MMOL Tabular Data: User Guide

Getting Started

This user guide is intended to help you get started using the tabular versions of the Medieval Manuscripts in Oxford Libraries catalogue data, and to give you examples of what you can do with the data.

This guide consists of the following sections:

Before you get started, please read the following notes on the limitations on the quality and completeness of the data.

Notes on Data Quality

The Bodleian's Western medieval manuscript catalogues, for which this processor is primarily designed, remains a work in progress both in the encoding into TEI-XML of data which have already been recorded, and in the recording of new data. The absence of information regarding e.g. decoration or musical notation should therefore not be taken as an indication that no such features exist for a given manuscript, only that these data have not been recorded in the digital catalogues.

Similarly, many of the datapoints presented through the outputs of this processor are the work of previous generations of librarians and scholars who had cataloguing priorities and practices that are different to those of modern cataloguers. In particular, in previous cataloguing cycles, measurements have often been rounded to the nearest 5mm or quarter/eighth of an inch, and features such as decoration have been described using assessments of aesthetic and material quality (e.g. "Good border"), rather than through dispassionate descriptions of their form and contents.

Users should therefore be aware of these limitations when engaging with the outputs of this processor, whether using the default outputs or custom ones.

1. The Data

The data outputs of this project are intended to be used by researchers of all levels, from those with no knowledge of the underlying TEI-XML data, through to experienced users who nevertheless want to access the data in a format that enables cross-comparison.

You do not need to be able to run the processor in order to access the outputs. They can be found in the `tabular_data/output` folder, where there are two kinds:

1. Collection: these are the data which have been extracted from the collection files, and so predominantly contain information about manuscripts and manuscript parts.

2. Authority: these are the data which have been extracted from the authority files which accompany the collection, and so predominantly contain information about people, places, and organisations associated with the collections.

The data are presented in three formats:

1. A single XLSX (Excel) file, containing tabs for each of the configured outputs. This is the recommended format, as the data in each tab are grouped into sections, with accompanying notes explaining the content of each column.
2. A set of CSV files, each one corresponding to a tab in the XLSX output.
3. A set of JSON files, each one corresponding to a tab in the XLSX output.

Details of the default contents of each tab of the Excel files (or individual CSV/JSON files) can be found below.

Collection

00_overview

Each row of the output file corresponds to a manuscript unit, be that a part (`msPart`) or a whole MS (`msDesc`). The output file contains a variety of basic information about the unit in question, including details of its origin, provenance, acquisition, contents, codicology, palaeography, and binding, and a number of measurements.

01_records

Each row of the output file corresponds to an individual `additional` element within the collection data, and contains enhanced metadata about the object in question, as well as details of the record history, the availability of the object and any digital surrogates, and printed and digital resources associated with the object.

02_binding

Each row of the output file corresponds to an individual `binding` element or a `dimensions` element with `@type="binding"`. This contains overview information about the object's binding, as well as dating and measurement information.

03_measurements

Each row of the output file corresponds to an individual set of measurements for a manuscript unit, as encoded within either (or both) of `layout` and `extent` elements. This contains information about the manuscript part's extent, as well as measurements and other details associated with rolls, fragments, leaves, columns, intercolumn widths, the written area, and ruling.

04_hands

Each row of the output file corresponds to an individual `handNote` element for a manuscript unit. This contains overview information about the hand in question, as well as details of any people associated with the hand, any locus information, details of dating, and palaeographic details.

05_contents

Each row of the output file corresponds to an individual work or part of a work contained within a manuscript unit in the collection. This contains locus and length information,

alongside details of the work's title subject, author, language, and paratext, as well as any bibliographic references associated with it in the collection file.

06_music

Each row of the output file corresponds to an individual description of musical notation encoded within a `musicNotation` element. This contains overview information about the notation in question, as well as locus and dimension information, and details of any people or bibliographic references associated with the notation.

07_decoration

Each row of the output file corresponds to an individual description of decoration encoded within a `decoNote` element. This contains overview information about the decoration in question, including its type, as well as locus and date information and details of any people, organisations, places, or bibliographic references associated with the decoration.

08_origins

Each row of the output file corresponds to an individual manuscript unit, and the information about its origins contained within the `origin` element (manuscript parts inherit data from their containing manuscript, should no other information be present). This contains overview information about the origin of the unit in question, as well as locus and date information, and details of any people, organisations, or bibliographic references associated with the manuscript unit's origins.

09_additions

Each row of the output file corresponds to an individual description of additions to a given manuscript unit, as given within the `additions` element. This contains overview information about the addition in question, as well as locus information, and details of any people or bibliographic references associated with the addition.

10_provenance_acquisition

Each row of the output file corresponds to an individual set of information given about the history of a manuscript unit, as contained within a `provenance` or `acquisition` element. This contains overview information about the historical detail in question, as well as date information, and details of any people, organisations, or places associated with the historical detail, with additional details given about former owners.

Authority

orgs

Each row of the output file corresponds to an organisation named within the authority file places.xml.

persons

Each row of the output file corresponds to a person named within the authority file persons.xml.

places

Each row of the output file corresponds to a place named within the authority file places.xml.

works

Each row of the output file corresponds to a work named within the authority file works.xml.

2. Worked Examples using the Spreadsheets

Follow along with these worked examples to gain an idea of the sorts of research questions that the MMOL Tabular Data can help you answer.

The examples here are given for Excel for Mac version 16, and reflect the state of the tabular data at the time this guide was compiled (July 2025), so the exact process and your eventual results may differ from what you see here.

Example 1: Cross-Comparison of Manuscript Contents

Question: What texts appear alongside the works of Catullus in the manuscripts catalogued in MMOL?

In order to answer this question, we need to narrow the data down to just those manuscripts containing the works of Catullus, so that we can look at the other authors that he appears alongside. This is relatively easy in spreadsheet format:

1. Open 'collection_data.xlsx'.
2. In the '00_overview' tab, navigate to the 'contents' section and find the 'author(s) referenced' column.
3. Click/tap on the downwards facing arrow on the right hand side of the column heading, to open the Filter options.
4. In the Search bar, enter 'Catullus'.
5. The only manuscripts visible should now be the relatively few containing works of Catullus.

By looking at the 'author(s) referenced' column, we are able to reach our conclusion. In the July 2025 version of the data, at least, Catullus' work appears in 5 manuscripts alongside the work of Tibullus, once each alongside Ovid, Propertius, and Pseudo-Virgil, and twice on its own.

(To clear existing filters from one query to the next, go to the 'Data' menu, followed by 'Clear Filters'.)

Example 2: Quantitative Codicology

Question: Are the leaf heights of 15th-century paper manuscripts in the catalogue more standardised than those of 15th-century manuscripts on parchment?

To answer this question, we need to narrow the data down in a more multi-dimensional way. We need to find the page heights of all the 15th-century paper manuscripts in the catalogue for which there are sufficient data, and then do the same for all the 15th-century parchment manuscripts. Here are the steps to follow:

1. Open 'collection_data.xlsx'.
2. In the '00_overview' tab, navigate to the 'origin' section and find the 'not before' column.
3. Click/tap on the downwards facing arrow on the right hand side of the column heading, to open the Filter options.

4. Since we're only interested in the 15th century, use the number filtering options ('Choose One' on Mac) to select values that are greater than or equal to 1400.
5. Navigate to the 'not after' column in the same section, and this time filter to select values that are less than or equal to 1500.
6. Navigate to the 'form' column in the 'codicology' section, and open the Filter options.
7. Untick all the options except for 'codex' (since we only want to look at whole manuscripts).
8. Navigate to the 'support' column in the same section, and open the Filter options, selecting just 'chart' (=paper).
9. Navigate to the 'max height' column in the 'leaves' section. This is the column we will use as our measurement, but you could add a column containing an average of the 'max height' and 'min height' columns if you so wished, and continue with those data.
10. In the Filter options for this column, deselect blank cells (this is at the top of the list on Windows, and at the bottom of the list on Mac).
11. To visualise the data, select the whole column by clicking/tapping on the column letters ('BH' in the July 2025 data), and then go to Excel's 'Insert' tab.
12. Selecting 'Recommended Charts' and then 'Histogram' should give you a quick and easy visualisation of the distribution of the leaf heights.
13. Rename this chart as '15th-Century Paper MSS Leaf Heights' or similar, and save a screenshot of the chart (since Excel updates charts as you change the Filter settings).
14. We now have our data for paper manuscripts. To get the same for parchment manuscripts, simply leave the other Filter options unchanged and navigate back to the 'support' column in the 'codicology' section, and replace 'chart' with 'perg' (=parchment).
15. Return to the 'max height' column in the 'leaves' section, and repeat steps 11–13 to produce the second chart.

By comparing the charts, we can therefore answer our question. The leaf heights of 15th-century parchment manuscripts in the MMOL catalogue appear roughly in line with a normal distribution, suggesting that their values coalesce around an average, but that this is not part of a standardised process. The paper manuscripts, meanwhile, appear to fit into two main categories of page height, suggesting certain standard paper sizes are common in the collection.

(To clear existing filters from one query to the next, go to the 'Data' menu, followed by 'Clear Filters'.)

Example 3: Quantitative Palaeography

Question: What are the 10 most common script types found in manuscripts catalogued in MMOL?

To answer this question, we need to produce counts of manuscripts according to their script type, using Excel's Pivot Table function. Here are the steps to follow:

1. Open 'collection_data.xlsx'.
2. In the '04_hands' column, select all the data by going to Edit > Select All, or by pressing Command/Control + A.
3. In the 'Insert' tab, select 'Pivot Table', and press 'OK'.
4. A new worksheet should appear containing a blank pivot table. In the 'Field Name' section of the 'PivotTable Fields' menu that should have appeared on the right, select 'shelfmark' and 'script'.

5. Drag 'shelfmark' from the 'Rows' into the 'Values' column, to gain a count of the manuscripts containing each script type.
6. On the pivot table itself, click/tap on the downwards facing arrow on the right hand side of the 'Row Labels' column heading, to open the Filter options.
7. In the Filter options, deselect blank cells (this is at the top of the list on Windows, and at the bottom of the list on Mac).
8. Also in the Filter options, select 'Count of shelfmark' in the 'Sort by:' dropdown list, and then click 'Descending', to ensure that the most frequent scripts appear at the top of the list.
9. To visualise the data, select any cell in the pivot table and then go to Excel's 'Insert' tab. Selecting a 2D pie chart will allow you to show the data as the proportion of a whole, excluding the more unusual or highly specified script types.

By looking at this pie chart and at the pivot table, we can say that, with just over a quarter of examples, Northern Textualis is the most common script type given in the collection (as defined by Albert Derolez (2003), *The Palaeography of Gothic Manuscript Books from the Twelfth to the Early Sixteenth Century*). However, the limited volume of data available in the catalogue (1,997 entries as of July 2025) means that we must be cautious about drawing too many conclusions from these data about the whole catalogue.

3. Setting up the Column Merge App (Intermediate)

The Column Merge app has been produced to allow you to combine columns from across different sheets of the output, without needing to engage in complex Excel formulae or running much code. This is useful when your requirements are more complex than have been accounted for in the default output data.

Follow the steps below to set up and run the column merge app on Mac, Windows, or Linux (steps 1–6 are shared with the Processor, as below).

1. Clone the Repository

Clone the repository to your local machine using git, or download the repository directly.

2. Open a Terminal or Command Prompt

Navigate to the `tabular_data` directory in your terminal (Mac/Linux) or Command Prompt/PowerShell (Windows), or open a new Terminal/Command Prompt window at that directory.

3. Check Python and pip Installation

In the Terminal or Command Prompt, run the following commands to make sure that Python 3 and pip are installed:

Mac/Linux:

```
python3 --version
pip3 --version
```

Windows:

```
python --version
pip --version
```

If Python is not installed, download Python from https://www.python.org/downloads/

4. Create a Virtual Environment

Run the following commands to create a virtual environment:

Mac/Linux:

```
python3 -m venv .venv
```

Windows:

```
python -m venv .venv
```

5. Activate the Virtual Environment

Mac/Linux:

```
source .venv/bin/activate
```

Windows (Command Prompt):

```
.venv\Scripts\activate.bat
```

Windows (PowerShell):

```
.venv\Scripts\Activate.ps1
```

6. Install Dependencies

Once the virtual environment is active, install the required Python packages:

```
pip install -r requirements.txt
```

7. Run the Web Application

Start the web server with:

```
python column_merge.py
```

You should see the following output:

```
Starting server at http://127.0.0.1:5000/
Press CTRL+C to quit.
```

8. Access the App

To access and use the app, open your web browser and go to:

```
http://127.0.0.1:5000/
```

9. Use the App

Follow the instructions on each page to perform the merge and download the resulting file. For worked examples, see below.

10. Stop the Server

To stop the app, return to the terminal and press:

```
Control + C
```

4. Worked Examples Using the Column Merge App (Intermediate)

Example 1: Find all the IIIF manifest URIs for manuscripts containing musical notation.

Suppose you are interested in performing a digital analysis of a large corpus of musical manuscripts, and you therefore need to find those manuscripts with IIIF manifest URIs which also contain musical notation. This is not immediately possible with the default output data, since the IIIF manifest URIs are contained in 01_records, while the details of musical notation are predominantly included in 06_music. To combine the two sets of information, we can follow the steps below:

1. With the Column Merge app running (see the steps above), navigate to http://127.0.0.1:5000/ in your web browser.
2. Click/tap on 'Select files' and navigate to the `tabular_data/output/collection/csv` directory, where the CSV output files for the collection are stored.
3. Select `01_records.csv` and `06_music.csv`, the two files we want to merge.
4. Click/tap 'Continue', and wait for the files to process.
5. On the next page ('Select Columns to Merge'), we need to select the columns that we want to retain from each CSV file. The 'Merge Key' (the column used to combine the files) can stay as 'metadata: part ID', since this will correctly align the files based on their shared identifiers. In the left-hand column ('01_records.csv'), tick the box labelled 'surrogates: full iiif manifest' to tell the app that you want to retain this column.
6. In the right-hand column, select the columns from 'overview: music' through to 'dimensions: unit', to retain information that might be useful for a digital study of musical notation.
7. Click/tap 'Continue'.
8. On the next page ('Select Required Values'), we need to select the columns for which data must appear in a given row in order for that row to be included. Given that we're only interested in manuscripts with a IIIF manifest file, tick 'surrogates: full iiif manifest' in the left-hand column.
9. Also select 'overview: music' in the right-hand column, to select only those manuscripts for which details of musical notation have been included in the catalogue.
10. Click/tap 'Merge'.
11. On the next page ('Merged Data Preview'), you will see a preview of the merged table, with the merge key (in this case, the manuscript part ID) in the first column, followed by the IIIF manifests and then details of the musical notation.
12. To download the data in either CSV or JSON format, click/tap on the appropriate button at the top of the page.

Example 2: Find available WikiData references for people referenced as authors.

Suppose now that you are interested in connecting the data found in the MMOL catalogue about authors to other sources of linked open data, such as Wikidata, and therefore want to know their Wikidata references. This is not immediately possible with the default output data, since details of authors are predominantly included in 05_contents, while Wikidata references are stored in the 'persons' section of the authority output. To combine the two sets of information, we can follow the steps below:

1. With the Column Merge app running (see the steps above), navigate to http://127.0.0.1:5000/ in your web browser.
2. Click/tap on 'Select files' and navigate to the `tabular_data/output/collection/csv` directory, where the CSV output files for the collection are stored.
3. Select `05_contents.csv`, the first of the two files that we want to merge.
4. Click/tap on 'Select files' again, and navigate to the `tabular_data/output/authority/csv` directory, where the CSV output files for the authority data are stored.
5. Select `persons.csv`, the second of the two files that we want to merge.
6. Click/tap 'Continue', and wait for the files to process. `05_contents.csv` is a large file, so this may take some time.
7. On the next page ('Select Columns to Merge'), we need to select the columns that we want to retain from each CSV file. We want to combine the tables based on the identifiers associated with the authors, so change the 'Merge Key' value for `05_contents.csv` to 'author: ID(s)', and change the value for `persons.csv` to 'identifiers: ID'.
8. In the left-hand column, select 'author: standard name(s)', to include the author's name.
9. In the right-hand column, select 'identifiers: URI (Wikidata)' which is the Wikidata reference we want to associate with each author.
10. Click/tap 'Continue'.
11. On the next page ('Select Required Values'), we need to select the columns for which data must appear in a given row in order for that row to be included. This time, we can tick 'Select all columns' and 'Exclude duplicates', since we only want entries containing all three values, and there is no value in this case of retaining duplicate rows.
12. Click/tap 'Merge'.
13. On the next page ('Merged Data Preview'), you will see a preview of the merged table, with the merge key (in this case, the person identifier) in the first column, followed by the text associated with a given author, their name, and their Wikidata reference.
14. To download the data in either CSV or JSON format, click/tap on the appropriate button at the top of the page.

5. Using the Processor (Intermediate)

If you want to run the processor on your own machine, either over the MMOL data or on another dataset, follow the steps below (steps 1 and 4–8 are shared with the Column Merge app, as above). These instructions can also be found in the `processor_setup_instructions.txt` file in the `tabular_data` directory.

1. Clone the Repository

Clone the repository to your local machine using git, or download the repository directly.

2. Update the Source Files (if required)

If you wish to run the processor over a subset of the MMOL data, or over any other dataset, you will need to change the data contained within the repository. Any .xml file found in the collection directory or its sub-directories will be treated as a file containing data about an individual manuscript unit. Any .xml file found in the root directory of the repository will be treated as authority files, and will be assumed to contain information on entities named in the collection files, linked using UIDs.

3. Modify the `_global_config.py` and Config Files (if required)

See the next section or the `read_me.txt` file in the `config` directory for further details.

4. Open a Terminal or Command Prompt

Navigate to the `tabular_data` directory in your terminal (Mac/Linux) or Command Prompt/PowerShell (Windows), or open a new Terminal/Command Prompt window at that directory.

5. Check Python and pip Installation

In the Terminal or Command Prompt, run the following commands to make sure that Python 3 and pip are installed:

Mac/Linux:

```
python3 --version
pip3 --version
```

Windows:

```
python --version
pip --version
```

If Python is not installed, download Python from https://www.python.org/downloads/

6. Create a Virtual Environment

Run the following commands to create a virtual environment:

Mac/Linux:

```
python3 -m venv .venv
```

Windows:

```
python -m venv .venv
```

7. Activate the Virtual Environment

Mac/Linux:

```
source .venv/bin/activate
```

Windows (Command Prompt):

```
.venv\Scripts\activate.bat
```

Windows (PowerShell):

```
.venv\Scripts\Activate.ps1
```

8. Install Dependencies

Once the virtual environment is active, install the required Python packages:

```
pip install -r requirements.txt
```

9. Run the Processor Script

Run the main script to process the data:

```
python processor.py
```

10. Output

After processing is complete, the results will be saved in the `output` folder.

6. Configuring the Processor (Advanced)

If you wish to customise the configuration files for your own purposes, you can do so by modifying the CSV files in the 'config' directory. This section of the guide contains information on the process of extraction, as well as what information is expected for both the 'collection' and the 'authority' configuration files.

Processing Principles

Any .xml file found in the root directory of the repository will be treated as authority files, and will be assumed to contain information on entities named in the collection files, linked using UIDs. Any XML file found in the collection directory or its sub-directories will be treated as a file containing data about an individual manuscript unit.

All configuration files will be processed in alphabetical order by filename, and this is also the order in which the output tabs will appear in the output XLSX files. If you wish to modify this order, include numbers at the beginning of your config filenames.

Authority files will be processed before collection files, to ensure that the data required for any authority lookups have already been extracted.

Authority Configuration Files

Any CSV file found in the tabular_data/config/authority directory will be treated as configuration files for authority output files.

For details of what the default configuration files contain, see the earlier section of the user guide.

Individual rows in the .csv configuration file will correspond to individual columns in the output file. All authority configuration files must begin with a column containing a unique identifier, in order for authority lookups to be possible.

In order for the data extraction to be successful, each .csv configuration file must have the following columns:

section

A top-level heading to group different output columns together. Consecutive output columns with the same section heading will be grouped into a single section within the XLSX output. This field is optional, but advised.

heading

A heading for the output column, within the previously designated section.

auth_file

The authority input file (without suffix) from which the data should be extracted.

xpath

The XPath query that should be used to extract the data for the present output column. Every XPath query in a given authority file must return exactly the same number of values as any other, else the code will fail. `string()` and `string-join()` are good ways of forcing multiple (or zero-length) values to be returned as a single value. XPath up to and including version 3.0 is supported, with the exception of FLOWR loops and conditional (e.g. if/else) statements.

separator

The string separator that should be used when the XPath returns multiple values but the code expects only one. The default values are as below, which can be modified or expanded in _global_config.py:

- "default" (returns "; " – this will also be used if a given separator field is empty or unrecognised)
- "comma"
- "semi-colon"
- "space"
- "empty" (no separator)

format

The desired format of the data returned in the current column. The default values are as below. On the expectation that the input data may not always consistently meet the requirements of a given data format (not least the fact that Excel struggles with dates before 1900), data values which fail to strictly adhere to the chosen data format will usually be silently formatted as text.

- "text" (formats as string)
- "number" (formats as double)
- "date" (formats as date if before 1950, else date-like string)
- "boolean" (true/false)
- "percentage" (formats decimal as percentage, rounded to two decimal places)

comment

An optional comment explaining the output of the XPath query, which (if present) will be included as a note attached to the relevant column heading in the XLSX output file.

The authority configuration files result in output files in CSV and JSON format of the same name, as well as a combined XLSX file with tabs corresponding to each configuration file. These files are stored in the `tabular_data/output/authority` directory.

Collection Configuration Files

Any CSV file found in the tabular_data/config/collection directory will be treated as configuration files for collection output files.

For details of what the default configuration files contain, see the earlier section of the user guide.

By default, all output files are configured to begin with columns containing comparable metadata such as shelfmark and the name of the collection file, to allow cross-comparison.

Individual rows in the .csv configuration file will correspond to individual columns in the output file. In order for the data extraction to be successful, each .csv configuration file must have the following columns:

section

A top-level heading to group different output columns together. Consecutive output columns with the same section heading will be grouped into a single section within the .xlsx output. This field is optional, but advised.

heading

A heading for the output column, within the previously designated section.

xpath

The XPath query that should be used to extract the data for the present output column. Every XPath query in a given authority file must return exactly the same number of values as any other, else the code will fail. `string()` and `string-join()` are good ways of forcing multiple (or zero-length) values to be returned as a single value. XPath up to and including version 3.0 is supported, with the exception of FLOWR loops and conditional (e.g. if/else) statements. If you wish to perform an 'authority lookup' and extract data from one of the authority output files, then this query must result in an identifier that corresponds with the identifier given in the first column of the relevant authority output file.

auth_file

The name of the authority output file from which the data should be extracted, if you are performing an authority lookup. Else this can be left blank.

auth_section

The top-level heading in the authority output file within which the desired lookup column can be found, if you are performing an authority lookup. This can be left blank if no authority lookup is being performed, or if the authority file has no section headings.

auth_heading

The heading in the authority output file of the desired lookup column. This can be left blank if no authority lookup is being performed.

separator

The string separator that should be used when the XPath returns multiple values but the code expects only one. The default values are as below, which can be modified or expanded in _global_config.py:

- "default" (returns "; " – this will also be used if a given separator field is empty or unrecognised)
- "comma"
- "semi-colon"
- "space"
- "empty" (no separator)

format

The desired format of the data returned in the current column. The default values are as below. On the expectation that the input data may not always consistently meet the requirements of a given data format (not least the fact that Excel struggles with dates before 1900), data values which fail to strictly adhere to the chosen data format will usually be silently formatted as text.

- "text" (formats as string)
- "number" (formats as double)
- "date" (formats as date if before 1950, else date-like string)
- "boolean" (true/false)
- "percentage" (formats decimal as percentage, rounded to two decimal places)

comment

An optional comment explaining the output of the XPath query, which (if present) will be included as a note attached to the relevant column heading in the XLSX output file.