CBOR 4 Proteomics

Olivier Langella
2026-11-23

Contents

1. Introduction 2
   1.1. CBOR 2
   1.2. Why using CBOR file format ? 2
2. mzCBOR: storing peak lists 3
3. mzCBOR documentation 4
   3.1. Introduction 4
   3.2. mzCBOR relation with mzML 4
   3.3. Storage efficiency 6
   3.4. CPU efficiency 6
   3.5. mzCBOR index file 6
4. mzML conversion to and from mzCBOR 8
   4.1. mzML to mzCBOR Conversion tool 8
   4.2. mzCBOR to mzML Conversion tool 9
5. Precursor finder 11
   5.1. Precursor finder 11
6. PeptidOMS 13
   6.1. PeptidOMS 13
   6.2. Source code 13
   6.3. PSM CBOR file workflow 13
   6.4. Command line interface 13
   6.5. JSON parameter file 13
   6.6. CBOR peptidoms eval PSM structure 17
   Bibliography 17
7. psmCBOR identification results 18
8. PSM CBOR format documentation 19
   8.1. Introduction 19
   8.2. psm CBOR format 19
   8.3. Root structure 21
   8.4. Process entries 22
   8.5. Protein description 22
   8.6. Sample description 22
   8.7. Scan description 23
   8.8. PSM description 25
9. Reading Sage results 26
   9.1. Convert Sage results to psmCBOR 26
10. Reading mzIdentML files 27
    10.1. Convert mzIdentML results to psmCBOR 27
11. psmcborfilter 28
12. psmcborproteinfilter 30
    12.1. psmcborproteinfilter 30
    12.2. JSON parameter file 30
    12.3. Filtering sage results PSMs and proteins at 1% FDR 30
13. psmcborfdr 32
14. mcqCBOR MassChroQ quantification 34
15. My Page 35

Chapter 1

Introduction

> **Web Ready!** **Info 1**
>
> The documentation is also available as HTML.

The PAPPSO CLI tools aims to provide tiny separated tools to manipulate huge proteomics data from peak lists, through PSMs, and quantification data.



Each of theses tools are compatible between them and can be used in UNIX pipes to save storage space during processing or allow remote data access and process it as streams.

It relies on data structures stored in CBOR format.

1.1. CBOR

The Concise Binary Object Representation (CBOR) is a binary data format designed by the Internet Engineering Task Force (IETF, RFC 8949). Its aim is to increase processing and transfer speeds, albeit at the cost of human data readability.

The use of this format as a replacement for mzML is intended not only to save storage space but also because of speed benefits during parsing and processing.

1.2. Why using CBOR file format ?

The CBOR file format is relatively recent (december 2020). It has several objectives, very well explained in this note : https://www.rfc-editor.org/rfc/rfc8949.html.

Among these objectives, the most important points are:

- Simple data structure, similar to JSON.
- So simple that CBOR libraries are available for various programming languages, featuring very small binary overhead (well below XML for sample)
- Serialised data
- Data compactness
- Frugal in CPU usage for both encoding and decoding.
- Suitable for high volume of data.
- Support all JSON data types for conversion to and from JSON.
- The format is designed for decades of use and extensible.

This makes it a format of choice for Mass Spectrometry data that must be stored reliably huge amount of data for decades while being reasonably compact.

Chapter 2

mzCBOR: storing peak lists



Chapter 3

mzCBOR documentation

3.1. Introduction

the CBOR-based format (mzCBOR) format replicates the structure of the mzML-formatted file into a CBOR binary file.

The Concise Binary Object Representation (CBOR) is a binary data format designed by the Internet Engineering Task Force (IETF, RFC 8949). Its aim is to increase processing and transfer speeds, albeit at the cost of human data readability.

The use of this format as a replacement for mzML is intended not only to save storage space but also because of speed benefits during parsing and processing.

CBOR key features Why using CBOR file format ?

3.2. mzCBOR relation with mzML

mzCBOR is an exact binary representation of an mzML data file. Each XML element and attributes is simply translated to data trees and arrays.

For example :

```
1 <cvList count="2">
2     <cv id="MS" fullName="Mass spectrometry ontology" version="4.1.38"
  URI="https://raw.githubusercontent.com/HUPO-PSI/psi-ms-CV/master/psi-ms.obo" />
3     <cv id="UO" fullName="Unit Ontology" version="09:04:2014" URI="https://raw.gi
  thubusercontent.com/bio-ontology-research-group/unit-ontology/master/unit.obo" />
4 </cvList>
```

is translated to :

```
1 "cvList": {
2     "count": 2,
3     "cv": [
4         {id: "MS", "fullName": "Mass spectrometry ontology", "version": "4.1.38",
  "URI": "https://raw.githubusercontent.com/HUPO-PSI/psi-ms-CV/master/psi-ms.obo"},
5         {id: "UO", "fullName": "Unit Ontology", "version": "09:04:2014", "URI":
  "https://raw.githubusercontent.com/bio-ontology-research-group/unit-ontology/
  master/unit.obo"}
6     ]
7 }
```

- Repeated elements are stored in arrays (this is the case in this example for “cv”)
- String values representing integers are stored as 64 bits integers
- String values representing floats are stored as 64 bits floats

3.2.1. Binary data array encoding

Unlike any other mzML elements, the “binaryDataArrayList” element has a specific translation.

```
<binaryDataArrayList count="2">
    <binaryDataArray encodedLength="6380">
        <cvParam cvRef="MS" accession="MS:1000514" value="" name="m/z array"
unitAccession="MS:1000040" unitName="m/z" unitCvRef="MS" />
        <cvParam cvRef="MS" accession="MS:1000523" value="" name="64-bit
float" />
        <cvParam cvRef="MS" accession="MS:1000574" value="" name="zlib
compression" />
        <binary>eJwt03tcj3f/wPGr...</binary>
    </binaryDataArray>
    <binaryDataArray encodedLength="9092">
        <cvParam cvRef="MS" accession="MS:1000515" value="" name="intensity
array" unitAccession="MS:1000131" unitName="number of counts" unitCvRef="MS" />
        <cvParam cvRef="MS" accession="MS:1000523" value="" name="64-bit
float" />
        <cvParam cvRef="MS" accession="MS:1000574" value="" name="zlib
compression" />
        <binary>eJwtWnlcCGsbnZKK...</binary>
    </binaryDataArray>
</binaryDataArrayList>
```

It is converted to an array :

```
"binaryDataArray": [
    {
        "bits": 64,
        "isInt": false,
        "compress": "zlib",
        "unit": "MS:1000514",
        "byteArray": 010101010101...,
    },
    {
        "bits": 64,
        "isInt": false,
        "compress": "zlib",
        "unit": "MS:1000515",
        "byteArray": 010101010101...,
    }
]
```

The base64 encoded mzML “binary” element is decoded into a byteArray (compressed if mzML was also containing compressed data).

Indeed, storing binary data as a base64 string is very expensive in storage space compared to the same data in binary string.

Any compression method used in mzML is allowed in mzCBOR.

3.2.2. mzCBOR header

The first element of the mzCBOR file is an “mzCBOR” element containing informations on the conversion software and universal identifier. The second element is the original mzML element with attributes from the original mzML file.

```
{
    "mzCBOR": {
```

```
        "mode": 0,
        "informations": {
            "software": "libpappsomspp",
            "version": "0.11.6",
            "type": "mzCBOR",
            "operation": "mzMLconvert",
            "cpu_used": 40,
            "pappsomspp_version": "0.11.6",
            "sysinfo_machine_hostname": "milano",
            "sysinfo_product_name": "Debian GNU/Linux 13 (trixie)",
            "timestamp": "2025-12-02T14:07:13",
            "uuid": "{2292d028-fbb7-485a-a82a-f5a03abc0267}"
        }
    },
    "mzML": {
        "xmlns": "http://psi.hupo.org/ms/mzml",
        "schemaLocation": "http://psi.hupo.org/ms/mzml http://psidev.info/files/
   ms/mzML/xsd/mzML1.1.0.xsd",
        "version": "1.1.0",
        "id": "20120906_raw_extract_1_A01_urnb-1"
    },
    "cvList":..
}
```

3.3. Storage efficiency

mzCBOR file size is 66% the mzML one.

3.4. CPU efficiency

	mzML	mzCBOR
TIC chromatogram	25.8692 s	18.3258 s
Retention timeline	8.02264 s	7.16747 s
Random access	24.6577 ms	786.731 µs

3.5. mzCBOR index file

Beside the mzCBOR data file, an index file companion can be generated by the conversion tool.

```
mzml2mzcbor --index -i /path/to_the_mzml/sources/ -o /path/to_the_mzcbor/
  destination/
```

This index file is also a CBOR file and contains positions offset of spectrum in the mzCBOR file.

3.5.1. mzCBOR index file structure

```
  "informations": {
    "software": "libpappsomspp",
    "version": "0.11.21",
    "type": "mzCBORindex",
    "operation": "mzMLconvert",
    "cpu_used": 32,
```

```
    "pappsomspp_version": "0.11.21",
    "sysinfo_machine_hostname": "proteus1",
    "sysinfo_product_name": "Debian GNU/Linux 13 (trixie)",
    "timestamp": "2026-06-15T15:52:08",
    "uuid": "{afbaeeb4-fa68-4cdf-b557-9bc17095b85e}"
  },
  "from": "{5e57e3ea-5380-4ce9-95c2-842e81e9d6e7}",
  "runIdList": ["20120906_balliau_extract_1_A01_urnb-1"],
  "runSpectrumIndexList": [
    [
      2874, 16271, 25262, 34729, 44519, 53323, 62664, 72247, 81814, 91817,
      ...
    ]
  ],
  "runSpectrumNativeIdList": [
    [
      "controllerType=0 controllerNumber=1 scan=1",
      "controllerType=0 controllerNumber=1 scan=2",
      ...
    ]
  ],
  "runSpectrumTotalIonCountList": [
    [
      17377980, 44878432, 45104248, 44734028, 42494200, 44990160, 44421312,
      ...
    ]
  ],
  "runSpectrumRtList": [
    [
      0.292552998, 0.71294202, 1.0739429999999999, 1.4343259800000001,
      ...
    ]
  ],
  "runSpectrumMsLevelList": [
    [
      1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
      ...
    ]
  ]
}
```

Chapter 4

mzML conversion to and from mzCBOR

4.1. mzML to mzCBOR Conversion tool

4.1.1. Conversion of a single mzML file to mzCBOR

mzml2mzcbor [OPTIONS]

The mzml2mzcbor command line interface program performs the conversion of files from the mzML format to a CBOR-based format (mzCBOR) that replicates the structure of the mzML-formatted file.

The Concise Binary Object Representation (CBOR) is a binary data format designed by the Internet Engineering Task Force (IETF, RFC 8949). Its aim is to increase processing and transfer speeds, albeit at the cost of human data readability.

The use of this format as a replacement for mzML is intended not only to save storage space but also because of speed benefits during parsing and processing.

Options

-i, --input	mzML input file
-o, --output	mzCBOR output file
-h, --help	Print the help message
-v, --version	Print the version of the sotfware

License GPL-3.0-or-later
Contribute https://forge.inrae.fr/pappso/pappsomspp

Debian package

Install PAPPSO source list

```
1 sudo apt update
2 sudo apt install libpappsomspp-cli
```

```
1 mzml2mzcbor -i LFQ_Astral_DDA_15min_50ng_Condition_A_REP1.mzML -o
  LFQ_Astral_DDA_15min_50ng_Condition_A_REP1.mzcbor
```

> In this example, the original file size of "LFQ_Astral_DDA_15min_50ng_Condition_A_REP1.mzML" is 1,9G. The "LFQ_Astral_DDA_15min_50ng_Condition_A_REP1.mzcbor" file size is 1,3G, for exactly the same amount of data.

4.1.2. Conversion of a whole directory

```
1  mzml2mzcbor -i /path/to_the_mzml/sources/ -o /path/to_the_mzcbor/destination/
```

> Do not forget the trailing "/"

In this example, all files with extension .mzML present in the source directory are converted to the same filename in the destination directory minus the extension .mzML replaced by .mzcbor

4.2. mzCBOR to mzML Conversion tool

4.2.1. Conversion of a single mzCBOR file to mzML

mzcbor2mzml [OPTIONS]

The mzcbor2mzml command line interface program performs the conversion of files from the CBOR-based format (mzCBOR) format that replicates the structure of the mzML-formatted file to the mzML standard.

The Concise Binary Object Representation (CBOR) is a binary data format designed by the Internet Engineering Task Force (IETF, RFC 8949). Its aim is to increase processing and transfer speeds, albeit at the cost of human data readability.

The use of this format as a replacement for mzML is intended not only to save storage space but also because of speed benefits during parsing and processing.

Options

-i, --input	mzCBOR input file
-o, --output	mzML output file
-h, --help	Print the help message
-v, --version	Print the version of the sotfware

License GPL-3.0-or-later
Contribute https://forge.inrae.fr/pappso/pappsomspp

Debian package

Install PAPPSO source list

```
1  sudo apt update
2  sudo apt install libpappsomspp-cli
```

```
1  mzcbor2mzml -i LFQ_Astral_DDA_15min_50ng_Condition_A_REP1.mzcbor -o
   LFQ_Astral_DDA_15min_50ng_Condition_A_REP1.mzML
```

The resulting mzML file is the exact image of the original mzML file with automatic XML indentation and real UTF-8 encoding (which is not particularly well guaranted in XML bioinformatics output).

4.2.2. Conversion of a whole directory

```
1  mzcbor2mzml -i /path/to_the_mzcbor/sources/ -o /path/to_the_mzml/destination/
```

> Do not forget the trailing “/”

In this example, all files with extension .mzcbor present in the source directory are converted to the same filename in the destination directory minus the extension .mzcbor replaced by .mzML

Chapter 5

Precursor finder

5.1. Precursor finder

mzCBOR data analysis : for each MS2 fragmentation spectrum, the MS1 isolation window is scrutinized to find or fix possible precursor mass and charge.

This helps identification engines: cofragmentations can be better identified, as well as wrong assignation caused by false isotope annotation.

precursorfinder [OPTIONS]

The precursorfinder command line interface program reads every MSn scans to find for each possible precursor in the MS1 isolation window a potential better precursor m/z and charge state.

Options

-i, --input	mzcbor input file
-o, --output	mzcbor output file
-p, --parameters	JSON parameter file

License GPL-3.0-or-later
Contribute https://forge.inrae.fr/pappso/pappsomspp

Debian package

Install PAPPSO source list

```
sudo apt update
sudo apt install libpappsomspp-cli
```

JSON default parameters :

```
{
  "ms1_tolerance_unit": "ppm",
  "ms1_tolerance": 10.0,
  "ms2_tolerance_unit": "dalton",
  "ms2_tolerance": 0.02,
  "minimum_complement_ion": 1,
  "intensity_ratio_threshold": 0.3,
  "ms2_max_peaks": 60,
  "potential_charge_states": [1, 2, 3]
}

```

`ms1_tolerance_unit`

string *Required* MS1 tolerance unit can be “dalton” or “ppm”

5.1.1. Use precursor finder on mzML file (example)




```
1  mzml2mzcbor -i input.mzML | precursorfinder - | mzcbor2mzml -o output.mzML
```

Chapter 6

PeptidOMS

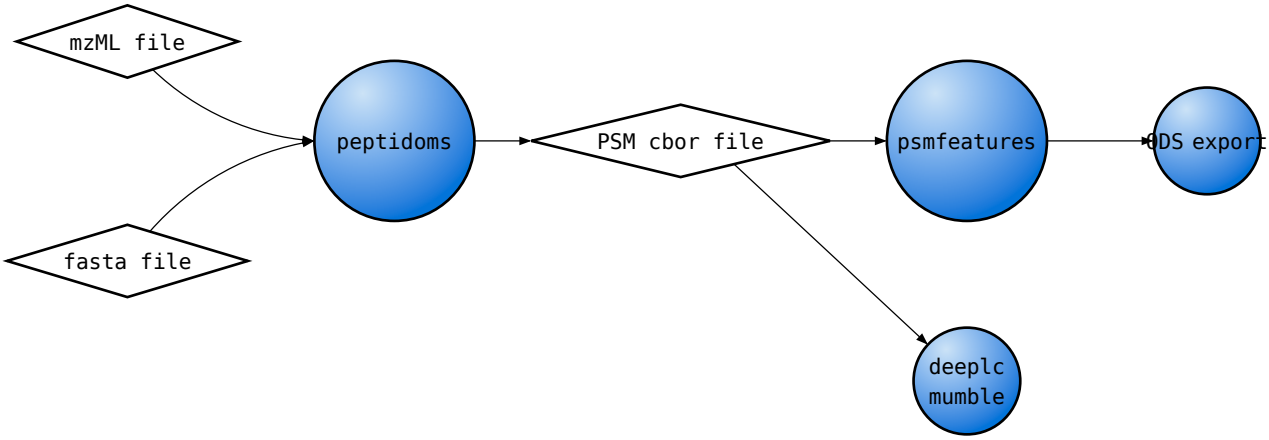
6.1. PeptidOMS

Spoms aligns MS2 fragmentation spectra directly to protein sequences. Algorithm is described in (Benoist et al., 2025).

6.2. Source code

Current source code is available at https://forge.inrae.fr/peptidoms/specpeptidoms.

6.3. PSM CBOR file workflow



6.4. Command line interface

Spoms works on a collection of MS/MS spectra (mzML file typically) and a protein sequence database (FASTA file) and produces a PSM CBOR file as output.

To get a readable file results, use “spoms-export” to produce JSON or ODS files:

6.5. JSON parameter file

Spoms parameters are specified in a JSON file

```
{
  "aa_table": {
    "C": { "modifications": ["MOD:00397"] }
  },
  "precursor": {
    "precursor_process": false,
    "ms1_tolerance_unit": "ppm",
    "ms1_tolerance": 10,
    "minimum_complement_ion": 1,
    "intensity_ratio_threshold": 0.3,
    "potential_charge_states": [1, 2, 3]
  },
  "spectrum": {
    "fragment_tolerance_unit": "dalton",
    "fragment_tolerance": 0.02,
    "deisotope": true,
    "minimum_mz": 150,
```

```
    "n_most_intense": 60
  },
  "pre_treatment": {
    "use_neutral_losses": false,
    "neutral_losses": {
      "NH3": 17.026549101523,
      "H2O": 18.010564684042,
      "NH3+NH3": 34.053098203046,
      "NH3+H2O": 35.037113785565,
      "H2O+H2O": 36.021129368084
    },
    "neutral_losses_scores": {
      "NH3": 3,
      "H2O": 2,
      "NH3+NH3": 1,
      "NH3+H2O": 1,
      "H2O+H2O": 1
    }
  },
  "alignment": {
    "nb_peaks_min": 30,
    "nb_min_aa_found": 20,
    "min_scenario_score": 10,
    "locations_score_threshold": 0.6,
    "nb_locations_saved": 25,
    "score": {
      "shut_down_3_peaks_version": false,
      "certainly_found_with_shift_main": -6,
      "found_main": 7,
      "found_with_shift_main": -8,
      "not_found_main": -4,
      "certainly_found_main": 10,
      "max_realignment_size_first_column": 4,
      "max_realignment_size": 4,
      "surplus": 5,
      "undefined_nter_penalty": 0,
      "undefined_cter_penalty": 0,
      "intensity_bonus_thresh": 0.8,
      "intensity_bonus": 0,
      "intensity_penalty_thresh": 0.3,
      "intensity_penalty": 0
    }
  },
  "post_treatment": {
    "maximum_tries_on_recursive_correction": 5000,
    "shut_down_peaks_cleaning": false,
    "shut_down_non_aligned_mass": false,
    "nb_interpretations_saved": 5,
    "score": {
      "certainly_found_post": 10,
      "found_post": 7,
      "certainly_found_with_shift_post": -6,
      "found_with_shift_post": -8,
      "not_found_post": -4,
      "max_realignment_size": 4
    }
  },
  "output": {
    "max_interpretations_per_spectrum": 5,
    "minimum_peptide_length": 3
  }
}

```

6.5.1. `aa_table` section *optional*

This section allows to define your own amino acid table.

By default, spoms uses this table :

```
{
  "C": {"modifications": ["MOD:00397"]}
}
```

for the *Carbamidomethyl* fixed modification on Cystein.

This term is defined in the PSI-MOD ontology as:

```
[Term]
id: MOD:00397
name: iodoacetamide derivatized residue
def: "A protein modification that is produced by reaction with iodoacetamide, usually
  replacement of a reactive hydrogen with a methylcarboxamido group." [PubMed:11327326,
  PubMed:11510821, PubMed:12422359, Unimod:4]
subset: PSI-MOD-slim
synonym: "Carbamidomethyl" RELATED PSI-MS-label []
synonym: "Iodoacetamide derivative" RELATED Unimod-description []
```

It is possible to override this amino acid modification table by providing you own definition.

13C residue modifications can be found in the PSI-MOD ontology with accession MOD:00842:

```
is_a: MOD:00842 ! (13)C labeled residue
```

15N residue modifications can be found in the PSI-MOD ontology with accession MOD:00843:

```
is_a: MOD:00843 ! (15)N labeled residue
```

For a complete 13C + 15N list of labelled residue, you can use :

```
{
  "A": {"modifications": ["UNIMOD:1297"]},
  "R": {"modifications": ["MOD:00587"]},
  "N": {"modifications": ["UNIMOD:1266", "UNIMOD:995"]},
  "D": {"modifications": ["UNIMOD:1298"]},
  "C": {"modifications": ["UNIMOD:1297", "MOD:00397"]},
  "E": {"modifications": ["UNIMOD:268"]},
  "Q": {"modifications": ["UNIMOD:772", "UNIMOD:995"]},
  "G": {"modifications": ["UNIMOD:2088"]},
  "H": {"modifications": ["UNIMOD:188", "UNIMOD:996"]},
  "I": {"modifications": ["MOD:01286"]},
  "L": {"modifications": ["MOD:01285"]},
  "K": {"modifications": ["MOD:00582"]},
  "M": {"modifications": ["UNIMOD:772", "UNIMOD:994"]},
  "F": {"modifications": ["MOD:00589"]},
  "P": {"modifications": ["UNIMOD:772", "UNIMOD:994"]},
  "S": {"modifications": ["UNIMOD:1296", "UNIMOD:994"]},
  "T": {"modifications": ["UNIMOD:1298"]},
  "W": {"modifications": ["UNIMOD:188", "UNIMOD:772", "UNIMOD:995"]},
  "Y": {"modifications": ["UNIMOD:269"]},
  "V": {"modifications": ["MOD:00588"]},
}
```

6.5.2. `precursor` section

`precursor_process`

boolean *Required* enable MS1 signal process to find alternative precursors. false by default, only works with mzML or mzCBOR peak list files

`ms1_tolerance_unit` **string** MS1 tolerance unit to use ("dalton" or "ppm", ppm by default)

`ms1_tolerance` **float** MS1 tolerance to look for peaks in MS1 signal (+/- this value)

`minimum_complement_ion` **integer** given a possible MS1 precursor mass : check the number of complement ion in the MS2 spectra.

`intensity_ratio_threshold` **float** Filter the MS1 signal for peaks of a minimum intensity compared to the official precursor intensity.

`potential_charge_states` **integer array** Look for potential charge states in the precursor isolation window (default is : [1,2,3]).

6.5.3. `spectrum` section

`fragment_tolerance_unit` **string** "dalton" by default
`fragment_tolerance` **float** mass delta precision (0.02 dalton by default)
`deisotope` **boolean** if true, process spectrum to merge isotopes into single charge peaks
`minimum_mz` **float** minimum m/z ratio to consider peaks in the spectrum
`n_most_intense` **int** consider only the N most intense peaks in the spectrum

6.5.4. `pretreatment` section

`use_neutral_losses` **boolean** if true, uses neutral losses peaks to score alignments
`neutral_losses` **dict** list of neutral losses to use and associated masses
`neutral_losses_scores` **dict** bonus scores associated to each neutral loss

6.5.5. `alignement` section

`nb_peaks_min` **int** Only spectra above that number of peaks are processed
`nb_min_aa_found` **int** Only spectra above that number of labels are processed
`min_scenario_score` **int** Only interpretations with a score over this threshold are returned to the user
`locations_score_threshold` **float** Excludes LssSM with a score below the best score multiplied by this factor when several are returned per spectrum by the preliminary alignment
`nb_locations_saved` **int** Number of Located subsequence Spectrum Matches (LssSM) per spectrum saved in memory at a time
`nb_returned_alignments` **int** Number of spectra alignments returned to the user
`score/undefined_nter_penalty` **int** if an unidentified mass remains in nter, apply this penalty to the alignment
`score/undefined_cter_penalty` **int** if an unidentified mass remains in cter, apply this penalty to the alignment
`score/intensity_bonus_thresh` **float** between 0 and 1. Above this proportion of the maximum peak intensity of the spectra, the peaks receive a bonus defined by `intensity_bonus`
`score/intensity_bonus` **int** bonus applied in the case described above
`score/intensity_penalty_thresh` **float** between 0 and 1. Below this proportion of the maximum peak intensity of the spectra, the peaks receive a penalty defined by `intensity_penalty`
`score/intensity_penalty` **int** penalty applied in the case described above

6.5.6. `post_treatment` section

`shut_down_peaks_cleaning` **boolean** Option (true/false) to correct specific alignments in the second alignment round
`shut_down_non_aligned_mass` **boolean**

`nb_interpretations_saved` **int** Number of backup interpretations per spectrum saved in memory at a time

6.5.7. `output` section

`max_interpretations_per_spectrum` **int** Number of interpretations per spectrum returned to the user

`minimum_peptide_length` **int** minimum number of amino acids for a PSM to be displayed (default is 3)

6.5.8. `alignment` section

6.5.9. `post_treatment` section

6.6. CBOR peptidoms eval PSM structure

Spoms writes results in a PSM CBOR file. For each PSM, Spoms uses the "eval" section to store its scores :

```
{
  "proforma": "AIADGSLLDLLR",
  "protein_list": [
    {
      "accession": "GRMZM2G083841_P01",
      "position": [429]
    }
  ],
  "eval": {
    "peptidoms": {
      "bracket": "[A][I]ADGSLLDLLR",
      "spc": 12,
      "score": 76,
      "nam": 0
    }
  }
}

```

Bibliography

Benoist, É., Jean, G., Rogniaux, H., Fertin, G., and Tessier, D. (2025). SpecPeptidOMS Directly and Rapidly Aligns Mass Spectra on Whole Proteomes and Identifies Peptides That Are Not Necessarily Tryptic: Implications for Peptidomics. *Journal of Proteome Research* 24, 2159–2172. doi: 10.1021/acs.jproteome.4c00870

Chapter 7

psmCBOR identification results



Chapter 8

PSM CBOR format documentation

8.1. Introduction

PSM CBOR file is designed to record any PSM (peptide spectrum match) in a compact binary file, easy to manipulate, versatile, extendable. This file is used as a stream in any condition, allowing the users to use unix pipes, compression algorithms, network transparency.

This way, from a DDA identification engine search result converted in PSM CBOR, any process can be added :

- Feature computations
- Prediction process (retention times, ion mobility, MS2 prediction...)
- Rescoring
- Filtering

8.2. psm CBOR format

It could be something like that :


```
{
  "informations": {
    "executable": "mcql",
    "type": "psm",
    "pappsomspp_version": "0.9.46",
    "sysinfo_machine_hostname": "proteus1",
    "sysinfo_product_name": "Debian GNU/Linux 12 (bookworm)",
    "timestamp": "2025-06-25T10:33:51"
  },
  "log": [],
  "parameter_map": {
    "sage": {},
    "xtandem": {},
    "specoms": {}
  },
  "target_fasta_files": ["zea_mays.fasta", "contaminant.fasta"],
  "decoy_fasta_files": ["rev_zea_mays.fasta", "rev_contaminant.fasta"],
  "protein_map": {
    "GRMZM2G083841_P01": {
      "description": "",
      "sequence": "",
      "target": true,
      "contaminant": false,
      "props": {},
      "eval": {}
    },
    "GRMZM2G083841_P02": {}
  },
  "sample_list": [
```

```
    {
      "name": "tandem2017_nopatch_20120906_balliau_extract_1_A01_urnb-1",
      "identification_file_list": [
        {
          "name": "/home/langella/data1/tandem/
tandem2017_nopatch_20120906_balliau_extract_1_A01_urnb-1.xml"
        }
      ],
      "peaklist_file": {
        "name": "tandem2017_nopatch_20120906_balliau_extract_1_A01_urnb-1.mzml"
      },
      "scan_list": [
        {
          "id": {
            "index": 1976
          },
          "precursor": {
            "z": 2,
            "mz": 1120.529471,
            "intensity": 500000,
            "alternatives": [
              {
                "z": 2,
                "mz": 1120.529471,
                "intensity": 500000
              }
            ]
          },
          "ms2": {
            "rt": 12648.87,
            "spectrum": {
              "mz": [1, 2, 3, 4],
              "intensity": [1, 2, 3, 4]
            }
          },
          "props": {},
          "psm_list": [
            {
              "proforma": "AQEEM[+15.99491]AQVAK",
              "protein_list": [
                {
                  "accession": "GRMZM2G083841_P01",
                  "positions": [15, 236]
                }
              ],
              "props": {
                "ion-series": {}
              },
              "eval": {
                "xtandem": {
                  "evalue": 0.0011
                },
                "specoms": {
                  "evalue": 0.0011
                }
              }
            }
```

```
86            ]
87          }
88        ]
89      }
90    ]
91  }
92
```

8.3. Root structure

root sections are required and the order must be respected.

`informations`

object *Required*

`log`

array *Required*

contains an array that logs all the previous “informations” sections. It helps to keep trace of PSM treatments.

`parameter_map`

dict *Required*

dictionary where each entry corresponds to a specific process (see Process entries [8.4.]). Each entry must contain the parameters used for this process.

`target_fasta_files`

array *Required*

List of file path to FASTA files (preferably absolute file path) used as reference target protein sequences for the identification engine.

`decoy_fasta_files`

array *Optional*

List of file path to FASTA files (preferably absolute file path) used as decoy protein sequences for the identification engine (if any and not generated on the fly).

`protein_map`

dict *Required*

contains an entry for each protein, using the accession as a unique identifier.

`sample_list`

array *Required*

an array of “sample” objects (see sample [8.6.])

8.4. Process entries

If a CBOR PSM process is intended to give new results in "eval" sections : it must have a unique key to define it. This key must corresponds to an entry (see parameter_map in root 5).

Currently, several entries are already defined, but anyone can create a new one:

xtandem psm and protein evaluations computed by the X!Tandem search engine
sage psm and protein evaluations computed by the Sage search engine

8.5. Protein description

The protein object is stored in the protein_map in root 8. The protein accession is used as a dictionary key to reference each protein object. Protein accession is used in psm [8.8.] to link PSMs to proteins.

"protein" object is composed of (the order of the elements must be respected) :

`description`

string *Required* — Protein description. This is formated exactly as a FASTA header : the first word **must** be the protein accession.

`sequence`

string *Required* — Protein amino acid sequence, maybe empty

`target`

boolean *Required* — Boolean (true by default), true if the protein belongs to the targeted sequences (target FASTA files). False otherwise.

`contaminant`

boolean *Required* — Boolean (false by default), true if the protein is tagged as a contaminant protein.

`props`

object *Optional* — Free structure designed to store any data (not related to a particular process) related to a protein as a propertie.

`eval`

object *Required* — Protein values computed by different algorithm or process (described in Process entry [8.4.]). Maybe empty.

8.6. Sample description

"sample" object is composed of :

`name`

string *Required* Sample name

`identification_file_list`

array *Optional* Identification engine search result files

`peaklist_file`

object *Required*

`scan_list`

array *Required* an array of “scan” objects (see scan [8.7.])

8.7. Scan description

“scan” object is composed of :

`id`

object *Required* Scan identifier object scan id [8.7.3.]

`precursor`

object *Required* Precursor description (MS1 related data) scan precursor [8.7.2.]

`ms2`

object *Required* MS2 related data scan ms2 [8.7.3.]

`props`

object *Optional* Free structure designed to store any data (not related to a particular process) related to a scan as a propertie.

`psm_list`

array *Required* an array of “psm” objects (see psm [8.8.])

8.7.1. Scan identifier description

`index`

integer *Optional*

Index of the scan in the original peak list file. Strongly recommended, but sometimes not available at all.

`native_id`

string *Required*

Unique identifier of the scan as defined in mzML file format.

`scan`

integer *Optional*

Scan number as written in the original peak list file.

8.7.2. Scan precursor description

`mz`

float *Required*

Scan precursor m/z

`z`

int *Required*

Scan precursor charge

`intensity`

float *Optional*

Scan precursor intensity

`mass`

float *Optional*

Scan precursor mass (molecule mass)

`mh`

float *Optional*

Scan precursor mass with 1 H+ (molecule mass + mhplus)

`alternatives`

array *Optional*

List of potential other precursors (not listed in the original mz files), called “alternatives”. each element is an object with “mz”, “z”, “intensity” of each potential precursor

8.7.3. Scan ms2 description

`rt`

float *Optional*

MS2 scan retention time

`spectrum`

object *Optional*

MS2 complete spectrum with “mz” and “intensity” arrays

8.8. PSM description

`proforma`

string *Required*

Peptide proforma notation

`protein_list`

array *Required*

Related proteins, described in c

`props`

object *Optional*

Peptide properties

`eval`

dict *Required*

Peptide values computed by different algorithm or process (described in Process entry [8.4.]).

8.8.1. Protein link description

`accession`

string *Required*

Protein accession : must be described in protein_map in root 8

`positions`

array *Required*

Positions of this peptide (proforma in psm 36) in the protein sequence. Maybe empty if the position is not known.

Important: a position starts at 0 (first amino acid) to N-1 (N is the protein length) for the last amino acid.

Chapter 9

Reading Sage results

9.1. Convert Sage results to psmCBOR

sage2psmcbor [OPTIONS] <JSON>

The sage2psmcbor command line interface program performs the conversion of files from Sage search engine combining JSON and TSV inputs to a CBOR-based format (psmCBOR) that serializes Peptide Spectrum Matches (PSMs).

Arguments

<JSON>	The Sage search engine JSON output file. It contains the search parameters and The path to the associated Tabulated Separated Value file (TSV) containing PSMs.

Options

-o, --output	A PSM results data file in the psmCBOR format

License GPL-3.0-or-later
Contribute https://forge.inrae.fr/pappso/pappsomspp

Debian package

Install PAPPSO source list

```
1 sudo apt update
2 sudo apt install libpappsomspp-cli
```

Chapter 10

Reading mzIdentML files

10.1. Convert mzIdentML results to psmCBOR

mzid2cbor [OPTIONS]

The mzid2cbor command line interface program performs the conversion of files from mzIdentML to a CBOR-based format (psmCBOR) that serializes Peptide Spectrum Matches (PSMs).

Options

-i, --input	A PSM results data file in the mzIdentML format
-o	A PSM results data file in the psmCBOR format
-h, --help	Displays help on commandline options
-v, --version	Displays version information

License GPL-3.0-or-later

Debian package

Install PAPPSO source list

```
1 sudo apt update
2 sudo apt install grouping-protein
```

Chapter 11

psmcborfilter

psmcborfilter [OPTIONS]

PSM CBOR data file processing: select scans and/or psms using threshold and properties.

Options

-i, --input	input psmCBOR file name. if not defined INPUT is read from the standard input
-o, --output	output psmCBOR file name. if not defined OUTPUT is written to the standard output
-p, --parameters	JSON scan and psm filter parameters. This can be the path to a JSON file or directly json content
--ms2	add or remove MS2 spectrum peak list (taken from mzML or mzCBOR file)

License GPL-3.0-or-later
Contribute https://codeberg.org/PAPPSO/pypsmcbor

Debian package

Install PAPPSO source list

```
sudo apt update
sudo apt install python3-pypappsocli
```

11.0.1. JSON parameter file

Filtering PSM to take only Sage PSMs where “spectrum_q” (spectrum Q-Value) is below the 1% threshold:

```
{ "psm_filters": [["sage", "spectrum_q", { "comp": "<", "threshold": 0.01 }]] }

```

Filtering scans on a retention time interval:

```
{ "scan_filters": [["ms2", "rt", { "range": [2400, 2450] }]] }

```

Filtering scans from a given spectrum index list:

```
{ "scan_filters": [["id", "index", { "list": [13258] }]] }

```

11.0.2. Filtering sage results PSMs at 1% FDR

```
{"psm_filters": [["sage", "spectrum_q",{"comp": "<", "threshold": 0.01}]] }
```

```
psmcborfilter -i sage_results.psmcbor -o sage_results_fdr_0.01.psmcbor -p
'{"psm_filters": [["sage", "spectrum_q",{"comp": "<", "threshold": 0.01}]] }'
```

Chapter 12

psmcborproteinfilter

12.1. psmcborproteinfilter

psmcborproteinfilter [OPTIONS]

PSM CBOR data file processing: filter protein and set a protein FDR threshold.

Options

-i, --input	input psmCBOR file name. if not defined INPUT is read from the standard input
-o, --output	output psmCBOR file name. if not defined OUTPUT is written to the standard output
-p, --parameters	JSON protein filter parameters

License GPL-3.0-or-later
Contribute https://codeberg.org/PAPPSO/pypsmcbor

Debian package

Install PAPPSO source list

```
sudo apt update
sudo apt install python3-pypappsocli
```

12.2. JSON parameter file

Filtering protein on the Sage spectrum Q-value :

- Compute the median spectrum Q-Value for each protein
- Sort protein on this median and compute a protein Q-value
- Filter proteins on this computed protein Q-value

```
{
  "protein_filter": {
    "fdr_threshold": 0.01,
    "protein_psm_score": {
      "engine": "sage",
      "field": "spectrum_q",
      "lower_is_better": true,
      "summarize": "median"
    }
  }
}

```

12.3. Filtering sage results PSMs and proteins at 1% FDR

One liner to filter sage results at 1% FDR at PSM and protein level

```
sage2psmcbor results.json | psmcborfilter -p '{"psm_filters": [["sage",
"spectrum_q",{"comp": "<", "threshold": 0.01}]] }' | psmcborproteinfilter -o
sage_results_fdr_0.01.psmcbor -p '{"protein_filter": {"fdr_threshold":
0.01,"protein_psm_score": {"engine": "sage","field":
"spectrum_q","lower_is_better": true,"summarize": "median"}}}'
```

Chapter 13

psmcborfdr

psmcborfdr [OPTIONS]

report FDR (False Discovery Rate) of a given psmCBOR file

Options

-i, --input	input psmCBOR file name. if not defined INPUT is read from the standard input

License GPL-3.0-or-later
Contribute https://codeberg.org/PAPPSO/pypsmcbor

Debian package

Install PAPPSO source list

```
sudo apt update
sudo apt install python3-pypappsocli
```

13.0.1. JSON output file

On any psmCBOR file containing a target/decoy result :

```
psmcborfdr -i sage_output.cbor
```

FDR results are printed in standard output in JSON format:


```
{
  "fdr_protein": {
    "target": 10812,
    "decoy": 1108,
    "ratio": 0.1024787273399926
  },
  "fdr_psm": {
    "has_target": 155844,
    "has_decoy": 1557,
    "total": 157401,
    "target_based_ratio": 0.009891932071587855,
    "decoy_based_ratio": 0.009891932071587855
  },
  "fdr_proforma": {
    "has_target": 14609,
    "has_decoy": 667,
    "total": 15276,
    "target_based_ratio": 0.04566326263419743,
    "decoy_based_ratio": 0.04566326263419743
  },
```

```
  "fdr_peptide": {
    "has_target": 13748,
    "has_decoy": 667,
    "total": 14415,
    "target_based_ratio": 0.046271245230662505,
    "decoy_based_ratio": 0.046271245230662505
  }
}

```

13.0.2. Compute FDR on PSMs filtered at 1% FDR

```
psmcborfilter -i sage_output.cbor -p '{"psm_filters": [["sage", "spectrum_q",
{"comp": "<", "threshold": 0.01}]]}' | python3 cli/psmcborfdr.py
```

Chapter 14

mcqCBOR MassChroQ quantification



Chapter 15

My Page