalFormatSymbols object instead, and these characters lose their special status. Two exceptions are the currency sign and quote, which are not localized.

|  |  |  |  |
| --- | --- | --- | --- |
| **Symbol** | **Location** | **Localized?** | **Meaning** |
| 0 | Number | Yes | Digit |
| # | Number | Yes | Digit, zero shows as absent |
| . | Number | Yes | Decimal separator or monetary decimal separator |
| - | Number | Yes | Minus sign |
| , | Number | Yes | Grouping separator |
| E | Number | Yes | Separates mantissa and exponent in scientific notation. Need not be quoted in prefix or suffix. |
| ; | Subpattern boundary | Yes | Separates positive and negative subpatterns |
| % | Prefix or suffix | Yes | Multiply by 100 and show as percentage |
| \u2030 | Prefix or suffix | Yes | Multiply by 1000 and show as per mille value |
| ¤ (\u00A4) | Prefix or suffix | No | Currency sign, replaced by currency symbol. If doubled, replaced by international currency symbol. If present in a pattern, the monetary decimal separator is used instead of the decimal separator. |
| ' | Prefix or suffix | No | Used to quote special characters in a prefix or suffix, for example, "'#'#" formats 123 to "#123". To create a single quote itself, use two in a row: "# o''clock". |

#### Scientific Notation

Numbers in scientific notation are expressed as the product of a mantissa and a power of ten, for example, 1234 can be expressed as 1.234 x 10^3. The mantissa is often in the range 1.0 ≤ x < 10.0, but it need not be. DecimalFormat can be instructed to format and parse scientific notation only via a pattern; there is currently no factory method that creates a scientific notation format. In a pattern, the exponent character immediately followed by one or more digit characters indicates scientific notation. Example: "0.###E0" formats the number 1234 as "1.234E3".

* The number of digit characters after the exponent character gives the minimum exponent digit count. There is no maximum. Negative exponents are formatted using the localized minus sign, not the prefix and suffix from the pattern. This allows patterns such as "0.###E0 m/s".
* The minimum and maximum number of integer digits are interpreted together:
  + If the maximum number of integer digits is greater than their minimum number and greater than 1, it forces the exponent to be a multiple of the maximum number of integer digits, and the minimum number of integer digits to be interpreted as 1. The most common use of this is to generate engineering notation, in which the exponent is a multiple of three, e.g., "##0.#####E0". Using this pattern, the number 12345 formats to "12.345E3", and 123456 formats to "123.456E3".
  + Otherwise, the minimum number of integer digits is achieved by adjusting the exponent. Example: 0.00123 formatted with "00.###E0" yields "12.3E-4".
* The number of significant digits in the mantissa is the sum of the minimum integer and maximum fraction digits, and is unaffected by the maximum integer digits. For example, 12345 formatted with "##0.##E0" is "12.3E3". To show all digits, set the significant digits count to zero. The number of significant digits does not affect parsing.
* Exponential patterns may not contain grouping separators.

#### Rounding

DecimalFormat provides rounding modes defined in [RoundingMode](https://docs.oracle.com/javase/8/docs/api/java/math/RoundingMode.html) for formatting. By default, it uses [RoundingMode.HALF\_EVEN](https://docs.oracle.com/javase/8/docs/api/java/math/RoundingMode.html#HALF_EVEN).

#### Digits

For formatting, DecimalFormat uses the ten consecutive characters starting with the localized zero digit defined in the DecimalFormatSymbols object as digits. For parsing, these digits as well as all Unicode decimal digits, as defined by [Character.digit](https://docs.oracle.com/javase/8/docs/api/java/lang/Character.html#digit-char-int-), are recognized.

#### Special Values

NaN is formatted as a string, which typically has a single character \uFFFD. This string is determined by the DecimalFormatSymbols object. This is the only value for which the prefixes and suffixes are not used.

Infinity is formatted as a string, which typically has a single character \u221E, with the positive or negative prefixes and suffixes applied. The infinity string is determined by the DecimalFormatSymbols object.

Negative zero ("-0") parses to

* BigDecimal(0) if isParseBigDecimal() is true,
* Long(0) if isParseBigDecimal() is false and isParseIntegerOnly() is true,
* Double(-0.0) if both isParseBigDecimal() and isParseIntegerOnly() are false.

1. Appendix – SQL Where Syntax

This chapter explains the syntax of the command line option (and corresponding property) –where.

This option contains a WHERE clause that will be used to select rows to export from one table.

The syntax must comply with the SQL syntax requirements and restrictions of TIBCO ActiveSpaces version 4.x.

The syntax is:

[<where clause>]

[<order clause>]

[<limit clause>]

The <where clause>, <order clause> and <limit clause> are optional but must be in the shown order (<where clause> first, then <order clause> then <limit clause>).

**Due to limitations in TIBCO ActiveSpaces:**

* **<order clause> and <limit clause> are not support in TableDelete tool.**
* **<order clause> and <limit clause> are not supported if –table-rowset is true in TableExport tool.**
  1. Where Clause

This clause filters rows by evaluating one or more predicates:

[NOT] <predicate> [AND | OR [ NOT] <predicate>]…

Each predicate tests a column value against another value.

For more details look at <https://docs.tibco.com/pub/as/4.4.0/doc/html/GUID-18D9A680-7AFD-4CBF-A496-117C5F4A783D.html>.

***Examples:***

opcoid = ‘xyz’ and date(transactiondate) = ‘2020-05-20’

* 1. Order Clause

The order clause will sort the extracted rows by column or alias name in ascending or descending order.

[ORDER BY <column\_or\_alias\_name> [ASC | DESC][, <column\_or\_alias\_name> [ASC | DESC]]…

***Examples:***

ORDERBY transactiondate DESC, amount ASC

* 1. Limit Clause

The limit clause will limit the number of extracted rows.

[LIMIT <number>]

***Examples:***

opcoid = ‘xyz’ and date(transactiondate) = ‘2020-05-20’ LIMIT 20

1. Appendix – Usage for Performance Testing

The tools can be used to measure performance of one TIBCO ActiveSpace grid. This chapter explains how to execute various tests.

* 1. How to check the performance of a BW Query-like activity?

To execute a query against one table with an SQL-like filter and to print out the result on standard output, execute:

./TableExport.sh --realm <realm> --table <table> --datetime-filter false --where “<filter>” --table-rowset true –-output stdout

***Examples:***

To search for rows containing an externaltransactionid equal to 123, do:

./TableExport.sh –-realm <realm> --table transactions --datetime-filter false –-where “externaltransactionid=’123’” –-table-rowset true –-columns transactionid,externaltransactionid,transactiondate

* 1. How to check the performance of one single GET?

If you execute the TableImport.sh tool with an input file, each line of the file will upsert one row in the table. The tool will execute a GET first before updating the row.

So, you can provide an input file containing only one transactionid to measure the performance of one GET operation.

***Example:***

To execute one single GET:

echo transactionid > /tmp/input.csv

echo 123 >> /tmp/input.csv

./TableImport.sh –-realm <realm> --table transactions --input /tmp/input.csv –key-columns transactionid

***Notes:***

* You must indicate the name of the column that contains primary keys for the table.
* If you add –batch-row-count 0 to the command line, the tool will not update any row and only perform GET API call, no PUT or PUTALL API call at all.
  1. How to run multiple GETs?

You can run the TableImport.sh tool to execute multiple GETs by putting multiple transactionids into each line of the input file.

It is possible to use TableExport to export all existing transactionid as well.

***Example:***

Extract all transactionids from the table:

./TableExport.sh –-realm <realm> --table transactions --output /tmp/ids –-datetime-filter false –-columns transactionid –-output-header-value transactionid –-max-rows 2000000

This forces the tool to write into a single file /tmp/ids\_1-1.csv all transactionids with the first line as “transactionid”.

Then you can reuse the /tmp/ids\_1-1.csv file to simulate multiple GETs (ensure debug is on to trace the GET latency on standard output and log file):

./TableImport.sh –-realm <realm> --table transactions --input /tmp/ids\_1-1.csv

The tool will stop when all the transactionids have been searched.

The same GETs can be executed on multiple threads (each thread runs the search of all transactionids):

./TableImport.sh –-realm <realm> --table transactions --input /tmp/ids\_1-1.csv –-threads 4

***Notes:***

* There is no delay inside the tool so the execution will constantly execute GETs and PUTs without any delay or pausing against the grid.
* Due to a defect in the tool (that will be fixed in version 0.4.4), in the file to import, the transactionid MUST be the first column and the first file line MUST contain transactionid in first position and in lowercase.
  1. How to simulate GETs and PUTs?

As for multiple gets, you can export all data or several columns from the table and then run the TableImport.sh to upsert the data again. By selecting batch-row-count as 1, you simulate a sequence of GET+PUT operation against the grid.

***Example:***

Export a few columns of all rows into a single file:

./TableExport.sh –-realm <realm> --table transactions --output /tmp/all –-datetime-filter false –-columns “transactionid,transactiondate,amount” –-output-header-value transactionid,transactiondate,amount –-max-rows 2000000

Then run TableImport.sh on one or multiple threads to simulate GET and PUT:

./TableImport.sh –-realm <realm> --table transactions --input /tmp/all\_1-1.csv –-batch-row-count 1

Or on multiple threads:

./TableImport.sh –-realm <realm> --table transactions --input /tmp/all\_1-1.csv –-batch-row-count 1 –-threads 4

***Notes:***

* There is no delay inside the tool so the execution will constantly execute GETs and PUTs without any delay or pausing against the grid.
* Due to a defect in the tool (that will be fixed in version 0.4.4), in the file to import, the transactionid MUST be the first column and the first file line MUST contain transactionid in first position and in lowercase.
* TableDelete