EXPONAM BUILDER CLI REFERENCE

Version 2.5

OVERVIEW

The Exponam Builder CLI is the command line utility for creating Exponam .BIG files from delimited files. This document describes the CLI's usage and features.

Determine which CLI is right for you

The CLI is available in two forms: as an executable application for use on Microsoft Windows, and as a jar file for use with Java on any platform.

> *The rest of this document describes the CLI features in terms of the Java CLI only. The features and options available within the CLI are identical with both. For any questions regarding the Windows CLI, please contact info@exponam.com.*

Obtaining the CLI

The CLI is available as a free download from Exponam.

1) Navigate to https://www.exponam.com and click the Developer Tools button.
2) Add the "Exponam Builder Command Line Interface (Java)" to your cart.
3) Proceed to checkout.
4) Enter the details on the checkout form and place the order.
5) You will receive an email with a download link for the jar file.
6) Download the jar file to an appropriate location on your computer.

Obtaining a license file

To unlock the full capabilities of the CLI, obtain a license file from Exponam. Contact Exponam at either:

- https://www.exponam.com/#contact, or
- sales@exponam.com

In the absence of a license file, the CLI is still functional, but narrows its capabilities to a demonstration mode. Without a license file, the CLI will:

- Apply an upper limit to the number of rows in the output file of 3 million.
- Restrict encryption.
- Restrict the incorporation of custom story files, instead applying an Exponam-standard story file to each file.
- Mark the output file producer as Exponam.

Using the CLI

In summary, the CLI creates output .BIG files from input delimited files according to mapping information held in sheet definition files. A .BIG file contains one or more sheets of data, where a sheet represents a table. Each sheet

can be populated with data from one or more input delimited files. Optionally, the CLI can apply properties and story files to the file, with public or private visibility.

Before enumerating the CLI parameters, a brief explanation of terms:

- Delimited files are text files containing tabular data, with a special character to separate fields. Common examples include csv (comma-separated value) files, pipe-delimited files (that use the | character between fields), and tab-delimited files.
- A sheet definition file provides the mappings for the columns in the delimited file to the Exponam format. A sheet definition can be produced for you using Exponam Builder, a Windows GUI application that serves as an optional adjunct to the CLI. Alternately, a sheet definition file can be created using any text editor, following the syntax documented in the appendix.
- Properties are optional metadata that you can apply to the document. Each property has a name and a value.
- Story files are files that you can embed in the document.

The parameters that can be passed to the CLI fall into the following categories:

- Sheets
  - `-n[ame][#] name` : an optional name for the sheet. An optional zero-based sheet number is appended if the .BIG file is to have multiple sheets.
  - `-s[heetDefinition][#] filename` : the sheet definition file describing the sheet.
  - `-i[nput][#] filename` : a delimited file containing the data to be ingested into the sheet. This parameter can be repeated if multiple delimited files are to be ingested into a given sheet.
- Data ingest treatment
  - `-skip` : skip a row if one or more fields within the row have invalid data
  - `-zero` : set fields with invalid data to an empty or null value
- Properties and stories
  - `-r name value` or `-cleartextProperty name value` : a property that will have public visibility
  - `-R name value` or `-secureProperty name value` : a property that will have private visibility
  - `-m filename` or `-cleartextStoryFile filename` : a story file with will have public visibility
  - `-M filename` or `-secureStoryFile filename` : a story file that will have private visibility
- Optimization level
  - `-optimizationLevel level` : degree of optimization to be performed during file generation, per the following (default is 1)
    - 1 : fast output generation with some file size reduction - simple data exchange where speed and small output size are both priorities
    - 2 : smallest possible output file size - simple data exchange where small output size is the priority
    - 3 : small output size with minimal search indexes - data exchange and edge data with minor gains in query performance
    - 4 : small output size with minimal search indexes - data exchange and edge data with incrementally greater gains in query performance
    - 5 : small output size with minimal search indexes - data exchange and edge data with incrementally greater gains in query performance
    - 6 : medium output size with light search indexes - data exchange and edge data with incrementally greater gains in query performance
    - 7 : medium output size with light search indexes - data exchange and edge data with incrementally greater gains in query performance
    - 8 : medium output size with light search indexes - data exchange and edge data with incrementally greater gains in query performance
    - 9 : medium output size with light search indexes - data exchange and edge data with incrementally greater gains in query performance
    - 10 : larger output size with moderate search indexes - for more demanding query environments
    - 11 : larger output size with moderate search indexes - for even more demanding query environments

        - 12 : larger output size with moderate search indexes - for even more demanding query environments
        - 13 : larger output size with comprehensive search indexes - for stand-alone data warehouse extracts and similar usages
        - 14 : larger output size with comprehensive search indexes - for stand-alone data warehouse extracts and similar usages
        - 15 : largest output size with fullest search indexes - for the most demanding query environments
- Provenance level
    - `-provenanceLevel level` : governs the level of detail emitted into provenance structures, per the following (default is 1):
        - 0 : summary only
        - 1 : summary and high level details
        - 2 : summary and all details
- Output
    - `-p[assword] text` : an optional password that can be supplied to encrypt the file
    - `-o[utput] filename` : the name for the output .BIG file
- License
    - `-l[icense] filename` : an optional license file that makes available the full capabilities of the CLI
- Other parameters
    - `-v[erbose]` : perform verbose logging
    - `-h[elp]` : write the available parameters and descriptions to the console

Examples

(1) Simple generation of a single sheet from a single csv file with no license file

-i data.csv -s sheet.xml -o output.big

(2) Generation with multiple files contributing to a single sheet with no license file

-i data1.csv -i data2.csv -s sheet.xml -o output.big

(3) Generation of two named sheets, each from its own csv file, with no license file

-i0 summary.csv -s summarySheet.xml -n0 Summary -i1 details.csv -s detailSheet.xml -n1 Details -o output.big

(4) Simple generation of a single sheet from a single csv file with a license file

-i data.txt -s sheet.xml -l license.lic -o output.big

(5) Simple generation with optimization level 2 (guaranteed smallest size output file) and provenance level 2 (summary and all details related to provenance)

-i data.txt -s sheet.xml -optimizationLevel 2 -provenanceLevel 2 -o output.big

(6) Generation with public and private properties and story files

-i data.csv -s sheet.xml -r Department Finance -R “Department Type” “Cost Center” -s LegalDisclaimer.rtf -S DataConfidentialityStandards.doc -o output.big

Additional Resources

Exponam Builder, a Windows GUI application, offers several capabilities to help you as you begin to use the Builder CLI. As with Builder CLI, the Builder is available as a free download. If you opt to obtain a license file, that license can be used with both the Builder and Builder CLI.

The first situation where you might use Builder is in generating sheet definition files. Provided you have a sample data file that is representative of the type of data you'll be working with, it's as simple as dragging-and-dropping that file on to Builder and then following a short sequence of steps in a wizard window. Builder will infer the data types from the file you provide, and allow you to modify its results: renaming columns, changing column types, reviewing the data that Builder sampled to arrive at its inferences, etc. Builder will then produce the sheet definition file, which you can subsequently use in both Builder and Builder CLI.

The second situation is the construction of the actual command line parameters. Depending on your objectives for the output file, the command line can become complex. Builder simplifies this for you. When you use Builder's wizard for generating a file, it will show you the command line that corresponds to the options you've selected. You can copy that command line to the clipboard for later use. Of course, Builder can also generate the .BIG file for you directly. But being able to use Builder to experiment with the options and arrive at the desired command line parameters can make it that much easier to then use those command line parameters on a platform such as Linux.

APPENDIX – SHEET DEFINITION FILES

The sheet definition file uses an XML format to describe the contents and interpretation of a delimited file. It is comprised of the following elements:

- Tag “headerRow”
  - attribute “present”
    - values “true” or “false”
- Tag “delimiter”
  - attribute “value”
    - values “,” or “|” or “\t”
- Tag “columns”
  - Tag “column”
    - attribute “name” column name
    - attribute “type”, values:
      - Char (a single 16-but Unicode character)
      - String
      - Float (32-bit IEEE 754 floating point)
      - Double (64-bit IEEE 754 floating point)
      - BigDecimal (intended for arbitrary-precision decimal arithmetic)
      - Byte (8-bit signed two’s complement integer with a minimum value of -128 and a maximum value of 127, inclusive)
      - Short (16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767, inclusive)
      - Integer (32-bit signed two's complement integer, which has a minimum value of -2^{31} and a maximum value of $2^{31}-1$)
      - Long (64-bit two's complement integer. The signed long has a minimum value of -2^{63} and a maximum value of $2^{63}-1$)
      - BigInteger (for arbitrarily large integer arithmetic)
      - Date,
      - DateTime,
      - Time,
      - Boolean
    - attribute “inputMask” (applied to inbound data; semantics depend on the target data type for the column)
    - attribute “displayMask” (applied on rendering column values in string representation; semantics depend on the data type for the column)

As an example, the following is the sheet definition file for the FOREX sample data file that comes bundled with Exponam Explorer:

```
<xml>
        <headerRow present="true"/>
        <delimiter value=","/>
        <columns>
                <column name="Currency Pair" type="String"/>
                <column name="Quote Date" type="Date"/>
```

```
            <column name="Quote Time" type="Time"  inputMask="mm:ss.f" displayMask="HH:mm:ss.f"/>
            <column name="Bid Price" type="Float"  displayMask="#,###0.0000###;(#,###0.0000###);-"/>
            <column name="Ask Price" type="Float"  displayMask="#,###0.0000###;(#,###0.0000###);-"/>
            <column name="Contributor Code" type="String"/>
            <column name="Region Code" type="String"/>
            <column name="City Code" type="String"/>
        </columns>
</xml>
```