ChemSpectra: A web-based spectra editor for analytical data

*Yu-Chieh Huanga, Pierre Tremouilhaca, An Nguyena, Nicole Jung\* a,b, Stefan Bräse\* a,b*

Email: yu-chieh.huang@kit.edu; pierre.tremouilhac@kit.edu; an.nguyen@kit.edu, nicole.jung@kit.edu; stefan.braese@kit.edu

aInstitute of Biological and Chemical Systems - Functional Molecular Systems (IBCS-FMS), Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany; bInstitute of Organic Chemistry, Karlsruhe Institute of Technology, Fritz-Haber-Weg 6, 76131 Karlsruhe, Germany.

**Contents**

[1. Spectra Transformation 2](#_Toc45381977)

[1.1. Spectra Transformation – Decoding and parsing NMR & IR files 3](#_Toc45381978)

[1.2 Spectra Transformation - Decoding and parsing MS files 5](#_Toc45381979)

[1.3 Spectra Transformation - Composing JCAMP-DX format 7](#_Toc45381980)

[2. Communication Overview with Chemotion ELN 9](#_Toc45381981)

[3. Additional rules for NMR multiplicity check 9](#_Toc45381982)

[Here is the additional rules: 10](#_Toc45381983)

[4. Limitations for parsing files of type fid 12](#_Toc45381984)

# 1. Spectra Transformation

Chem-spectra-app can process several possible input formats: JCAMP-DX, mzML and RAW files. Table S1 lists the accepted formats, filename extensions and spectrum types. Several classes are built to take care of different file extensions and spectra types (see figure S1).

Table S1. Possible input formats for chem-spectra-app.

|  |  |  |
| --- | --- | --- |
| **File format** | **Extension** | **Applicable types** |
| JCAMP-DX | .jcamp, .jdx and , .dx | NMR, IR and MS |
| mzML | .mzml | MS |
| raw | .raw | MS |

The identification of the provided data type is achieved by the given file extension and attributes. If the file extension is .mzml or .raw, MS related python classes will handle the file parsing and output composition. A JCAMP-DX[[1]](#endnote-2) file may contain NMR, IR or MS data, therefore, the JcampBaseConverter class assigns the provided type from the metadata that is provided with the data file. In a next step, the file is submitted to the different classes and a unified JCAMP-DX output format with the extension “.jdx” is created.



Figure S1. Flow chart of spectra transformation.

####

## 1.1. Spectra Transformation – Decoding and parsing NMR & IR files

In chem-spectra-app, NMR and IR spectra are sourced from the JCAMP-DX format. The data is read with a modified parser from nmrglue[[2]](#endnote-3)[[3]](#footnote-2)a, a python library

To identify the spectrum type in the data file, the ‘DATATYPE’ field (JCAMP standard) is used. The value of 'NMR SPECTRUM', 'NMRSPECTRUM', 'INFRARED SPECTRUM', or 'MASS SPECTRUM', is used to assign the file to one of the supported file types. During the further processing, different features will be applied based on the extracted type,

The JCAMP-DX format may contain one or more sections which are named “BLOCKS”. For NMR spectra, these blocks may contain time-domain or a frequency-domain data points, a real part and an imaginary part, or a peak list. If there are real and imaginary parts in a target data table, only the real part is processed. The IR spectrum has a simpler configuration, and the IR JCAMP-DX file contains only one block and one data table giving the recorded signals.

After reading a file, parameters and data points are stored inside a python object, including auto-picked and manual-picked peaks. If there are no existing auto-picked peaks, the scipy library[[4]](#endnote-4) will identify the peaks automatically. Since the user might set a new reference which induces lateral movement, this will also reflect to auto-picked peaks. When new user-defined peaks are present in the parameters, they will replace the previously given manual-picked part. A complete flow chart is shown in the figure S2.



Figure S2. Flow chart of reading a JCAMP-DX file.

## 1.2 Spectra Transformation - Decoding and parsing MS files

MS spectra can be managed for three possible formats (JCAMP, mzMl and RAW). In order to unify the processing of the spectra, all files are converted to the JCAMP-DX format (see figure S3). Mzml and raw formats are dedicated to MS spectra. Mzml[[5]](#endnote-5) is an open-source format that is similar to the human readable XML tree structure. Since the RAW format is given as a binary data file which is not human readable, it has to be decoded before a processing with ChemSpecta is possible. For this purpose, msConvert in Proteowizard[[6]](#endnote-6) is employed to convert MS files from raw to mzML. To this aim, msConvert in a docker container is called by the MSConvert class in the chem-spectra-app to achieve this. Then, mzML files are converted to JCAMP-DX using pymzML[[7]](#endnote-7), an open-source python mass spectrometry file parser.

In general, MS spectra files contain multiple scans reflecting the whole measurement. The chem-spectra-app manages this challenge by giving all available scans of which the most suitable one has to be selected by the user via the UI. Based on the different possible requests, the JCAMP-DX file is sent back to the client side or stored in the Chemotion-ELN server. Meanwhile, if an image of the spectrum is needed, it will be generated by the python matplotlib library[[8]](#endnote-8).

If the input file is a JCAMP-DX file, the process is similar to that of NMR and IR spectra. The “DATATYPE” field is “MASS SPECTRUM” and it is parsed and compiled using JcampMSConverter and MSComposer classes respectively. However, if the input format is not one of raw, mzML or JCAMP-DX, the user needs to convert the file into one of them by another software.



Figure S3. Mzml and raw conversion to JCAMP-DX inside the MSConvert class.

####

## 1.3 Spectra Transformation - Composing JCAMP-DX format

When the decoding and parsing is finalized, the next step is to compose a new JCAMP-DX file based on the obtained python object. The application composes JCAMP-DX files following the standard JCAMP-DX specification.[[9]](#endnote-9) There are three inner blocks which consist of spectrum points, auto-picked peaks and manually-picked peaks. An outer link block wraps these three blocks. An example for such a composed NMR JCAMP-DX file is given in figure S4.

The new composed JCAMP file only contains the necessary information, and redundant spectrum data, like imaginary data points, are removed. It is possible for users to open the same spectrum multiple times for editing and reviewing, and data transmission is reduced with this compact file.

During the design of ChemSpectra, the reduction of server loading was one of our requirements. Initially, the extraction of files in the server and then sending raw data points to the client side was also considered. Other services, like NMRPro, adopt this pattern. However, this configuration requests additional resource in the server to process the files every time when a user opens the editor. On the contrary, sending a file to the frontend and decoding by jcampconverter in the client browser decreases the server loading.



Figure S4. An abstraction of a composed JCAMP-DX file.

In addition to the spectra in a machine readable format, an image format is necessary for reporting functions. Therefore, a png file is generated by the matplotlib library from the given data points. Both JCAMP-DX file and the image file are zipped together. They are either sent back to the chem-spectra-client upon download, or the information is unzipped and saved inside the Chemotion-ELN server.

# 2. Communication Overview with Chemotion ELN

Figure S6 illustrates the communications of the standalone service: upload, download, and edit files. All actions start with the user input. The system is built in a way that the Client and ELN blocks, which are the traffic regulator of the system, can be exchanged.



Figure S6. Communication overview for upload, download and edit.

# 3. Additional rules for NMR multiplicity check

Multiplicities will be automatically inferred by the library[[10]](#endnote-10),[[11]](#endnote-11) and compared by additional rules to ensure the correctness.

## Here is the additional rules:

When a user assigns several peaks forming a multiplicity, selected peaks will be inferred by the library to extract the multiplicity type first, and the peak count will be verified to avoid obvious mistakes. For example, a multiplicity is inferred as a doublet, but there are three peaks given which violates the definition of the doublet. Therefore, it will be assigned as a type ‘m’ multiplet, and the user should identify the multiplicity manually.

The verification of the peak count is achieved by comparing the inferred multiplicity type with the upper limit of the possible count. The count should be equal or less than the upper limit. The upper limit is calculated as the pseudocode, and here is one example. The type ‘tdd’ has two ‘d’ and one ‘t’, and the upper limit is:

 **1 \* (2** peaks for one ‘d’ **\* 2** ‘d’ in the string**) \* (3** peaks for one ‘t’ **\* 1** ‘t’ in the string**) = 12**.

If the peak count is equal to or smaller than 12, ‘tdd’ is reasonable, and it will be given to the user. Otherwise, the multiplicity will be overwritten as ‘m’.

The pseudocode 1

|  |
| --- |
| **function** matchStingAndReplace **is** **input** *string* sourceType *string* targetType **output** *string* resultType *int* count resultType ← sourceType count ← how many sbustring targetType in the sourceType **if** count > 0 **do** resultType ← remove all sbustring targetType from the sourceType **return** resultType, count**function** verifyPeakCount **is** **input** *string* inferredType *array* peaks **output** *bool* withInLimit /\* parse inferredType and update the upper limit \*/ limit ← 1 resultType, count ← matchStingAndReplace(inferredType, 'quint') limit ← limit \* (5 \*\* count) resultType, count ← matchStingAndReplace(resultType, 'sept') limit ← limit \* (7 \*\* count) resultType, count ← matchStingAndReplace(resultType, 'd') limit ← limit \* (2 \*\* count) resultType, count ← matchStingAndReplace(resultType, 't') limit ← limit \* (3 \*\* count) resultType, count ← matchStingAndReplace(resultType, 'q') limit ← limit \* (4 \*\* count) resultType, count ← matchStingAndReplace(resultType, 'h') limit ← limit \* (6 \*\* count) resultType, count ← matchStingAndReplace(resultType, 'o') limit ← limit \* (8 \*\* count) resultType, count ← matchStingAndReplace(resultType, 'n') limit ← limit \* (9 \*\* count) withInLimit ← false **if** peaks.length <= limit **do** withInLimit ← true **return** withInLimit |

# 4. Limitations for parsing files of type fid

ChemSpectra can parse fid files that are generated by 1D-NMR spectroscopic measurements (tested for several Bruker NMR instruments). The fid-files can also be parsed and converted by nmrglue, if the fid is part of a zip folder that is uploaded. Some limitations were found with respect to the quality of processing of fid files. Figure 7 shows the ChemSpectra processing of a 13C NMR data file which was provided as jcamp file. In this case, the Fourier transformation of the originally obtained fid file was done by Bruker software and the resulting JCAMP was uploaded to ChemSpectra. In contrast, Figure 8 shows the result of the fid conversion done by ChemSpectra based on a provided zip folder including the originally gained fid file. The lower quality of the processing, which needs further corrections by phase correction, is a result of nmrglue with automatically phase correction which was used to generate the necessary jcamp data.



Figure S7. Data obtained after upload of a JCAMP file to ChemSpectra for exemplary used 13C NMR data. Fourier transformation and automatic phase corrected was done before and is already included to the provided JAMP file.



Figure S8. Data obtained after upload of a fid file (included to a zip file) to ChemSpectra. The Fourier transformation is done by nmrglue.

1. <http://www.jcamp-dx.org/> [↑](#endnote-ref-2)
2. Helmus JJ, Jaroniec CP (2013) Nmrglue: An open source Python package for the analysis of multidimensional NMR data, J. Biomol. NMR 55:355-367. [↑](#endnote-ref-3)
3. a The original nmrglue only outputs the first data table. The modified version enables the output of all data table. <https://github.com/ComPlat/nmrglue/commits/show-all-data> [↑](#footnote-ref-2)
4. Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, van der Walt SJ, Brett M, Wilson J, Millman KJ, Mayorov N, Nelson ARJ, Jones E, Kern R, Larson E, Carey CJ, Polat I, Feng Y, Moore EW, VanderPlas J, Laxalde D, Perktold J, Cimrman R, Henriksen I, Quintero EA, Harris CR, Archibald AM, Ribeiro AH, Pedregosa F, van Mulbregt P, SciPy 1.0 Contributors. (2019) **SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python**. preprint [arXiv:1907.10121](https://arxiv.org/abs/1907.10121) [↑](#endnote-ref-4)
5. Deutsch EW (2010) Mass Spectrometer Output File Format mzML. Methods Mol Biol. 2010; 604:319–331. [↑](#endnote-ref-5)
6. Chambers MC, MacLean B, Burke R, Amode D, Ruderman DL, Neumann S, Gatto L, Fischer B, Pratt B, Egertson J, Hoff K, Kessner D, Tasman N, Shulman N, Frewen B, Baker TA, Brusniak MY, Paulse C, Creasy D, Flashner L, Kani K, Moulding C, Seymour SL, Nuwaysir LM, Lefebvre B, Kuhlmann F, Roark J, Rainer P, Detlev S, Hemenway T, Huhmer A, Langridge J, Connolly B, Chadick T, Holly K, Eckels J, Deutsch EW, Moritz RL, Katz JE, Agus DB, MacCoss M, Tabb DL, Mallick P (2012) A cross-platform toolkit for mass spectrometry and proteomics. Nature Biotechnology 30:918-920. [↑](#endnote-ref-6)
7. Kösters M, Leufken J, Schulze S, Sugimoto K, Klein J, Zahedi RP, Hippler M, Leidel SA, Fufezan C (2018) pymzML v2.0: introducing a highly compressed and seekable gzip format, Bioinformatics 34:2513–2514. [↑](#endnote-ref-7)
8. <https://matplotlib.org/> [↑](#endnote-ref-8)
9. <http://www.jcamp-dx.org/protocols.html>

<http://publications.iupac.org/pac/2001/pdf/7311x1765.pdf> [↑](#endnote-ref-9)
10. https://github.com/cheminfo-js/spectra [↑](#endnote-ref-10)
11. Cobas JC, Constantino-Castillo V, Martín-Pastor M, del Río-Portilla F. A two-stage approach to automatic determination of 1H NMR coupling constants. Magn Reson Chem. 2005;43(10):843‐848. doi:10.1002/mrc.1623 [↑](#endnote-ref-11)