**1. Explanation of the Example Data and Model Files and their usage with JSONGrapher**

A zipfile is included with the article that has example data files and example model files.

All of the example files (JSON, CSV, TSV) can be used with www.JSONGrapher.com by dragging and dropping. Files must be uploaded/dragged into JSONgrapher one file at a time. By uploading/dragging data and models which are of the same type, they will be plotted together. For example, multiple CO2 adsorption isotherms can be plotted together (whether from some data or from a model). Data which was collected with different units will have automatic unit conversion in order to plot the data together.

Different data types (such as adsorption isotherms and infrared spectra) will not be plotted together, and instead an error message will be displayed.

Importantly, some of the files for CO2 adsorption isotherms are actually model files, and not data files. For example, ToBeMadeByAshi.json is a model file, which JSONGrapher then evaluates to create the plot.

Currently, JSONGrapher supports three formats: ".JSON”, “.csv”, and “.tsv”. The file format will be recognized as long as the period and the letters are anywhere in the filename. The csv and tsv files are comma separated and tab separated, respectively. Tab separated files are advantageous to include because they allow the inclusion of commas in series names, and some molecule names have commas.

For all file formats, the units must be in the x label and the y label, within parentheses.

**2. Technical Considerations: File Formats and Schema**

There are two primary technical decisions to make when it comes to a single implementation of the experiential results. (a) Which computer language to use? (b) Which data format to use?

On the question of which computer language to use, a decision was made to use JavaScript for this example on the basis that this would allow the simplest ease of use: the user simply needs to have a modern browser. The infrastructure is thus independent of operating system and does not require any familiarity with command lines, compilation, etc. In the present version, an internet connection is also required. Although an internet connection is required in the current implementation, an advantage of the current implementation is that the software is open source and online-hosted such that any users can make improvements, and when these are accepted into the master branch they will be immediately reflected to all users. The JavaScript is presently intentionally written in such a way that the computing power is provided by the user’s computer (not by the server), though it is possible to use cloud computing for simulations in the future. A decision was then made to use the software plotly (plotly.com) as this enables versatile and interacting plotting of graphs with an open source framework.

On the question of which data format to use, there are several common structured formats that can be considered, and conversion can occur between file formats. JSON, YAML, CSV, HDF5. The key considerations for the data format are that it should have the ability to store metadata, hierarchical data, should have a robust schema framework, and ideally be human readable and human editable. HDF5 is not human-readable, but compatibility with HDF5 is desirable as HDF5 has been designed for managing extremely large and complex data collections. A schema is a set of ‘rules’ that specify a standard for a file. For example, a schema could specify that adsorption isotherm data must have units of pressure for the x axis, and units of mass or moles divided by mass or moles for the y axis. A thorough discussion of schema is beyond the scope of this work, but we note that for each data type provided to JSONGrapher, there must be an existing schema in the schema directory. A generic schema can be setting ScatterPlot or XY as the DataType.

YAML can store meta data, has a robust schema framework and is human readable, can in principle store hierarchical data, though is not commonly used to store data. <https://yaml.org/>

JSON can store meta data, has a robust schema framework (though the schema libraries are less robust than those for YAML), is human readable, is hierarchical, is commonly used for data, and is the most easily converted to HDF5, and the most easily accessible by JavaScript. Plotly is designed with this recognition, and a plotly JSON schema exists and is included in JSONGrapher. Technically, JSON data can be included within YAML files, but in practice the two are often treated as separate formats. <https://json-schema.org/>

CSV / TSV file formats can store meta data, but do not have a robust schema framework (though schema frameworks do exist), but are not well suited to hierarchical data storage.

From the above considerations, JSON was chosen as the preferred format.

However, for users who are creating a file “by hand” with this use of spreadsheet software, the CSV / TSV format is most accessible. Thus, CSV compatibility has been included, by mapping the fields onto those of the JSON schema. That is, the CSV is internally converted to JavaScript arrays that are equal to a JSON, and then treated as a JSON. However, because of the limitations of CSV files, not all of the fields are editable through CSV (for example, the CSV file does not allow changing the plot layout). It is possible to add more complexity to the CSV format support, but not as facile, since it would require more complex parsing of the CSV file. The simplest way to support additional fields in the CSV will be to allow one line to be a string of unlimited length that can include JSON.

The Schema are also created in a hierarchical way, which we explain here. The way hierarchical schema are treated in YAML and JSON are different. YAML allows flexible ‘importing’ of fields from ‘parent’ Schema. With JSON Schema, the concept of a ‘parent’ schema does not exist: the analogous feature is to use the $ref keyword in such a way that requires the record to conform to *both* schema completely. In order to circumvent the lack of parent schema in JSON Schema, and to maintain facile compatibility with the CSV method of creating records, the current solution is to make child schema include all fields from the parent schema and to give the child schema filenames that include the parent schema after a double-underscore separator. For example, CO2\_\_adsorption\_isotherm is a subset of the data type adsorption\_isotherm and DRIFTS\_\_IR\_\_vibrational\_spectrum is a subset of IR\_\_vibrational\_spectrum and also a subset of vibrational\_spectrum.

**3. Usability Considerations**

It is important for software to be easily usable, particularly if an experience economy is the goal. Accordingly, we included several considerations (but not exhaustive) to avoid the unpleasant experience of a user getting “stuck” when attempting to use JSONGrapher.

* For any file added, the software checks that the file (or data after conversion) is valid json.
* If two data sets of incompatible types are attempted to be plotted together, the software produces an error message notifying the user.
* If a data set is missing required fields (such as units) the software will notify the user
* We provide a way for users to download the most recent data as JSON. Downloading as CSV is on the roadmap as a desired feature.

**References (for paper, not just for supporting information).**