
2isa Documentation

Release

Martin Larralde, Tom Lawson, Reza Salek

February 09, 2017

1	Documentation	3
1.1	Command Line Applications / Libraries	3
1.2	Graphical User Interfaces	31
1.3	Galaxy interfaces	45
1.4	Other	51
2	About	55
2.1	Contacts	55

The 2isa suite is a collection of programs that can extract metadata out of various metabolomics file formats and write them to [ISA-Tab](#) files, creating the backbone for an ISA-Tab study that can then be shared on databases such as [MetaboLights](#). Right now the following source files are supported:

- .mzML** XML-like files containing derived spectral data of mass spectrometry scans
- .imzML** XML-like files containing metadata about imaging mass spectrometry scans
- .nmrML** XML-like files containing derived spectral data of nuclear magnetic resonance scans

Documentation

1.1 Command Line Applications / Libraries

The command line applications / libraries are currently divided into the `mzml2isa` package (covering `mzML` and `imzML`) and `nmrml2isa` (covering `nmrML`).

1.1.1 `mzml2isa`

`mzml2isa` is a program that allows you to convert metabolomic studies in **.mzML** (and **.imzML**) format to the open **ISA-Tab** standard supported by the MetaboLights database.

Installation

`mzml2isa` is available on PyPI, so if `pip` is installed on your machine, installing `mzml2isa` should be quite straightforward, without the need to worry about dependencies. It is also possible to install `mzml2isa` development version directly from the [mzml2isa GitHub repository](#).

Windows

Windows users can either install `mzml2isa` as described below or use a ‘ready to go’ Windows executable (**no python install required**). The executables be found on the `mzml2isa` [Github release section](#). See file `mzml2isa_cli.exe`.

Dependencies

Warning: Without `pip` you’ll have to install the requirements yourself, otherwise running `setup.py` will fail when trying to import **`mzml2isa`** for the first time.

Requirements

`pronto` an interface to ontology files (used to import the MS controlled vocabulary)

Extras

lxml an XML parser generally quicker than the builtin python xml.cElementTree (used as an alternative to parse the mzML files)

Note: If installing lxml on windows, you can either:

1. Compile it directly (requires Microsoft Visual C++ to be installed on the Windows machine): run the command `set STATICBUILD=true && pip install lxml`
 2. Install the [unofficial windows binaries \(whls\)](#)
-

PyPI (stable version)

With pip Just run one of the following commands in a terminal:

```
pip install mzml2isa          # install in /usr, needs super-user rights
pip install mzml2isa --user  # install in ~/.local/
```

Without pip This requires the python setuptools module to be installed, as well as the dependencies listed above. Download the latest stable release from the [PyPI repository of mzml2isa](#) , unpack it and install it by running the following commands:

```
tar xf mzml2isa-xx.yy.zz.tar.gz # replace with whatever version you downloaded
cd mzml2isa-xx.yy.zz.tar.gz
python setup.py install          # will require super-user rights
```

GitHub (development version)

Warning: Please check the current version of the program passes the Travis CI validation beforehand, or you could be installing a non-functional version of the program ! The program is working if the badge above displays .

With pip

```
pip install git+git://github.com/ISA-tools/mzml2isa
```

Without pip

```
git clone https://github.com/ISA-tools/mzml2isa
cd mzml2isa
python setup.py install
```

Usage

Folder structure

mzML For mzML files, mzml2isa only requires that you put all you files in the same folder, and parsing should work fine. Note that reference to RAW files is extracted from the mzML files, so if you plan to create an ISA archive after mzml2isa creates ISA files, don't forget to include those as well.

Example structure:

```
/
home/
  metabolomics/
    MZML study/      # the name of the folder doesn't matter
      Sample1.mzML   # the name of the file must correspond to the sample name
      Sample2.mzML
      ...
```

imzML For imzML files, you must have your imzML files in the same folder, but also the low- and high-res images of your scans (*.jpg, *.ibd, *.ndpi, etc...) named exactly like the imzML file of that particular scan (as mzml2isa scans the directory of the source file to eventually extract the location of those images):

```
/
home/
  metabolomics/
    MTBLS289/
      A14 5cm S3.ndpi
      A14 5cm S3.jpg           # Sample name will be
      A14 5cm S3-centroid.ibd  # "A14 5cm S3"
      A14 5cm S3-centroid.imzML # and centroid and profile
      A14 5cm S3-profile.ibd   # scans will be merged.
      A14 5cm S3-profile.imzML
      A14 10cm S3.ndpi
      A14 10cm S3.jpg          # Sample name will be
      A14 10cm S3-centroid.ibd # "A14 10cm S3"
      A14 10cm S3-centroid.imzML # and centroid and profile
      A14 10cm S3-profile.ibd  # scans will be merged
      A14 10cm S3-profile.imzML
      ...
```

Furthermore, mzml2isa tries to merge centroid and profile scans of the same sample into the same ISA-Tab row. Merging of said files relies on alphabetic order of files to make centroid and profile scans correspond to each other, which means the following structure will work:

```
IMZML study/
  Sample1_centroid.imzML
  Sample1_profile.imzML
  Sample2_centroid.imzML
  Sample2_profile.imzML
  ...
```

This will also work:

```
IMZML study/
  c_Sample1.imzML
  p_Sample1.imzML
  c_Sample2.imzML
  p_Sample2.imzML
  ...
```

Or even this:

```
IMZML study/
  Sample1_1.imzML
  Sample1_2.imzML
  Sample2_1.imzML
```

```
Sample2_2.imzML
...
```

Any structure in which centroid and profile scans are sequentially in the same rank following alphabetic order will probably work.

CLI

```
mzml2isa -i /path/to/mzml/folder -o /path/to/out_folder -s STUDYID
```

Python module

```
from mzml2isa.parsing import full_parse
full_parse("/path/to/mzml/folder", "/path/to/out_folder", "STUDYID")
# this will do the same thing as the command line above.
```

See the [Examples](#) and the [API reference](#) for more hindsights.

Editing with ISAcreeator

The ISA-Tab structure can be further populated with the [ISAcreeator software](#).

See brief [tutorial](#) for more details.

Examples

CLI

Create a new MetaboLights study with existing nmrML data:

```
mzml2isa -i "/home/metabolomics/NMRML study" -o "/home/metabolomics/isa" -s MZML1
```

This will create a new folder and write the following ISA-Tab files:

```
/
home/
  metabolomics/
    isa/
      a_MZML1_metabolite_profiling_mass_spectrometry.txt
      i_Investigation.txt
      s_MZML1.txt
```

Python module

Note: Syntax for imzML and mzML is the same, except when the class *mzMLmeta* is used: for imzML files, simply replace *mzMLmeta* with *imzMLmeta*.

Extract metadata of a file called *sample1.mzML* and print a JSON dictionary containing those metadata

```
python -m mzml2isa.mzml sample1.mzML
```

Or within a Python program

```
from mzml2isa.mzml import mzMLmeta
mz = mzMLmeta('sample1.mzML')

# json formatted dict
print(mz.meta__json)

# same metadata, Python dict
print(mz.meta['Sample Name']['value'])
```

Frequently Asked Questions

What metadata does mzml2isa extract from .mzML files ?

See *mzML*.

What metadata does mzml2isa extract from .imzML files ?

See *imzML*.

How do I report an issue / ask for a new feature ?

Go on the [GitHub repository issue tracker](#) of the program, check that no similar issue already exist, and if so, open a new issue and include the following information:

- what software and software version you used (you can know the version of the program by running `mzml2isa --version` in a terminal)
- **one of the files you used** (without this, nothing can be done to analyze the issue !)
- any remark you might have on how you think your file may *not* be generic (special instrument, multiple instruments, special parameters, etc.)
- the complete traceback of the error if you report a crash

The program crashed when I used it on my files, what should I do ?

While the **mzML** file format is supposed to be a standard, it so happens that depending on what hardware and software were used to generate .mzML files after a Mass Spectrometry study, some tags may not be named the same, and that some metadata may be absent or elsewhere than expected in the final .mzML file.

Even though the example files used to test mzml2isa are thought to be comprehensive, it might happen that the files you use are one of the special cases we did not consider. But a new version of the program may have corrected the bug that hit you already, so make sure **you are running the latest version of mzml2isa**.

To communicate that crash to us, see *How do I report an issue / ask for a new feature ?*

Extracted Terms list

mzML

Data Transformation Name

```
- type:
  list of cv terms
- definition:
  The successive transformation process the MS data underwent
  (all being subclasses of NMR:1400042)
- example value:
  {"entry_list": [
    {"accession": "http://purl.obolibrary.org/obo/MS_1000544",
      "name": "Conversion to mzML", "ref": "MS"},
    {"accession": "http://purl.obolibrary.org/obo/MS_1000035",
      "name": "peak picking", "ref": "MS"}
  ]}
```

Data Transformation software

```
- type:
  cv term
- definition:
  The software that was used for the data transformations
- example value:
  {"accession": "http://purl.obolibrary.org/obo/MS_1000615",
    "name": "ProteoWizard software", "ref": "MS"}
```

Data Transformation software version

```
- type:
  free text
- definition:
  The version of the data transformation software
- example value:
  {"value": "3.0.6002"}
```

Data file checksum type

```
- type:
  cv term w/ value
- definition:
  The type of the checksum used to check the data file integrity
- example value:
  {"accession": "http://purl.obolibrary.org/obo/MS_1000569",
    "name": "SHA-1", "ref": "MS",
    "value": "58f67219180d0e67da2b8f1652681b925371fcb9"}
```

Data file content

```
- type:
  list of cv terms
- definition:
  The contents of the Raw Data files
- example value:
  {"entry_list": [
    {"accession": "http://purl.obolibrary.org/obo/MS_1000579",
      "name": "MS1 spectrum", "ref": "MS"},
    {"accession": "http://purl.obolibrary.org/obo/MS_1000580",
```

```
    "name": "MSn spectrum", "ref": "MS"}
  ]},
```

Derived Spectral Data File

```
- type:
  free text
- definition
  The name of the derived spectral data file (the .mzML file itself)
- example value:
  {"value": "Person30_RBC_elder_NEG.mzML"}
```

Detector

```
- type:
  cv term
- definition
  The detector used in the mass spectrometer (child of MS:1000026)
- example value:
  {"accession": "http://purl.obolibrary.org/obo/MS_1000253",
   "name": "electron multiplier", "ref": "MS"}
```

Inlet type

```
- type
  cv term
- definition
  The type of inlet (child of MS:1000007)
- example value:
  {"accession": "http://purl.obolibrary.org/obo/MS_1000057",
   "name": "electrospray inlet", "ref": "MS"}
```

Instrument

```
- type
  cv term
- definition
  The mass spectrometry instrument used for the scan
- example value:
  {"accession": "http://purl.obolibrary.org/obo/MS_1000449",
   "name": "LTQ Orbitrap", "ref": "MS"}
```

Instrument manufacturer

```
- type
  cv term
- definition
  The manufacturer of the MS instrument
- example value:
  {"accession": "http://purl.obolibrary.org/obo/MS_1000483",
   "name": "Thermo Fisher Scientific instrument model", "ref": "MS"}
```

Instrument serial number

```
- type
  free text
- definition
  The serial number of the MS instrument
  "value": "1057B"
```

Instrument software

```
- type
  cv term
- definition
  The software used by the MS instrument
- example value:
  {"accession": "http://purl.obolibrary.org/obo/MS_1000532",
   "name": "Xcalibur", "ref": "MS"}
```

Instrument software version

```
- type:
  free text
- definition
  The version of the Instrument software
- example value:
  {"value": "2.2"}
```

Ion source

```
- type
  cv term
- definition
  The method by which gas phase ions are generated from the sample.
- example value:
  {"accession": "http://purl.obolibrary.org/obo/MS_1000073",
   "name": "electrospray ionization", "ref": "MS"}
```

MS Assay Name

```
- type
  free text
- definition
  The name of the MS Assay (mzML file name w/o extension)
- example value
  {"value": "Person30_RBC_elder_NEG"}
```

Mass analyzer

```
- type
  cv term
- definition
  How ions are separated according to their mass-to-charge ratio
- example value
  {"accession": "http://purl.obolibrary.org/obo/MS_1000083",
   "name": "radial ejection linear ion trap", "ref": "MS"}
```

Native spectrum identifier format

```
- type
  cv term
- definition
  The format of the native spectrum identifier
- example value
  {"accession": "http://purl.obolibrary.org/obo/MS_1000768",
   "name": "Thermo nativeID format", "ref": "MS"}
```

Number of scans

```
- type:
  int
- definition
```

```

The number of scans performed
- example value
{"value": 1073}

```

Raw Spectral Data File

```

- type
  free text
- definition
  The raw spectral data file (as extracted from the .mzML)
- example value
{"value": "20120627_SAM00533_RBC_NEG.RAW"}

```

Raw data file format

```

- type
  cv term
- definition
  The format of the raw data file
- example value
{"accession": "http://purl.obolibrary.org/obo/MS_1000563",
 "name": "Thermo RAW format","ref": "MS"}

```

Sample Name

```

- type
  free text
- definition
  The name of the sample (populated as the .mzML filename w/o extension)
- example value
{"value": "Person30_RBC_elder_NEG"}

```

Scan m/z range

```

- type
  free text
- definition
  The isolation window m/z range used during the scans
- example value
{"value": "50-1945"}

```

Scan polarity

```

- type
  free text
- definition
  The polarity of the scan (can be 'positive', 'negative', 'alternating' or 'n/a')
- example value
{"value": "negative"}

```

Note: It would be preferable for this to be a cv term, but while the PSI-MS ontology provides [positive scan](#) and a [negative scan](#) term, it doesn't provide any **alternating scan** term although referencing it among the possible [scan polarity](#) values. A request to add **alternating scan** to the MS ontology will probably be made.

Time range

```

- type:
  free text
- definition

```

```
The range of scan start times of all the scans
- example value
{"value": "0.0061-30.0042"}
```

imzML

All the aforementioned (if relevant), and the following

Binary file checksum type

```
- type:
  cv term w/ value
- definition
  The value and type of the checksum used to check the binary file.
- example value:
{"accession": "IMS:1000091", "name": "ibd SHA-1", "ref": "IMS"}
```

Detector mode

```
- type
  list of cv terms
- definition
  The mode the detector was set on during the scans
- example value
{"entry_list": [
  {"accession": "http://purl.obolibrary.org/obo/MS_1000118",
   "name": "pulse counting", "ref": "MS"},
]}
```

High-res image

```
- type
  free text
- definition
  The name of the high-res image file
- example value
{"value": "BRB04W.ndpi"}
```

Warning: Although the default value for this term will always be extracted as "name_of_the_imzML_file".ndpi, mzml2isa will always **try to find** that file in the parsed folder. Which means, if it can find it, **nothing will be extracted for that value.**

Ibd binary type

```
- type
  cv term
- definition
  The type of data contained in the ibd binary file
- example value
{"accession": "IMS:1000031",
 "name": "processed", "ref": "IMS"}
```

Line scan direction

```
- type
  cv term
- definition
  The direction in which the lines of the sample were scanned
```



```
- example value
{"accession": "IMS:1000491",
 "name": "linescan left right","ref": "IMS"}
```

Linescan sequence

```
- type
  cv term
- definition
  The direction of the succession of the assembling of the linescans.
- example value
{"accession": "IMS:1000401",
 "name": "top down","ref": "IMS"}
```

Max count of pixel x

```
- type
  cv term w/ value
- definition
  The max length of the x axis among all scans
- example value
{"accession": "IMS:1000042","name": "max count of pixel x",
 "ref": "IMS","value": "103"}
```

Max count of pixel y

```
- type
  cv term w/ value
- definition
  The max length of the y axis among all scans
- example value
{"accession": "IMS:1000043","name": "max count of pixel y",
 "ref": "IMS","value": "58"}
```

Max dimension x

```
- type
  cv term w/ value w/ unit
- definition
  The size of the largest scanned line of x axis
- example value
{"accession": "IMS:1000044",
 "name": "max dimension x", "ref": "IMS",
 "value": "9095"
 "unit": {"accession": "UO:0000017",
          "name": "micrometer","ref": "UO"} }
```

Max dimension y

```
- type
  cv term w/ value w/ unit
- definition
  The size of the largest scanned line of y axis
- example value
{"accession": "IMS:1000044",
 "name": "max dimension x", "ref": "IMS",
 "value": "8900"
 "unit": {"accession": "UO:0000017",
          "name": "micrometer","ref": "UO"} }
```

Pixel size x

```
- type
  cv term w/ value w/ unit
- definition
  The size of the x dimension of a pixel
- example value
  {"accession": "IMS:1000046",
   "name": "pixel size x", "ref": "IMS",
   "value": "85",
   "unit": {"accession": "UO:0000017",
            "name": "micrometer", "ref": "UO"} }
```

Pixel size y

```
- type
  cv term w/ value w/ unit
- definition
  The size of the y dimension of a pixel
- example value
  {"accession": "IMS:1000047",
   "name": "pixel size y", "ref": "IMS",
   "value": "85",
   "unit": {"accession": "UO:0000017",
            "name": "micrometer", "ref": "UO"} }
```

Scan pattern

```
- type
  cv term
- definition
  The pattern how the image was scanned (child of IMS:1000041)
- example value
  {"accession": "IMS:1000413",
   "name": "flyback", "ref": "IMS"}
```

Scan type

```
- type
  cv term
- definition
  The direction in which the lines were scanned (child of IMS:1000048)
- example value
  {"accession": "IMS:1000480",
   "name": "horizontal line scan", "ref": "IMS"}
```

Solvent

```
- type
  cv term w/ value
- definition
  The solvent used on the sample
- example value
  {"accession": "IMS:1001211",
   "name": "solvent", "ref": "IMS",
   "value": "methanol"}
```

Note: That parameter was only seen once, and it was a free text value. It would be better if this was a cv term, so changing the solvent column in the ISA file may be considered.

Solvent flowrate

```
- type
  cv term w/ value w/ unit
- definition
  The flowrate of the solvent on the sample
- example value
  {"accession": "IMS:1001213",
   "name": "solvent flowrate", "ref": "IMS",
   "value": "1.5"
   "unit": {"accession": "IMS:1000131",
            "name": "milliliter per minute", "ref": "IMS"} }
```

Spectrum representation

```
- type
  list of cv terms
- definition
  The way in which the spectra are represented
- example value
  {"entry_list": [
    {"accession": "MS:1000127",
     "name": "centroid spectrum", "ref": "MS"}
  ]}
```

Target material

```
- type
  free text
- definition
  The material that is targeted by the IMS instrument
- example value
  {"accession": "IMS:1000202",
   "name": "target material", "ref": "IMS",
   "value": "Glas"}
```

Universally unique identifier

```
- type
  cv term w/ value
- definition
  The UUID of the IBD binary file
- example value
  {"accession": "IMS:1000080",
   "name": "universally unique identifier", "ref": "IMS",
   "value": "{6E2F1092-F43A-4169-94CB-61DD5A61047E}"}
```

Warning: The following items are being extracted with a **redundant cv term**, which we'll probably get rid of in the future:

- *Target material*
- *Solvent*
- *Solvent flowrate*
- *Max dimension x*
- *Max dimension y*
- *Max count of pixel x*
- *Max count of pixel y*
- *Max dimension x*
- *Max dimension y*

API reference

mzMLmeta

class `mzml2isa.mzml.mzMLmeta` (*in_file*, *ontology=None*, *complete_parse=False*)

Class to store and obtain the meta information from the mzML file

The class uses the xpaths of mzML locations and then extracts meta information at these locations. The meta info taken is determined by the ontology terms and a set of rules associated with that term e.g. if it can be repeated, if has associated software if it has a value as well as name.

tree

lxml.etree.ElementTree – the tree object created from the mzML file

ns

dict – a dictionary containing the xml namespace mapping

obo

pronto.Ontology – the ontology object

meta

dict – structured dictionary containing extracted metadata

env

dict – the *environment variables*, tag names that are not standards among different mzML files.

build_env()

Build the env and the ns dictionaries.

cvParam_loop (*elements*, *location_name*, *terms*)

Loop through the elements and eventually update the self.meta dict.

Parameters

- **elements** (*iterator*) – the element containing the cvParam tags
- **location_name** (*str*) – Name of the xml location
- **terms** (*dict*) – terms that are to be extracted

data_file_content()

Extract the *Data file content* from all scans.

This method is called only in the case the FileContent xml Element contained no actual cvParam elements. This was witnessed in at least one file from a Waters instrument.

derived()

Get the derived meta information

The derived meta information includes all tags that are solely based on the file name, such as *MS Assay Name*, *Derived Spectral Data File* or *Sample Name*.

extract_meta (*terms*, *xpaths*)

Extract meta information for CV terms based on their location in the xml file

Updates the self.meta dictionary with the relevant meta information

Parameters

- **terms** (*dict*) – The CV and search parameters required at the xml locations
- **xpath** (*dict*) – the xpath locations to search

See also:`cvParam_loop`**merge_entries** (*name*)

An unoptimized way of merging meta entries only made of duplicates.

This is only useful when the `spectrum_meta` method is called, as a way of reducing the size of some meta entries that add no interesting information (for instance, when all binary data is compressed the same way, it is useless to know that for each scan).

Parameters *name* (*str*) – the entry to de-duplicate

Returns the list of the list with deduplicated arguments

Return type `list`

Note: Using an `OrderedSet` to deduplicate while preserving order may be a good idea (see <http://code.activestate.com/recipes/576694/>) for actual implementation

meta_isa

Returns the metadata dictionary with actual ISA headers

meta_json

Returns the metadata dictionary in json format

mzrange ()

Try to extract the m/z range of all scans.

polarity ()

Iterates over all scans to get the *average* polarity.

scan_num ()

Extract the total number of scans.

software (*soft_ref*, *name*)

Get associated software of cv term. Updates the self.meta dictionary

Parameters

- **soft_ref** (*str*) – the reference to the software found in another xml “ref” attribute.
- **name** (*str*) – Name of the associated CV term that the software is associated to.

spectrum_meta ()

Extract information of each spectrum in entry lists.

This method is only called is the **complete_parse** parameters was set as True when the `mzMLmeta` object was created. This requires more time as iterating through hundreds of elements is bound to be more performance hungry than just a few elements. It is believed to be useful when `mzml2isa` is used as a parsing library.

timerange ()

Try to extract the Time range of all the scans.

Time range consists in the smallest and largest time the successive scans were started. The unit (most of the time *minute*) will be extracted as well if possible.

urlize ()

Urlize all accessions within the meta dictionary

ISA_Tab

class `mzml2isa.isa.ISA_Tab` (*out_dir*, *name*, ***kwargs*)

Class to export a list of mzML or imzML metadata dictionaries to ISA-Tab files

usermeta

dict – a dictionary containing metadata defined by the user, such as “Study Publication” or “Submission Date”. Defaults to None.

isa_env

dict – a dictionary containing various environment variables associated with the current ISA_Tab object (such as ‘Study file name’, ‘Written assays’, etc.)

create_assay (*metalist*, *headers*, *data*, ***kwargs*)

Write the assay file

Parameters

- **metalist** (*list*) – a list of mzml or imzml metadata dictionaries
- **datatype** (*str*) – the datatype of the study (either ‘mzML’ or ‘imzML’)
- **headers** (*list*) – the list containing the assay headers row

Keyword Arguments **split** (*bool*, *optional*) – a boolean stating if assay files should be split based on their polarities. [default: True]

create_investigation (*metalist*, *datatype*)

Write the investigation file

Parameters

- **metalist** (*list*) – a list of mzml or imzml metadata dictionaries
- **datatype** (*str*) – the datatype of the study (either ‘mzML’ or ‘imzML’)

create_study (*metalist*, *datatype*)

Write the study file

Parameters

- **metalist** (*list*) – a list of mzml or imzml metadata dictionaries
- **datatype** (*str*) – the datatype of the study (either ‘mzML’ or ‘imzML’)

make_assay_template (*metalist*, *datatype*)

Generate the assay template rows

Parameters

- **metalist** (*list*) – a list of mzml or imzml metadata dictionaries
- **datatype** (*str*) – the datatype of the study (either ‘mzML’ or ‘imzML’)

Returns

tuple: a tuple containing the list containing the assay headers row and the list containing the assay data row based on the cardinality of elements in the metalist

static unparameter (*string*)

Extract string ‘s’ from ‘Parameter Value[s]’

Parameters **string** (*str*) – full string

Returns the extracted substring

Return type `str`

write (*metalist*, *datatype*, ***kwargs*)
Generate and write the ISA files

Parameters

- **metalist** (*list*) – a list of mzml or imzml metadata dictionaries
- **datatype** (*str*) – the datatype of the study (either ‘mzML’ or ‘imzML’)

Keyword Arguments **split** (*bool*, *optional*) – a boolean stating if assay files should be split based on their polarities. [default: True]

1.1.2 nmrm12isa

DOI

nmrm12isa is a program that allows you to convert metabolomic studies in **.nmrML** format to the open **ISA-Tab** standard supported by the MetaboLights database.

Installation

nmrm12isa is available on PyPI, so if *pip* is installed on your machine installing nmrm12isa should be quite straightforward, without the need to worry about dependencies. It is also possible to install nmrm12isa development version from its [GitHub repository](#). Please note that the GitHub version of the program may fix bugs but also introduce new bugs, and that it might not work on your files.

Windows

Windows users can either install mzml2isa as described below or use a ‘ready to go’ Windows executable (**no python install required**). The executables be found on the nmrm12isa [Github release section](#). See file nmrm12isa_cli.exe.

Dependencies

Warning: Without pip you’ll have to install the requirements yourself, otherwise running `setup.py` will fail when trying to import **nmrm12isa** for the first time.

Requirements

pronto an interface to ontology files (used to import the MS controlled vocabulary)

chainmap (**required for Python 2 or PyPy**) a polyfill implementation of `collections.ChainMap` (available in Python 3 stdlib)

Extras

lxml an XML parser generally quicker than the builtin python `xml.cElementTree` (used as an alternative to parse the nmrML files)

Note: If installing lxml on windows, you can either:

1. Compile it directly (requires Microsoft Visual C++ to be installed on the Windows machine): run the command `set STATICBUILD=true && pip install lxml`
 2. Install the [unofficial windows binaries \(whls\)](#)
-

PyPI (stable version)

With *pip* Just run one the following commands in a terminal:

```
pip install nmrml2isa          # install for all users, needs super-user rights
pip install nmrml2isa --user  # install only for the current user, no rights needed
```

Without *pip* This requires the python `setuptools` module to be installed, as well as the dependencies listed above. Download the latest stable release from the [PyPI repository](#), unpack it and install it by running the following commands:

```
tar xf nmrml2isa-xx.yy.zz.tar.gz # replace with whatever version you downloaded
cd nmrml2isa-xx.yy.zz.tar.gz
python setup.py install          # will require super-user rights
```

GitHub (development version)

Warning: Please check the current version of the program passes the Travis CI validation beforehand, or else you're likely to install a non-functioning version of the program ! The program is working if the badge above displays .

With *pip*

```
pip install git+git://github.com/ISA-tools/nmrml2isa
```

Without *pip*

```
git clone https://github.com/ISA-tools/nmrml2isa
cd nmrml2isa
python setup.py install
```

Usage

Folder structure

nmrML nmrml2isa only requires that you put all you nmrML and zipped raw files in the same folder, and parsing should work fine. Note that reference to RAW files is extracted from the mzML files, so if you plan to create an ISA archive after mzml2isa creates ISA files, don't forget to include those as well.

Example structure:

```
/
home/
  metabolomics/
    nmrML study/      # the name of the folder doesn't matter
```



```

Sample1.nmrML # the name of the file must correspond to the sample name
Sample2.zip   # the raw files should be zipped and called exactly like the nmrML
Sample2.nmrML
Sample2.zip
...

```

CLI

```
nrmrml2isa -i /path/to/nmrml/folder -o /path/to/out_folder -s STUDYID
```

Python module

```

from nrmrml2isa.parsing import full_parse
full_parse("/path/to/nmrml/folder", "/path/to/out_folder", "STUDYID")
# this will do the same thing as the command line.

```

See the [Examples](#) and the [API reference](#) for more hindsights.

Editing with ISAcreeator

The ISA-Tab structure can be further populated with the [ISAcreeator](#) software.

See brief [tutorial](#) for more details.

Examples

All these examples assume we are working on a folder organized as the following (which is not mandatory, you just have to put all your nmrML and zipped raw files in the same folder):

```

/
home/
  metabolomics/
    NMRML study/ # the name of the folder doesn't matter
      Sample1.nmrML # the name of the file must correspond to the sample name
      Sample1.zip   # the raw files must be zipped and named as the nmrML file
      Sample2.nmrML
      Sample2.zip
      ...

```

Standalone program

Create a new MetaboLights study with existing nmrML data:

```
nrmrml2isa -i "/home/metabolomics/NMRML study" -o "/home/metabolomics/isa" -s NMRML1
```

This will create a new folder and write the following ISA-Tab files:

```

/
home/
  metabolomics/
    isa/
      a_NMRML1_metabolite_profiling_NMR_spectroscopy.txt

```

```
i_Investigation.txt
s_NMRML1.txt
```

Python module

To do the exact same thing within a Python program:

```
from nmrml2isa.parsing import full_parse
full_parse("/home/metabolomics/NMRML study", "/home/metabolomics/isa", "NMRML1")
```

It is also possible only to extract metadatas: for instance, let's say we want to separate .nmrML files within the same directory based on what the Acquisition Nucleus was for that particular NMR scan:

```
from nmrml2isa.nmrml import nmrMLmeta
import os

sorted_files = {}
in_dir = '/home/metabolomics/NMRML study'

for file in os.listdir(in_dir):

    meta = nmrMLmeta(os.path.join(in_dir, file)).meta

    if 'Acquisition Nucleus' in meta.keys():

        if meta['Acquisition Nucleus']['name'] not in sorted_files.keys():
            sorted_files[meta['Acquisition Nucleus']['name']] = []

        sorted_files[meta['Acquisition Nucleus']['name']].append(file)
```

At the end of that snippet, `sorted_files` keys will be the different acquisition nuclei used and the value of each key will be a list of nmrML files where that nucleus was used.

Frequently Asked Questions

What metadata does nmrml2isa extract from .nmrML files ?

nmrml2isa extracts a lot more information than it actually writes to the final ISA-Tab files. To see a list of the metadata that are extracted and can be accessed, look at the [Extracted Terms list](#) page.

How do I report an issue / ask for a new feature ?

Go on the [GitHub repository issue tracker](#) of the program, check that no similar issue already exist, and if so, open a new issue and include the following information:

- what software and software version you used (you can know the version of the program by running `nmrml2isa --version` in a terminal)
- **one of the files you used** (without this, nothing can be done to analyze the issue !)
- any remark you might have on how you think your file may *not* be generic (special instrument, multiple instruments, special parameters, etc.)
- the complete traceback of the error if you report a crash

The program crashed when I used it on my files !

As the **nmrML** file format is quite new, there is still issues to fix concerning the standardization of the format. The converter might not work as intended, there might be missing tags in the nmrML file you tried to parse.

But most of the times those problems are not really complicated to mend, and they help making nmrml2isa a more and more generic program.

In order to help the resolution of such problems, see [How do I report an issue / ask for a new feature ?](#)

The parameter xxxxx does not get extracted properly !

First of all: is it present in the source .nmrML file ? It could happen that some parameters were not successfully written into the converted file when the nmrML converter converted the Raw NMR files.

Also, nmrml2isa relies on **controlled vocabulary terms** to extract those parameters from the source nmrML, so if the parameters are present but not labeled, there are large chances those parameters won't be extracted from the nmrML file.

If however you still think the problem comes from our side, see [How do I report an issue / ask for a new feature ?](#)

The parameter xxxxx is not extracted !

There is large chances that if a parameter you want to extract was not extracted is that the extraction for that parameter was not implemented. See [How do I report an issue / ask for a new feature ?](#) to report it and create a feature request.

Extracted Terms list

1r Data File

```
- type:
  free text
- definition:
  The path to the 1r Data File (relative to folder containing the .nmrML being parsed)
- example value:
  {'value': 'ADG10003u_007/10/pdata/1/1r'}
```

1r Data Format

```
- type:
  cv term
- definition:
  The format of the 1r Data file (subclass of NMR:1001459)
- example value:
  {"accession": "http://nmrml.org/cv/v1.0.rc1/nmrCV.owl#NMR:1400320",
   "name": "Bruker UXNMR/XWIN-NMR format", "ref": "NMRCV"}
```

1r Data Type

```
- type:
  cv term
- definition:
  The type of the 1r Data (subclass of NMR:1000317)
- example value:
  {"accession": "http://nmrml.org/cv/v1.0.rc1/nmrCV.owl#NMR:1000319",
   "name": "1R file", "ref": "NMRCV"}
```

Acquisition Nucleus

```
- type:
  cv term
- definition:
  The Nucleus which was used for the acquisition (reference to CHEBI is preferred)
- example value:
  {"accession": "http://purl.obolibrary.org/obo/CHEBI_49637",
   "name": "hydrogen atom", "ref": "CHEBI"}
```

Acquisition Parameter Data File

```
- type:
  free text
- definition:
  The path to the Acquisition Parameter Data File
- example value:
  {"value": "ADG10003u_007/10/acqus"}
```

Acquisition Parameter Data Format

```
- type:
  cv term
- definition:
  The format of the Acquisition Parameter Data File
- example value:
  {"accession": "http://nmrml.org/cv/v1.0.rc1/nmrCV.owl#NMR:1400320",
   "name": "Bruker UxnMR/XWIN-NMR format", "ref": "NMRCV"}
```

Baseline Correction Method

```
- type:
  cv term
- definition:
  The algorithm used for baseline correction
- example value:
  {"accession": "http://nmrML.org/nmrCV#NMR:1000225",
   "name": "baseline correction using polynomial function", "ref": "NMRCV"}
```

Calibration Reference Shift

```
- type:
  float w/ unit
- definition:
  The shift used for calibration
- example value:
  {}
```

Data Transformation Name

```
- type:
  list of cv terms
- definition:
  The successive transformation process the NMR data underwent
  (all being subclasses of NMR:1400042)
- example value:
  {"entry_list": [
    {"accession": "http://nmrml.org/cv/v1.0.rc1/nmrCV.owl#NMR:1000071",
     "name": "phase correction", "ref": "NMRCV"},
    {"accession": "http://nmrml.org/cv/v1.0.rc1/nmrCV.owl#NMR:1400044",
```

```
    "name": "fourier transformation", "ref": "NMRCV"}
  ]}
```

Data Transformation software

```
- type:
  cv term
- definition:
  The software used for the data transformation (subclass of NMR:1400213)
- example value:
  {"accession": "http://nmrML.org/nmrCV#NMR:1000352",
   "name": "Bruker XWIN-NMR software", "ref": "NMRCV"}
```

Data Transformation software version

```
- type:
  free text
- definition:
  The version of the data transformation software
- example value:
  {"value": "Version 3.5"}
```

Decoupling Nucleus:

```
- type:
  cv term
- definition:
  The atom used for decoupling (if any) (reference to CHEBI is preferred)
- example value:
  {"accession": "http://purl.obolibrary.org/obo/CHEBI_49637",
   "name": "hydrogen atom", "ref": "CHEBI"}
```

Derived Spectral Data File

```
- type:
  free text
- definition:
  The file containing the Derived Spectral Data (the nmrML file itself)
- example value:
  {"value": "MMBBI_10M12-CE01-1a.nmrML"}
```

First Order Phase Correction

```
- type:
  float w/ unit
- definition:
  The value of the parameter used to correct the phase.
- example value:
  {"value": "-9.745789",
   "unit": {"accession": "http://purl.obolibrary.org/obo/UO_0000185",
            "name": "degree", "ref": "UO"} }
```

Free Induction Decay Data File

```
- type:
  free text
- definition:
  The file containing the FID data
- example value:
  {"value": "ADG10003u_007/10/fid"}
```

Free Induction Decay Data Format

```
- type:
  cv term
- definition:
  The format of the Free Induction Decay Data file (subclass of NMR:1000767)
- example value:
  {"accession": "http://nmrML.org/nmrCV#NMR:1002003",
   "name": "Varian FID format", "ref": "NMRCV"}
```

Instrument

```
- type:
  cv term
- definition:
  The NMR instrument that was used (subclass of NMR:1000031)
- example value:
  {"accession": "http://nmrml.org/cv/v1.0.rc1/nmrCV.owl#NMR:1400236",
   "name": "INOVA", "ref": "NMRCV"}
```

Instrument manufacturer

```
- type:
  cv term
- definition:
  The manufacturer of the instrument (subclass of NMR:1400255)
- example value:
  {"accession": "http://nmrml.org/nmrCV.owl#NMR:1400257",
   "name": "Varian", "ref": "NMRCV"}
```

Instrument software

```
- type:
  cv term
- definition:
  The software used by the NMR instrument (subclass of NMR:1400213)
- example value:
  {"accession": "http://nmrML.org/nmrCV#NMR:1000352",
   "name": "Bruker XWIN-NMR software", "ref": "NMRCV"}
```

Instrument software version

```
- type:
  free text
- definition:
  The version of the instrument software
- example value:
  {"value": "Version 3.5"}
```

Irradiation Frequency

```
- type:
  float w/ unit
- definition:
  The irradiation frequency
- example value:
  {"value": "699.873291",
   "unit": {"accession": "http://purl.obolibrary.org/obo/UO_0000325",
            "name": "megaHertz", "ref": "UO"} }
```

Magnetic field strength

```
- type:
  float w/ unit
- definition:
  The strength of the magnetic field (a.k.a Effective Excitation Field strength)
- example value:
  {"value": "16.43813416009019",
   "unit": {"accession": "http://purl.obolibrary.org/obo/UO_0000228",
            "name": "tesla", "ref": "UO"} }
```

Note: If the nmrML file also references an Acquisition Nucleus, and that this acquisition nucleus is among known ones (1H, 2H, 13C, 14N, 15N, 17O, 31P), then the Magnetic field strength will be converted from mHz to Tesla for the output to be SI-compliant.

NMR Assay Name

```
- type:
  free text
- definition:
  The name of the NMR Assay (default is nmrML file name w/o extension)
- example value:
  {"value": "ADG10003u_007"}
```

NMR Probe

```
- type:
  cv term or free text
- definition:
  The NMR probe used for the scans.
- example value:
  {"name": "5 mm PATXI 1H-13C/15N XYZ-GRD Z561501/0002"
   "ref": "", "accession": ""} # No CV term
  {"accession": "http://nmrml.org/cv/v1.0.rc1/nmrCV.owl#NMR:1000236",
   "name": "5mm HCN probe", "ref": "NMRCV"} # CV term
```

Note: If the nmrML file contains both a controlled vocabulary referenced probe and an UserParam with a free text probe reference, only the controlled vocabulary term will be extracted.

NMR tube type

```
- type:
  cv term
- definition:
  The sample tube used in the NMR instrument (subclass of NMR:1400132)
- example value:
  {"accession": "http://nmrml.org/cv/v1.0.rc1/nmrCV.owl#NMR:1400132",
   "name": "Sample-tube", "ref": "NMRCV"}
```

Number of data points

```
- type:
  int
- definition:
  The number of data points measured
- example value:
  {"value": 65536}
```

Number of steady state scans

```
- type:
  int
- definition:
  The number of steady state scans performed
- example value:
  {"value": 8}
```

Number of transients

```
- type:
  int
- definition:
  The number of scans performed
- example value:
  {"value": 128}
```

Processing Parameter Data File

```
- type:
  free text
- definition:
  The file containing the Processing Parameter data
- example value:
  {"value": "ADG10003u_007/10/pdata/1/procs"}
```

Processing Parameter Data Format

```
- type:
  cv term
- definition:
  The format of the Processing Parameter Data file
- example value:
  {"accession": "http://nmrml.org/cv/v1.0.rc1/nmrCV.owl#NMR:1400320",
   "name": "Bruker UXNMR/XWIN-NMR format", "ref": "NMRCV"}
```

Pulse Sequence Data File

```
- type:
  free text
- definition:
  The file containing the Pulse sequence data
- example value:
  {"value": "ADG10003u_007/10/pulseprogram"}
```

Pulse Sequence Data Format

```
- type:
  cv term
- definition:
  The format of the Pulse Sequence Data file
- example value:
  {"accession": "http://nmrml.org/cv/v1.0.rc1/nmrCV.owl#NMR:1400320",
   "name": "Bruker UXNMR/XWIN-NMR format", "ref": "NMRCV"}
```

Pulse Width

```
- type:
  float w/ unit
- definition:
```



```

The pulse width of the scan
- example value:
  {"value": "7.750000",
   "unit": {"accession": "http://purl.obolibrary.org/obo/UO_0000029",
            "name": "microsecond", "ref": "UO"} }

```

Pulse sequence

```

- type:
  cv term or free text
- definition:
  The pulse sequence used for the scans (subclass of NMR:1400037)
- example value:
  {"name": "noesypr1d"
   "ref": "", "accession": ""} # No CV term
  {"name": "1D carr purcell meiboom gill pulse sequence",
   "accession": "http://nmrML.org/nmrCV#NMR:1400167", "ref": "NMRCV"} # CV term

```

Note: If the nmrML file contains both a controlled vocabulary referenced pulse sequence and an UserParam with a free text pulse sequence defined, only the controlled vocabulary term will be extracted.

Relaxation Delay

```

- type:
  float w/ unit
- definition:
  The relaxation delay
- example value:
  {"value": "3.000000",
   "unit": {"accession": "http://purl.obolibrary.org/obo/UO_0000010",
            "name": "secon", "ref": "UO"} }

```

Sample Name

```

- type:
  free text
- definition:
  The name of the sample (default is nmrML file w/o extension)
- example value:
  {"value": "ADG10003u_007"}

```

Sampling Strategy

```

- type:
  cv term
- definition:
  The sampling strategy used during the NMR scan (subclass of NMR:1000348)
- example value:
  {"accession": "http://nmrml.org/cv/v1.0.rc1/nmrCV.owl#NMR:1000349",
   "name": "uniform sampling", "ref": "NMRCV"}

```

Spectral Denoising Method

```

- type:
  cv term
- definition:
  The method used for spectral denoising

```

Spinning Rate

```
- type:
  float w/ unit
- definition:
  The spinning rate (if any)
- example value:
  {"value": "0",
   "unit": {"accession": "http://purl.obolibrary.org/obo/UO_0000169",
            "name": "dimensionless", "ref": "UO"} }
```

Sweep Width

```
- type:
  float w/ unit
- definition:
  The sweep width of the scan
- example value:
  {"value": "14005.602241",
   "unit": {"accession": "http://purl.obolibrary.org/obo/UO_0000106",
            "name": "hertz", "ref": "UO"} }
```

Study_contacts

```
- type:
  list
- definition:
  A list of contacts references found in the nmrML file
- example value:
  [{"first_name": "Lutger", "mid": "A.", "last_name": "Wessjohann",
   "mail": "Ludger.Wessjohann [a] ipb-halle.de"},
   {"first_name": "Mohamed", "mid": "A.", "last_name": "Frag",
   "mail": "mfrag73 [a] yahoo.com"}]
```

Temperature

```
- type:
  float w/ unit
- definition:
  The temperature of the sample
- example value:
  {"value": "300.0000",
   "unit": {"accession": "http://purl.obolibrary.org/obo/UO_0000012",
            "name": "kelvin", "ref": "UO"} }
```

Window Function Method

```
- type:
  cv term
- definition:
  The format of the window function method that was used (subclass of NMR:1400068)
- example value:
  {"accession": "http://nmrml.org/cv/v1.0.rcl/nmrCV.owl#NMR:1400069",
   "name": "exponential multiplication window function", "ref": "NMRCV"}
```

X axis range

```
- type:
  free text w/ unit
- definition:
  The range of the X axis
- example value:
```

```
{ "value": "2-25",
  "unit": { "accession": "http://purl.obolibrary.org/obo/UO_0000169",
            "name": "parts per million", "ref": "UO" } }
```

Y axis type

```
- type:
  cv term
- definition:
  The unit of the Y axis of the spectrum
- example value:
  { "accession": "http://purl.obolibrary.org/obo/UO_0000169",
    "name": "dimensionless", "ref": "UO" }
```

Zero Value Phase Correction

```
- type:
  float w/ unit
- definition:
  The parameter used for the Zero Value Phase Correction
- example value:
  { "value": "-37.977290"
    "unit": { "accession": "http://purl.obolibrary.org/obo/UO_0000185",
              "name": "degree", "ref": "UO" } }
```

API reference

nmrMLmeta

```
class nmrm2isa.nmrml.nmrMLmeta (in_file, cached_onto=None)
```

ISA_Tab

```
class nmrm2isa.isa.ISA_Tab (out_dir, name, usermeta=None, template_directory=None)
```

1.2 Graphical User Interfaces

The graphical user interfaces are divided into mzmlisa-qt (covering mzML), imzml2isa-qt (covering imzML) and nmrm2isa-qt (covering nmrML).

1.2.1 mzml2isa-qt

mzml2isa-qt is a Graphical User Interface for mzml2isa, that allows converting mzML files to ISA-Tab files with the help of mzml2isa as well as adding more metadata to the generated ISA files through its interface.

Installation

mzml2isa-qt is available on PyPI, so if `pip` is installed on your machine, installing mzml2isa should be quite straightforward, without the need to worry about dependencies. It is also possible to install mzml2isa development version directly from the [mzml2isa-qt GitHub repository](#).

Warning: mzml2isa-qt is a **Python3** program only !

Windows

Windows users can either install mzml2isa-qt as described below or use a 'ready to go' Windows executable (**no python install required**). The executables be found on the mzml2isa-qt [Github release section](#). See file mzml2isa_gui.exe.

Dependencies

Warning: Without pip you'll have to install the requirements yourself, otherwise running `setup.py` will fail when trying to import **nmrml2isa** for the first time.

Requirements

nmrml2isa the parsing library (see [here](#))

PyQt5 the GUI toolkit used by nmrml2isa-qt (see [here](#) for more information about PyQt)

PyPI (stable version)

With *pip* Just run one of the following commands in a terminal:

```
pip install mzml2isa-qt          # install on the machine, needs super-user rights
pip install mzml2isa-qt --user  # install only for the current user, no rights needed
```

Without *pip* This requires the python `setuptools` module to be installed, as well as the dependencies listed above. Download the latest stable release from the [PyPI repository](#), unpack it and install it by running the following commands:

```
tar xf mzml2isa-qt-xx.yy.zz.tar.gz # replace with whatever version you downloaded
cd mzml2isa-qt-xx.yy.zz.tar.gz
python setup.py install             # will require super-user rights
```

GitHub (development version)

With *pip*

```
pip install git+git://github.com/ISA-tools/mzml2isa-qt
```

Without *pip*

```
git clone https://github.com/ISA-tools/mzml2isa-qt
cd mzml2isa-qt
python setup.py install
```

Usage

Video tutorial

See below for 5 min video for how to use either `mzml2isa-qt`, `imzml2isa-qt` or `nmrml2isa-qt`.

<https://www.youtube.com/watch?v=xy3uusQRkbI>

Folder structure

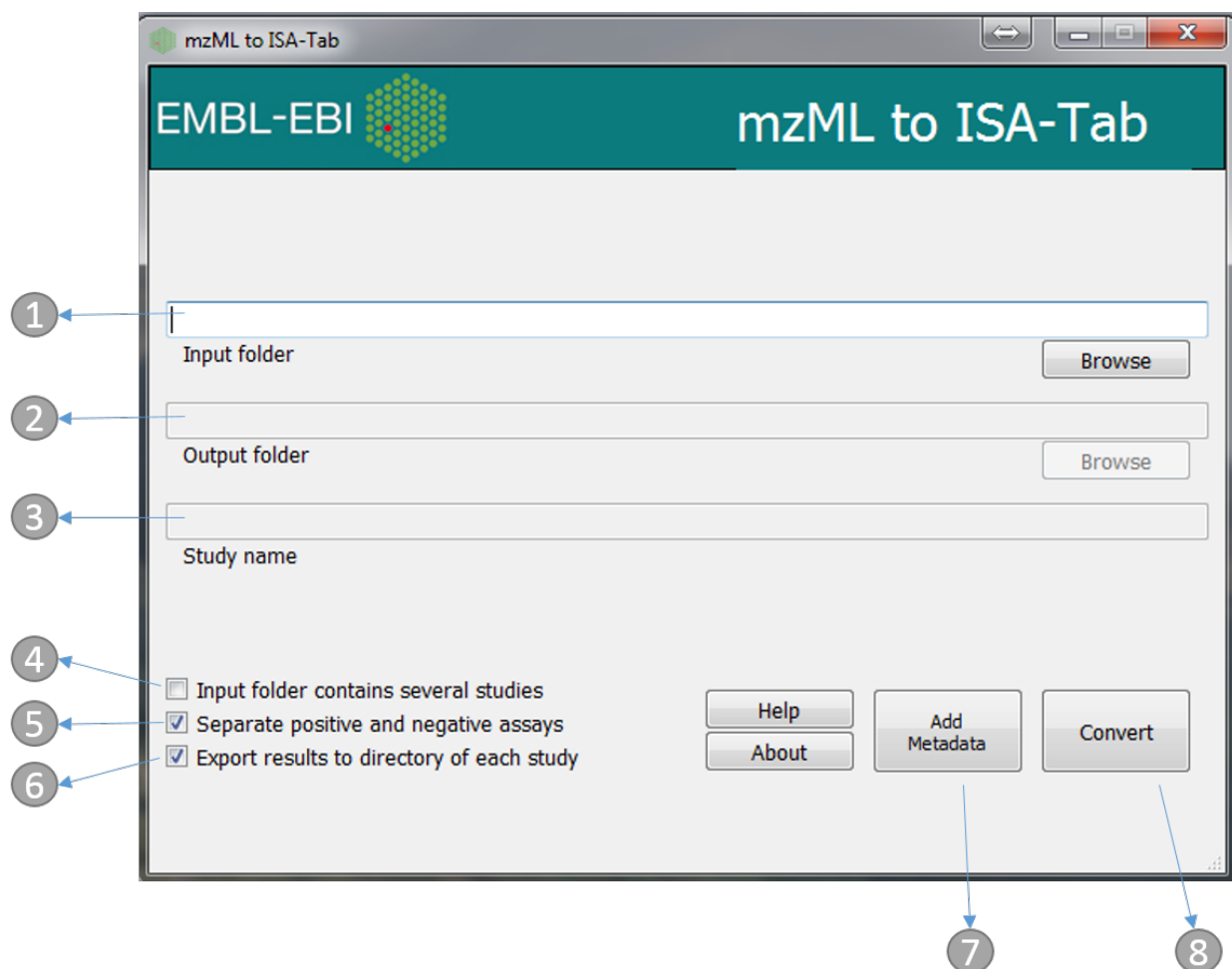
For `mzML` files, `mzml2isa` only requires that you put all your files in the same folder, and parsing should work fine. Note that reference to `RAW` files is extracted from the `mzML` files, so if you plan to create an `ISA` archive after `mzml2isa` creates `ISA` files, don't forget to include those as well.

Example structure:

```
/
home/
  metabolomics/
    MZML study1/      # the name of the folder doesn't matter
      Sample1.mzML    # the name of the file must correspond to the sample name
      Sample2.mzML
      ...
    MZML study2/      # the name of the folder doesn't matter
      Sample1.mzML    # the name of the file must correspond to the sample name
      Sample2.mzML
      ...
```

See the [mzml2isa](#) documentation for more details.

Starting interface



1. Directory containing the mzML files. e.g. 'MZML study1' in the above example
2. Output folder (automatically the same as input)
3. Study name (automatically the same as the input folder name, e.g. 'MZML study1' in the above example)
4. If the 'input folder' contains multiple studies then mzML2ISA will be run each study, e.g. in the above example if 'metabolomics' folder was chosen as the 'input folder' then the parsing will be performed on both studies ('MZML study 1', 'MZML study 2')
5. If selected, separate assay files will be created for positive and negative assays
6. If selected, the output folder will be the same as the input folder and input folder name will be used as the study name
7. The 'Add Metadata' button will open a new dialog box where users can add more metadata
8. Start the conversion process!

Adding additional metadata

Users can add additional metadata manually through the dialog box below. However, all this information can be added at the final stage with ISAcreeator tool.

Additional Metadata

Study Investigation Experiments Material / Methods

Study

Identifier
will use study identifier name from main window

Title *

Submission date: 2000-01-01 Release date: 2000-01-01

Description *

Study Publication

PubMed identifier: e.g. 26496010 DOI: e.g. 10.1016/j.jhep.2013.12.002

Title

Status

Authors

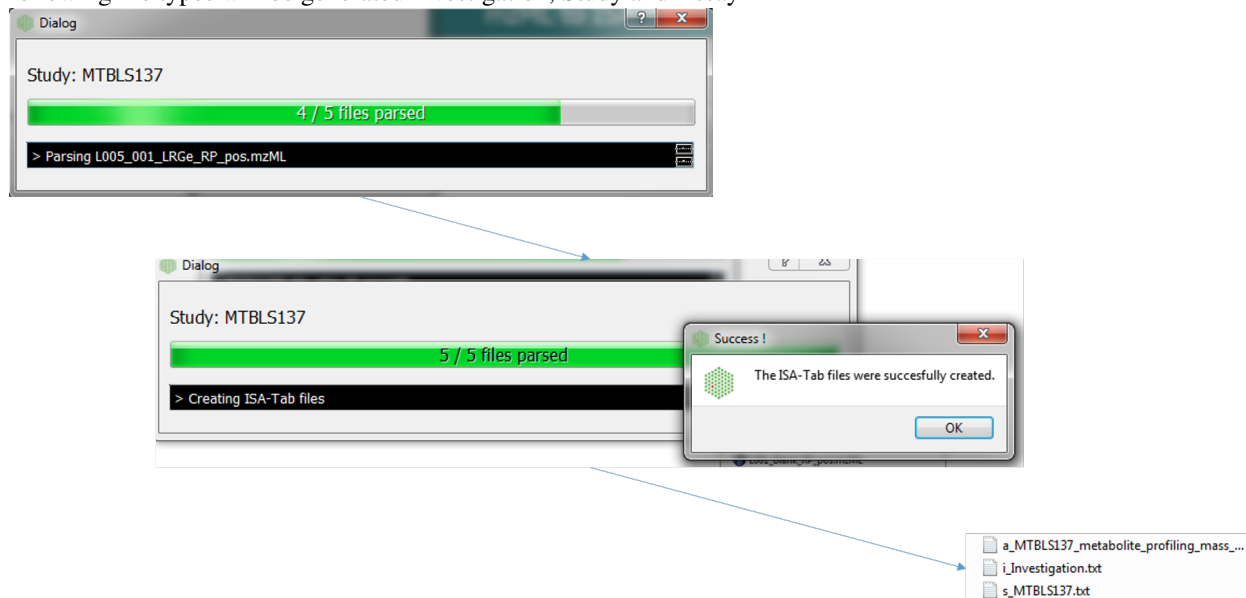
Study Contacts *

*: Fields required by a MetaboLights study

OK Cancel Apply

Conversion

The following dialog boxes after the convert button is pressed. By default this is run in parallel (multiple cores). The following file types will be generated Investigation, Study and Assay



Editing with ISAcreeator

The ISA-Tab structure can be further populated with the [ISAcreeator software](#).

See brief [tutorial](#) for more details.

1.2.2 imzml2isa-qt

imzml2isa-qt is a Graphical User Interface for mzml2isa, that allows converting imzML files to ISA-Tab files with the help of mzml2isa as well as adding more metadata to the generated ISA files through its interface.

Installation

imzml2isa-qt is available on PyPI, so if `pip` is installed on your machine, installing mzml2isa should be quite straightforward, without the need to worry about dependencies. It is also possible to install mzml2isa development version directly from the [imzml2isa-qt GitHub repository](#).

Warning: imzml2isa-qt is a **Python3** program only !

Windows

Windows users can either install imzml2isa-qt as described below or use a ‘ready to go’ Windows executable (**no python install required**). The executables be found on the imzml2isa-qt [Github release section](#). See file `imzml2isa_gui.exe`.

Dependencies

Warning: Without `pip` you’ll have to install the requirements yourself, otherwise running `setup.py` will fail when trying to import **nmrml2isa** for the first time.

Requirements

nmrml2isa the parsing library (see [here](#))

PyQt5 the GUI toolkit used by nmrml2isa-qt (see [here](#) for more information about PyQt)

PyPI (stable version)

With *pip* Just run one of the following commands in a terminal:

```
pip install imzml2isa-qt          # install on the machine, needs super-user rights
pip install imzml2isa-qt --user  # install only for the current user, no rights needed
```


Without *pip* This requires the python `setuptools` module to be installed, as well as the dependencies listed above. Download the latest stable release from the [PyPI repository](#), unpack it and install it by running the following commands:

```
tar xf imzml2isa-qt-xx.yy.zz.tar.gz # replace with whatever version you downloaded
cd imzml2isa-qt-xx.yy.zz.tar.gz
python setup.py install             # will require super-user rights
```

GitHub (development version)

With *pip*

```
pip install git+git://github.com/ISA-tools/imzml2isa-qt
```

Without *pip*

```
git clone https://github.com/ISA-tools/imzml2isa-qt
cd imzml2isa-qt
python setup.py install
```

Usage

Video tutorial

See below for 5 min video for how to use either `mzml2isa-qt`, `imzml2isa-qt` or `nmrml2isa-qt`.

<https://www.youtube.com/watch?v=xy3uusQRkbI>

Folder structure

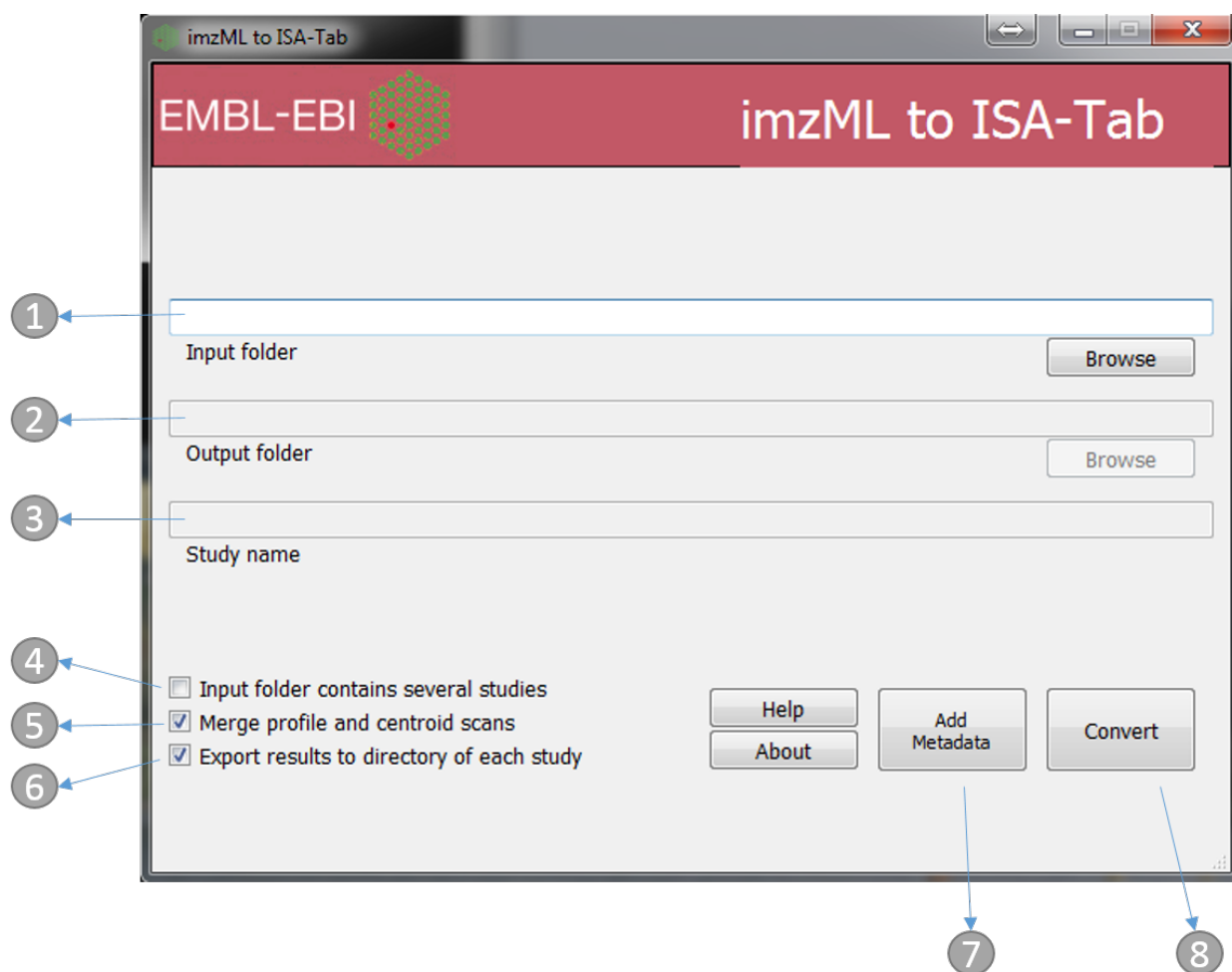
For `imzML` files, you must have your `imzML` files in the same folder, but also the low- and high-res images of your scans (`*.jpg`, `*.ibd`, `*.ndpi`, etc...) named exactly like the `imzML` file of that particular scan (as `mzml2isa` scans the directory of the source file to eventually extract the location of those images):

```
/
home/
  metabolomics/
    imzml_study_1/
      A14 5cm S3.ndpi
      A14 5cm S3.jpg           # Sample name will be
      A14 5cm S3-centroid.ibd  # "A14 5cm S3"
      A14 5cm S3-centroid.imzML # and centroid and profile
      A14 5cm S3-profile.ibd   # scans will be merged.
      A14 5cm S3-profile.imzML
      A14 10cm S3.ndpi
      A14 10cm S3.jpg          # Sample name will be
      A14 10cm S3-centroid.ibd # "A14 10cm S3"
      A14 10cm S3-centroid.imzML # and centroid and profile
      A14 10cm S3-profile.ibd   # scans will be merged
      A14 10cm S3-profile.imzML
      ...
    imzml_study_2/
      B14 5cm S3.ndpi
      B14 5cm S3.jpg
```

```
B14 5cm S3-centroid.ibd
B14 5cm S3-centroid.imzML
```

See the [mzml2isa](#) documentation for more details.

Starting interface



1. Directory containing the mzML files. e.g. 'imzml_study_1' in the above example
2. Output folder (automatically the same as input)
3. Study name (automatically the same as the input folder name, e.g. 'imzml_study_1' in the above example)
4. If the 'input folder' contains multiple studies then mzML2ISA will be run each study, e.g. in the above example if 'metabolomics' folder was chosen as the 'input folder' then the parsing will be performed on both studies ('imzml_study_1', 'imzml_study_2')
5. If selected, when profile and centroided files of the same run are available the two files will be merged and saved under a single row in the assay file
6. If selected, the output folder will be the same as the input folder and input folder name will be used as the study name
7. The 'Add Metadata' button will open a new dialog box where users can add more metadata

8. Start the conversion process!

Adding additional metadata

Users can add additional metadata manually through the dialog box below. However, all this information can be added at the final stage with ISAcceptor tool.

Additional Metadata

Study Investigation Experiments **Material / Methods**

Preparation

Sample mounting Name: e.g. Thermo Scientific X72 SuperFrost Plus Adhesion slide Ref: IRI: o d

Sample preservation Name: e.g. Flash Freezing Ref: e.g. OBI IRI: e.g. http://purl.obolibrary.org/obo/OBI_0001952 o d

Sectioning Instrument Name: e.g. Thermo Scientific Shandon Cryotome Ref: IRI: o d

Section thickness Value: IRI:

Imaging

Spatial Resolution Value: IRI:

Stain

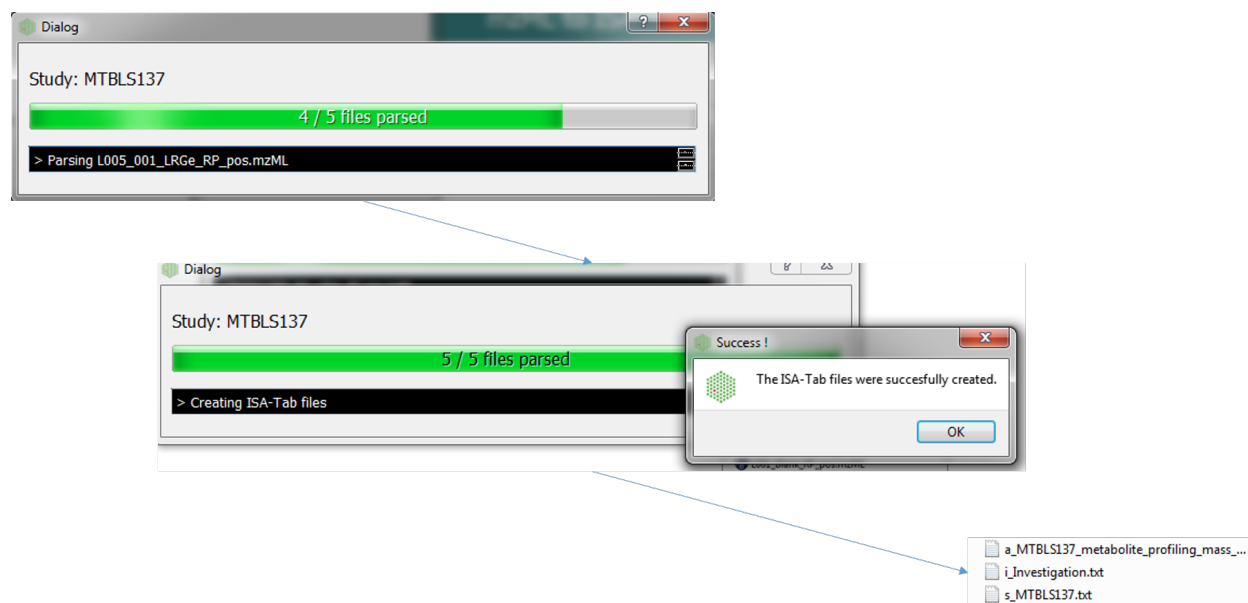
Stain Name: e.g. Night blue Ref: e.g. CHEBI IRI: e.g. http://purl.obolibrary.org/obo/CHEBI_87645 o d

*: Fields required by a MetaboLights study

OK Cancel Apply

Conversion

The following dialog boxes after the convert button is pressed. By default this is run in parallel (multiple cores). The following file types will be generated Investigation, Study and Assay



Editing with ISAcceptor

The ISA-Tab structure can be further populated with the [ISAcceptor](#) software.

See brief [tutorial](#) for more details.

1.2.3 nmrml2isa-qt

nmrml2isa-qt is a Graphical User Interface for nmrml2isa, that allows converting mzML files to ISA-Tab files with the help of nmrml2isa as well as adding more metadata to the generated ISA files through its interface.

Installation

nmrml2isa-qt is available on PyPI, so if `pip` is installed on your machine, installing nmrml2isa should be quite straightforward, without the need to worry about dependencies. It is also possible to install nmrml2isa development version directly from the [nmrml2isa-qt GitHub repository](#).

Warning: nmrml2isa-qt is a **Python3** program only !

Windows

Windows users can either install nmrml2isa-qt as described below or use a 'ready to go' Windows executable (**no python install required**). The executables be found on the nmrml2isa-qt [Github release section](#). See file `nmrml2isa_gui.exe`.

Dependencies

Warning: Without `pip` you'll have to install the requirements yourself, otherwise running `setup.py` will fail when trying to import **nmrml2isa** for the first time.

Requirements

nmrml2isa the parsing library (see [here](#))

PyQt5 the GUI toolkit used by `nmrml2isa-qt` (see [here](#) for more information about PyQt)

PyPI (stable version)

With `pip` Just run one of the following commands in a terminal:

```
pip install nmrml2isa-qt          # install in /usr, needs super-user rights
pip install nmrml2isa-qt --user  # install in ~/.local
```

Without `pip` This requires the python `setuptools` module to be installed, as well as the dependencies listed above. Download the latest stable release from the [PyPI repository](#), unpack it and install it by running the following commands:

```
tar xf nmrml2isa-qt-xx.yy.zz.tar.gz # replace with whatever version you downloaded
cd nmrml2isa-qt-xx.yy.zz.tar.gz
python setup.py install              # will require super-user rights
```

GitHub (development version)

With `pip`

```
pip install git+git://github.com/ISA-tools/nmrml2isa-qt
```

Without `pip`

```
git clone https://github.com/ISA-tools/nmrml2isa-qt
cd nmrml2isa-qt
python setup.py install          # will require super-user rights
```

Usage

Video tutorial

See below for 5 min video for how to use either `mzml2isa-qt`, `imzml2isa-qt` or `nmrml2isa-qt`.

<https://www.youtube.com/watch?v=xy3uusQRkbl>

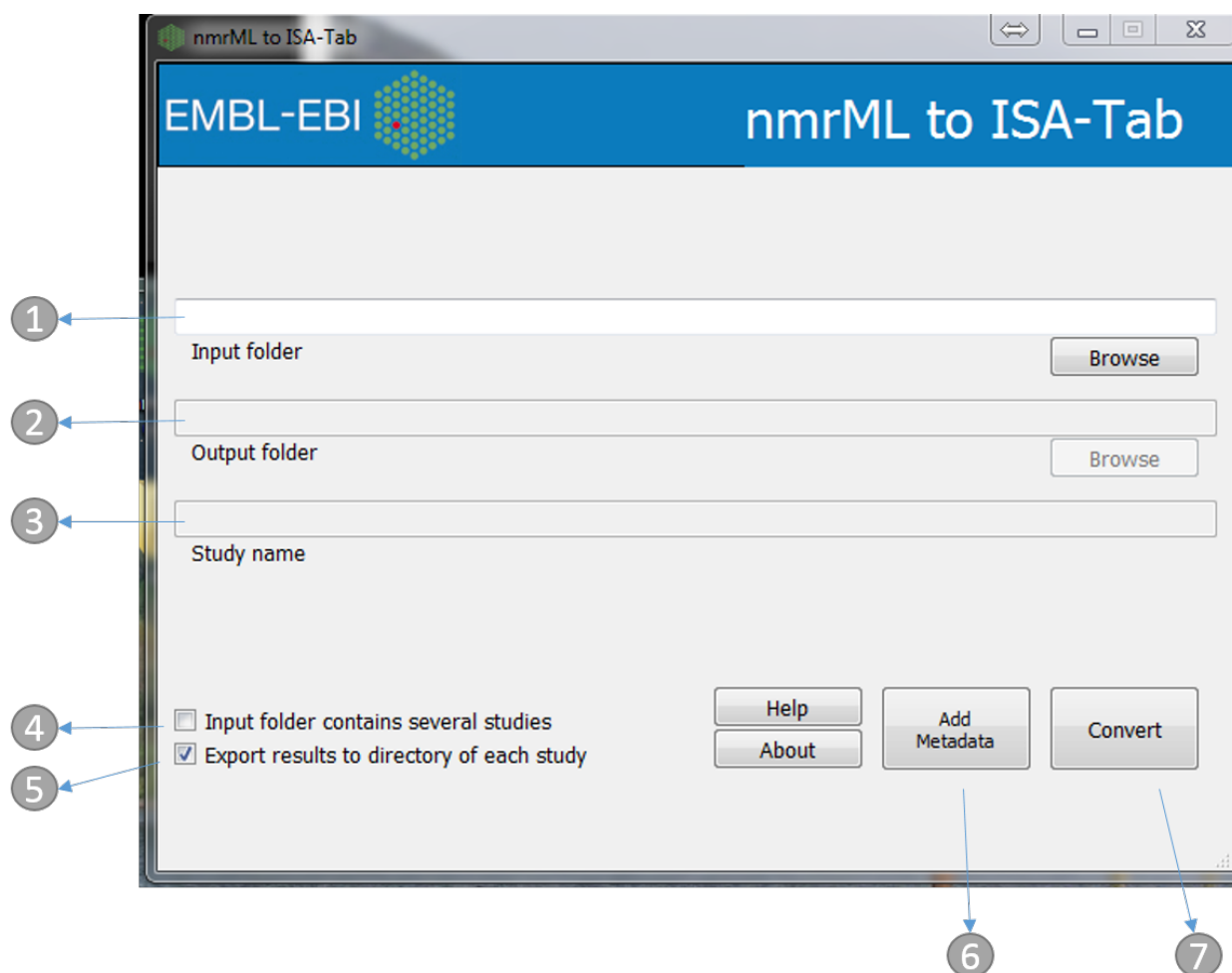
Folder structure

nrmml2isa only requires that you put all you nmrML and zipped raw files in the same folder, and parsing should work fine. Note that reference to RAW files is extracted from the mzML files, so if you plan to create an ISA archive after mzml2isa creates ISA files, don't forget to include those as well.

Example structure:

```
/
home/
  metabolomics/
    nmrML_study1/      # the name of the folder doesn't matter
      Sample1.nmrML    # the name of the file must correspond to the sample name
      Sample2.zip      # the raw files should be zipped and called exactly like the nmrML
      Sample2.nmrML
      Sample2.zip
      ...
    nmrML_study2/
      Sample1.nmrML
      Sample2.zip
      Sample2.nmrML
      Sample2.zip
      ...
```

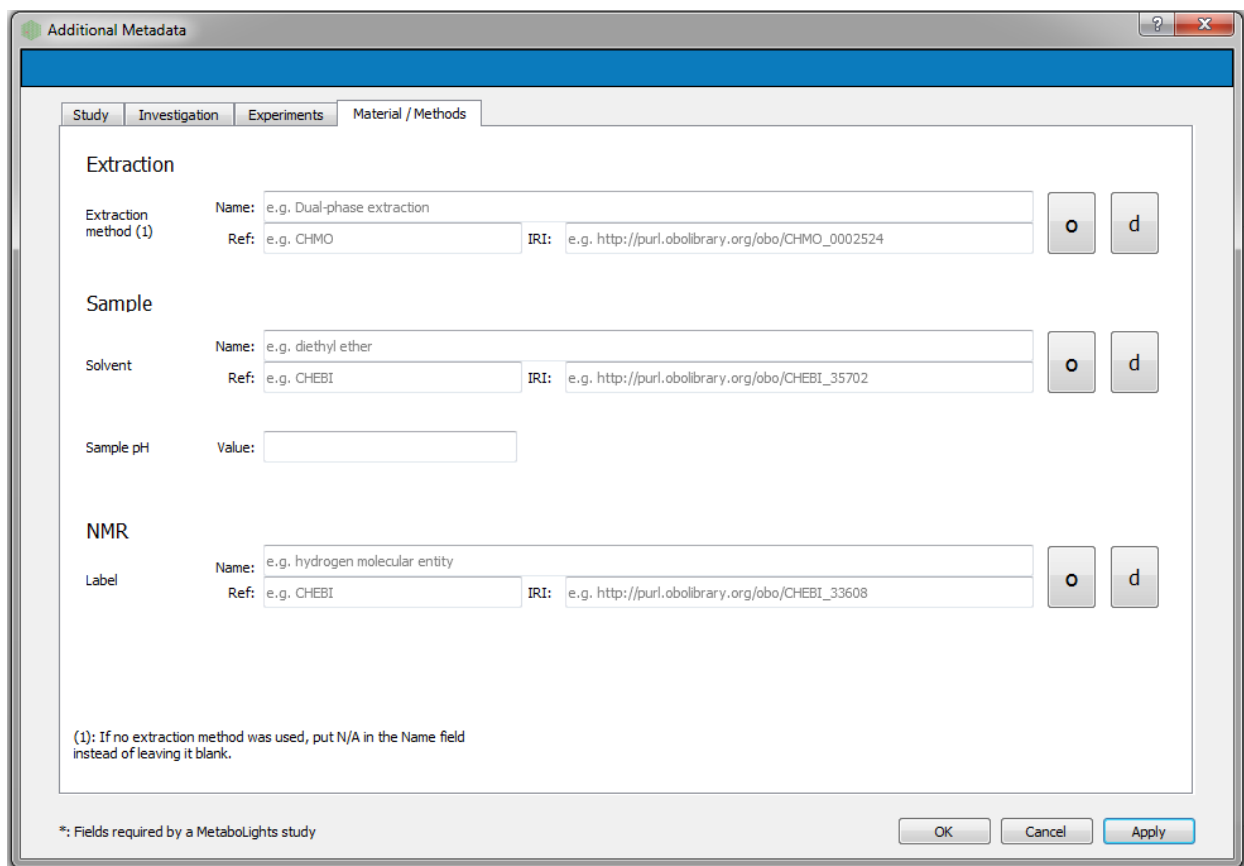
Starting interface



1. Directory containing the mzML files. e.g. 'nmrML_study1' in the above example
2. Output folder (automatically the same as input)
3. Study name (automatically the same as the input folder name, e.g. 'nmrML_study1' in the above example)
4. If the 'input folder' contains multiple studies then mzML2ISA will be run each study, e.g. in the above example if 'metabolomics' folder was chosen as the 'input folder' then the parsing will be performed on both studies ('nmrML_study 1', 'nmrML_study 2')
5. If selected, separate assay files will be created for positive and negative assays
6. If selected, the output folder will be the same as the input folder and input folder name will be used as the study name
7. The 'Add Metadata' button will open a new dialog box where users can add more metadata
8. Start the conversion process!

Adding additional metadata

Users can add additional metadata manually through the dialog box below. However, all this information can be added at the final stage with ISAcceptor tool.



The 'Additional Metadata' dialog box is shown with the 'Material / Methods' tab selected. It contains three main sections: 'Extraction', 'Sample', and 'NMR'. Each section has a 'Name' field, a 'Ref' field, and an 'IRI' field, with 'o' and 'd' buttons to the right. The 'Extraction' section has a 'Value' field for 'Sample pH'. A note at the bottom states: '(1): If no extraction method was used, put N/A in the Name field instead of leaving it blank.' The bottom of the dialog has 'OK', 'Cancel', and 'Apply' buttons.

Extraction

Extraction method (1) Name: e.g. Dual-phase extraction Ref: e.g. CHMO IRI: e.g. http://purl.obolibrary.org/obo/CHMO_0002524 o d

Sample

Solvent Name: e.g. diethyl ether Ref: e.g. CHEBI IRI: e.g. http://purl.obolibrary.org/obo/CHEBI_35702 o d

Sample pH Value:

NMR

Label Name: e.g. hydrogen molecular entity Ref: e.g. CHEBI IRI: e.g. http://purl.obolibrary.org/obo/CHEBI_33608 o d

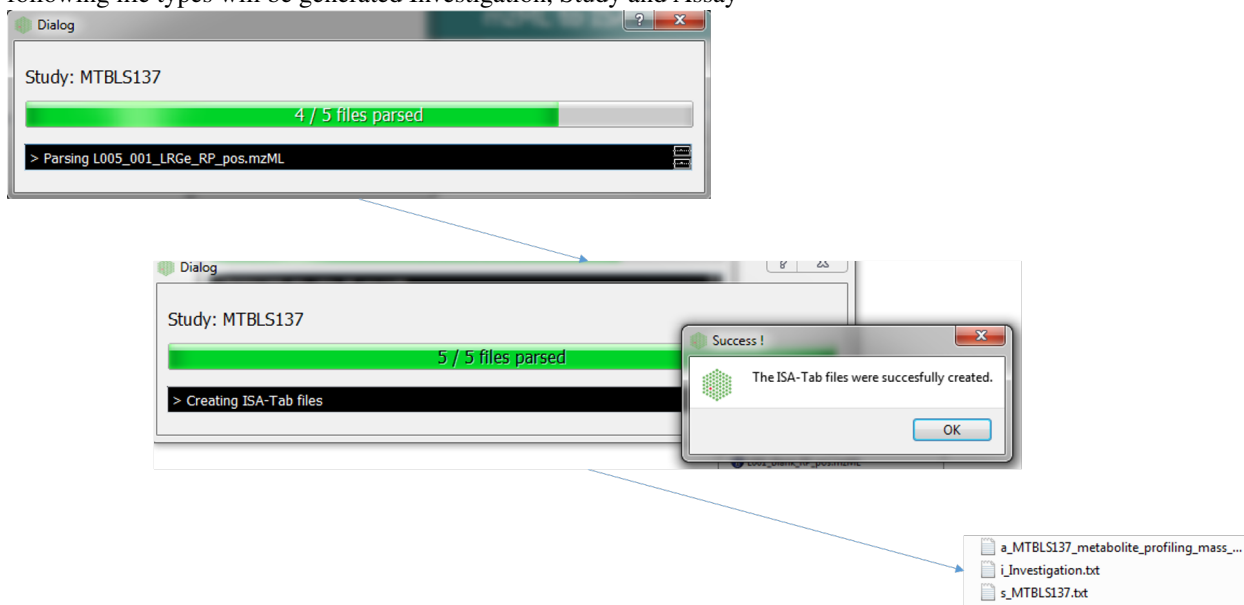
(1): If no extraction method was used, put N/A in the Name field instead of leaving it blank.

*: Fields required by a MetaboLights study

OK Cancel Apply

Conversion

The following dialog boxes after the convert button is pressed. By default this is run in parallel (multiple cores). The following file types will be generated Investigation, Study and Assay



Editing with ISAcreeator

The ISA-Tab structure can be further populated with the [ISAcreeator software](#).

See brief [tutorial](#) for more details.

1.3 Galaxy interfaces

The Galaxy interfaces are divided into `mzml2isa-galaxy` (covering `mzML` and `imzML`) and `nmrml2isa-galaxy` (covering `nmrML`)

1.3.1 mzml2isa-galaxy

`mzml2isa-galaxy` is a Galaxy wrapper for the `nmrml2isa` python package tool.

[Galaxy repository](#) is a web based workflow platform that can be used to perform bioinformatics in a reproducible and sharable environment.

Installation

The recommended installation is via the [toolshed repository](#). Dependencies should be installed automatically when using Galaxy version `>= 16.10`.

The dependencies are dealt with Bioconda. To ensure that Bioconda is working check to make sure the following settings are in the `config/galaxy.ini` file:

```
# dependencies before each job runs.
conda_auto_install = True
# Set to True to instruct Galaxy to install Conda from the web automatically
# if it cannot find a local copy and conda_exec is not configured.
conda_auto_init = True
```

Usage

Folder structure

For `mzML` files, `mzml2isa` only requires that you put all your files in the same folder, and parsing should work fine. Note that reference to RAW files is extracted from the `mzML` files, so if you plan to create an ISA archive after `mzml2isa` creates ISA files, don't forget to include those as well.

Example structure:

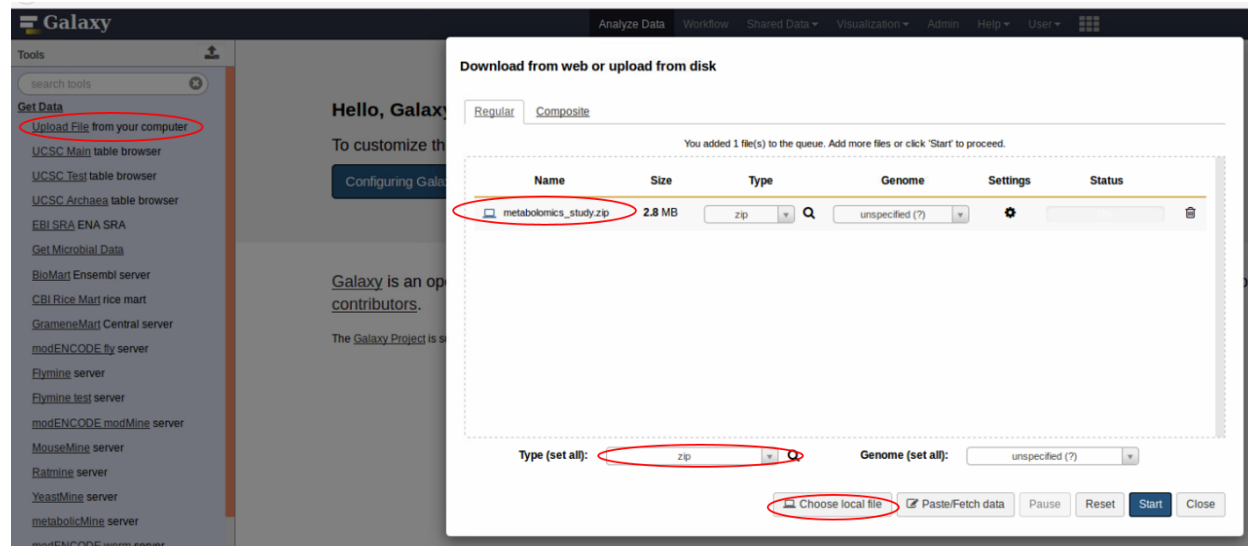
```
/
home/
  metabolomics/
    MZML study1/      # the name of the folder doesn't matter
      Sample1.mzML    # the name of the file must correspond to the sample name
      Sample2.mzML
      ...
    MZML study2/      # the name of the folder doesn't matter
      Sample1.mzML    # the name of the file must correspond to the sample name
      Sample2.mzML
      ...
```

See the [mzml2isa](#) documentation for more details.

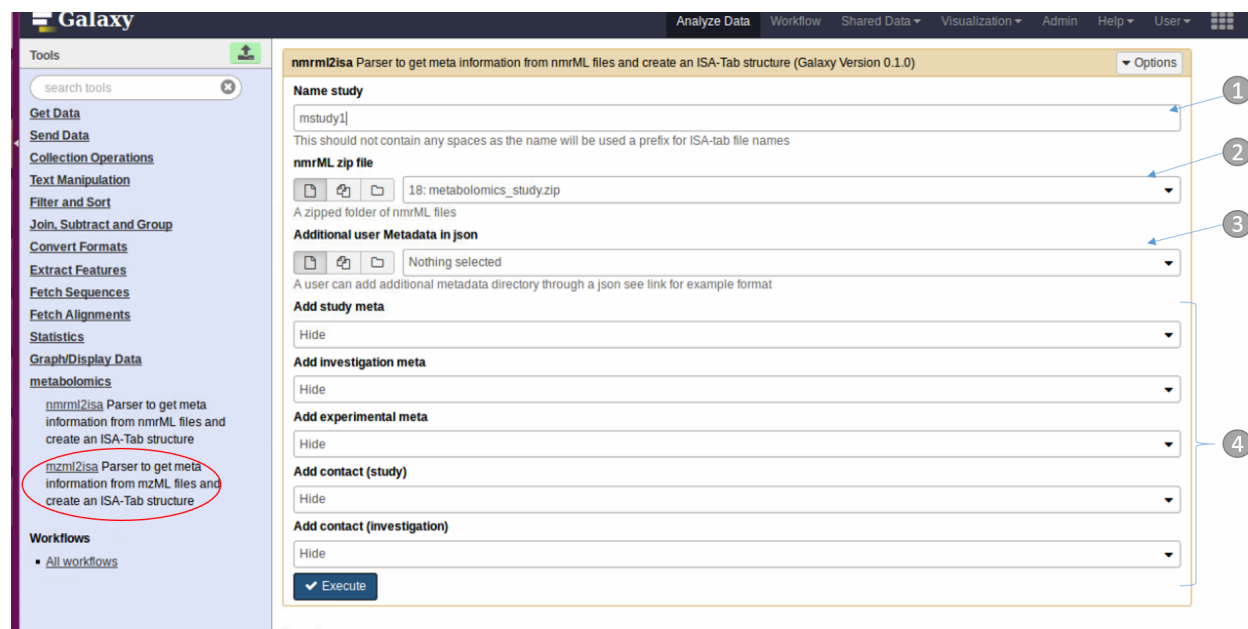
The folders need to be zipped before adding to galaxy e.g. MZML study1.zip. The zipped folders are then added 1 at a time through the GetData interface of Galaxy.

Getting data into Galaxy

Using the GetData tab, add the zipped metabolomics study into galaxy. Remember to use the 'zip' file type



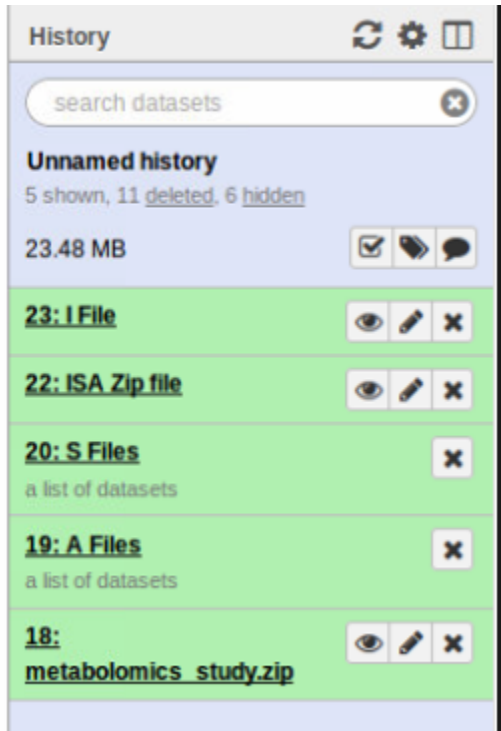
Running nmrml2isa



1. Study name
2. zipped folder containing the nmrML files
3. For advanced users only. This allows any additional metadata to be added in json format see [json format](#)

- Optional simple additional metadata can be added manually through the dropdown tabs

Output



The output consists of the following:

- I File: Investigation file
- S Files: Dataset collection containing 1 or more study files
- A Files: Dataset collection containing 1 or more assay files
- ISA zip file: A zip file containing all of the outputs

Editing with ISAcreeator

The ISA-Tab structure can be further populated with the [ISAcreeator software](#).

See brief [tutorial](#) for more details.

1.3.2 nmrml2isa-galaxy

nmrml2isa-galaxy is a Galaxy wrapper for the nmrml2isa python package tool.

[Galaxy repository](#) is a web based workflow platform that can be used to perform bioinformatics in a reproducible and sharable environment.

Installation

The recommended installation is via the [toolshed repository](#). Dependencies should be installed automatically when using Galaxy version ≥ 16.10 .

The dependencies are dealt with Bioconda. To ensure that Bioconda is working check to make sure the following settings are in the config/galaxy.ini file:

```
# dependencies before each job runs.
conda_auto_install = True
# Set to True to instruct Galaxy to install Conda from the web automatically
# if it cannot find a local copy and conda_exec is not configured.
conda_auto_init = True
```

Usage

Folder structure

nmrml2isa only requires that you put all you nmrML and zipped raw files in the same folder, and parsing should work fine. Note that reference to RAW files is extracted from the mzML files, so if you plan to create an ISA archive after mzml2isa creates ISA files, don't forget to include those as well.

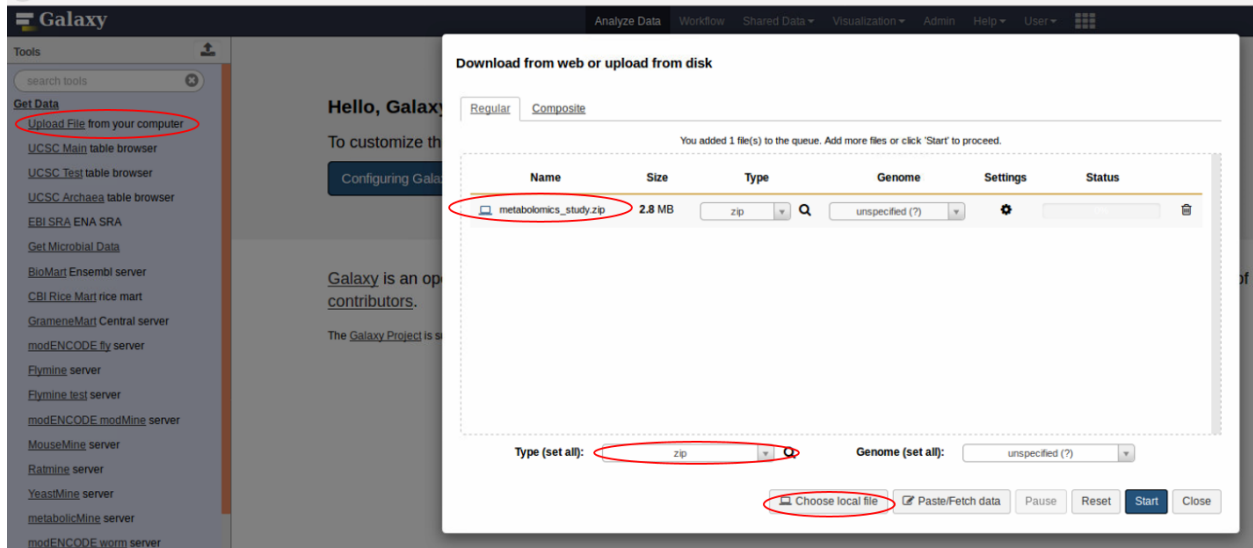
Example structure:

```
/
home/
  metabolomics/
    nmrML_study1/      # the name of the folder doesn't matter
      Sample1.nmrML    # the name of the file must correspond to the sample name
      Sample2.zip      # the raw files should be zipped and called exactly like the nmrML
      Sample2.nmrML
      Sample2.zip
      ...
    nmrML_study2/
      Sample1.nmrML
      Sample2.zip
      Sample2.nmrML
      Sample2.zip
      ...
```

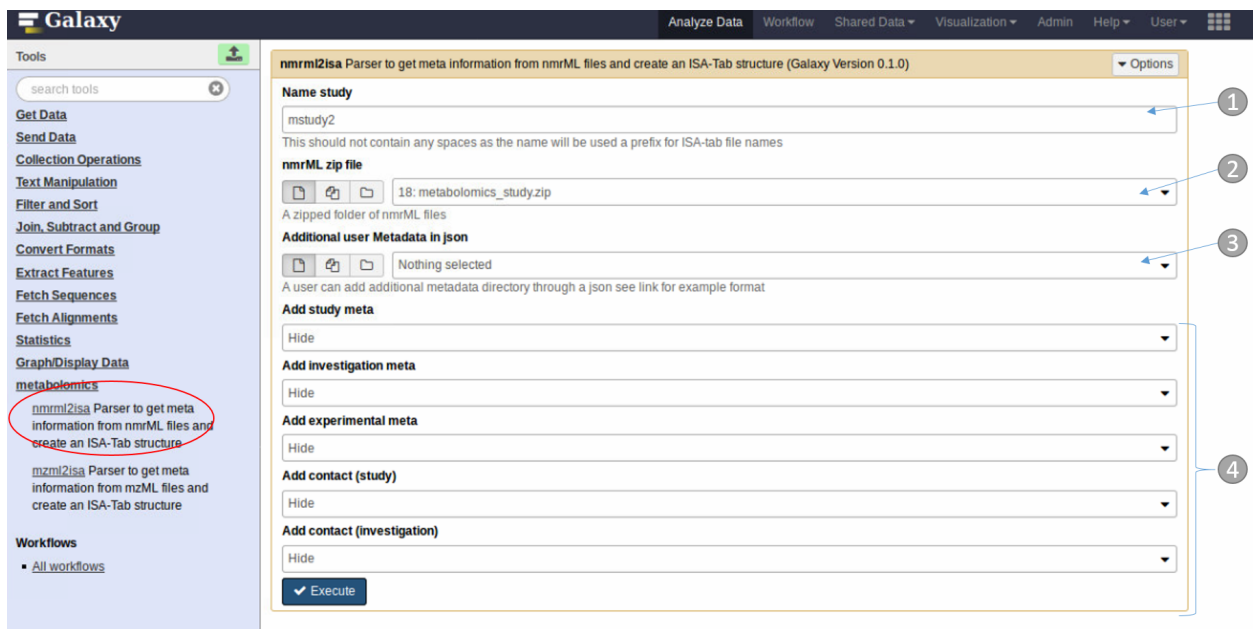
The folders need to be zipped before adding to galaxy e.g. nmrML_study1.zip. The zipped folders are then added 1 at a time through the GetData interface of Galaxy.

Getting data into Galaxy

Using the GetData tab, add the zipped metabolomics study into galaxy. Remember to use the 'zip' file type

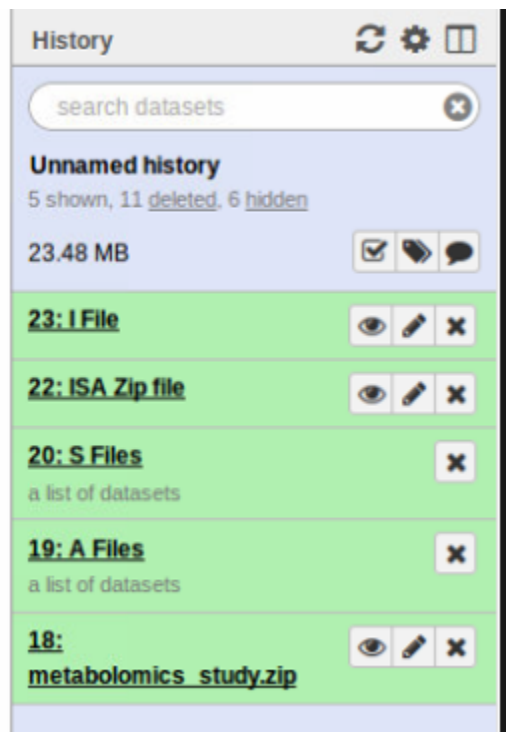


Running nmrml2isa



1. Study name
2. zipped folder containing the nmrML files
3. For advanced users only. This allows any additional metadata to be added in json format
4. Optional simple additional metadata can be added manually through the dropdown tabs

Output



The output consists of the following:

- I File: Investigation file
- S Files: Dataset collection containing 1 or more study files
- A Files: Dataset collection containing 1 or more assay files
- ISA zip file: A zip file containing all of the outputs

Editing with ISAcreeator

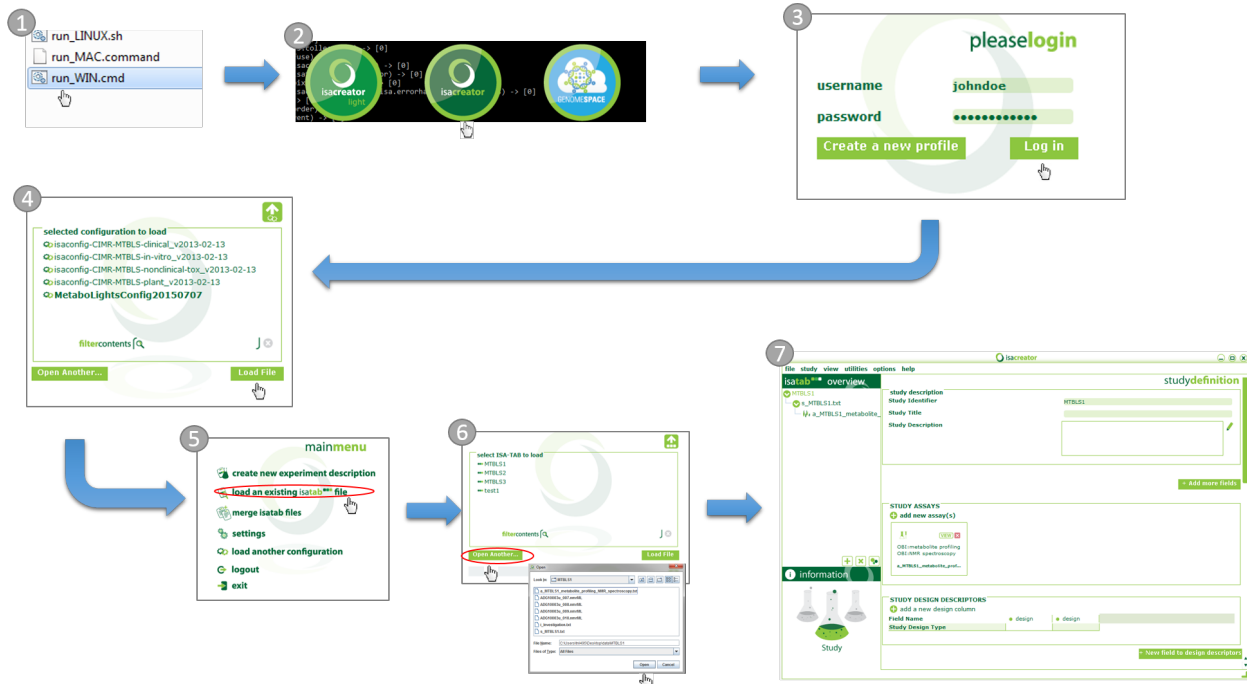
The ISA-Tab structure can be further populated with the [ISAcreeator software](#).

See brief [tutorial](#) for more details.

1.4 Other

1.4.1 Other

Editing with ISAcceptor



The ISA-Tab structure generated from the 2ISA tools can be further populated with the **ISAcceptor** software.

The workflow being as follows:

1. Open the ISAcceptor software (available for Mac, Linux and Windows)
2. Select the ISAcceptor button
3. Login (create a profile if the first time being used)
4. Choose a configuration that is suited for the study
5. Choose the 'Load and existing isatab file' option
6. Click on the 'open another' button and find the folder containing the newly created ISA-tab files
7. Fill all remaining fields with ISAcceptor

A video of the standard Metabolights procedure can be found on [youtube](#).

Metadata in JSON format

Warning: *For advanced use and developers only*

Metadata extraction in JSON format

All metadata extracted from the XML files can be saved in a JSON format. The format is easily interchangeable with python dictionaries can also be easily ‘piped’ to other bioinformatic pipelines regardless of programming language used.

Adding experimental metadata via JSON format

Additional metadata not found in the XML files can be added in a JSON format e.g. organism studied, chromatography used, contact details etc.

This allows existing bioinformatic/computational workflows, which may already have collected some additional meta-data, to automatically populate additional entries in the ISA-Tab files.

Example format:

```
{
  "characteristics": {
    "organism": {
      "name": "",
      "accession": "",
      "ref": ""
    },
    "organism_variant": {
      "name": "",
      "accession": "",
      "ref": ""
    },
    "organism_part": {
      "name": "",
      "accession": "",
      "ref": ""
    }
  }
  ....
}
```

See the following link for full [example of JSON used for mzml2isa](#).

Example templates for the metadata JSON can be extracted from either mzml2isa-qt, imzml2isa-qt or nmrml2isa-qt.

Run either of the following:

```
python3 -m mzml2isa_qt.usermeta
python3 -m imzml2isa_qt.usermeta
python3 -m nmrml2isa_qt.usermeta
```

Add any metadata through the GUI and then click apply. The terminal will then spit out the JSON text.

Adding additional metadata via JSON for CLI:

For the CLI simply use the `-m` option and direct to the json file.

```
mzml2isa -i /path/to/mzml/folder -o /path/to/out_folder -s STUDYID -m metadata.json
nmrml2isa -i /path/to/mzml/folder -o /path/to/out_folder -s STUDYID -m metadata.json
```

Adding additional metadata via JSON for API:

For the API the `usermeta` parameter can be used to pass the JSON metadata as a python dictionary to the `ISA_Tab` class. See the following API documentation for class `mzml2isa.isa.ISA_Tab` and class `nmrml2isa.isa.ISA_Tab`

Adding additional metadata via JSON for Galaxy:

This metadata can be added manually via the dropdown options or via a prepared JSON file using the `Additional user metadata in json` option.

Adding additional metadata via JSON for GUI:

Not possible. This metadata is added directly via the GUI using the `usermeta` dialog.

2.1 Contacts

2.1.1 Martin Larralde

Co-author of **mzml2isa**, author of **nmrml2isa**
Ecole Normale Supérieure de Cachan, France
martin.larralde@ens-cachan.fr

2.1.2 Tom Lawson

Co-author of **mzml2isa**
University of Birmingham, United Kingdom
tnl495@bham.ac.uk

2.1.3 Reza Salek

Scientific coordinator of **mzml2isa** and **nmrml2isa**
European Bioinformatics Institute, United Kingdom
reza.salek@ebi.ac.uk

Symbols

1r Data File, [23](#)
 1r Data Format, [23](#)
 1r Data Type, [23](#)

A

Acquisition Nucleus, [24](#)
 Acquisition Parameter Data File, [24](#)
 Acquisition Parameter Data Format, [24](#)

B

Baseline Correction Method, [24](#)
 Binary file checksum type, [12](#)
 build_env() (mzml2isa.mzml.mzMLmeta method), [16](#)

C

Calibration Reference Shift, [24](#)
 create_assay() (mzml2isa.isa.ISA_Tab method), [18](#)
 create_investigation() (mzml2isa.isa.ISA_Tab method), [18](#)
 create_study() (mzml2isa.isa.ISA_Tab method), [18](#)
 cvParam_loop() (mzml2isa.mzml.mzMLmeta method), [16](#)

D

Data file checksum type, [8](#)
 Data file content, [8](#)
 Data Transformation Name, [8](#), [24](#)
 Data Transformation software, [8](#), [25](#)
 Data Transformation software version, [8](#), [25](#)
 data_file_content() (mzml2isa.mzml.mzMLmeta method), [16](#)
 Decoupling Nucleus, [25](#)
 Derived Spectral Data File, [9](#), [25](#)
 derived() (mzml2isa.mzml.mzMLmeta method), [16](#)
 Detector, [9](#)
 Detector mode, [12](#)

E

env (mzMLmeta attribute), [16](#)

extract_meta() (mzml2isa.mzml.mzMLmeta method), [16](#)

F

First Order Phase Correction, [25](#)
 Free Induction Decay Data File, [25](#)
 Free Induction Decay Data Format, [26](#)

H

High-res image, [12](#)

I

Ibd binary type, [12](#)
 Inlet type, [9](#)
 Instrument, [9](#), [26](#)
 Instrument manufacturer, [9](#), [26](#)
 Instrument serial number, [9](#)
 Instrument software, [9](#), [26](#)
 Instrument software version, [10](#), [26](#)
 Ion source, [10](#)
 Irradiation Frequency, [26](#)
 isa_env (ISA_Tab attribute), [18](#)
 ISA_Tab (class in mzml2isa.isa), [18](#)
 ISA_Tab (class in nmrm2isa.isa), [31](#)

L

Line scan direction, [12](#)
 Linescan sequence, [13](#)

M

Magnetic field strength, [26](#)
 make_assay_template() (mzml2isa.isa.ISA_Tab method), [18](#)
 Mass analyzer, [10](#)
 Max count of pixel x, [13](#)
 Max count of pixel y, [13](#)
 Max dimension x, [13](#)
 Max dimension y, [13](#)
 merge_entries() (mzml2isa.mzml.mzMLmeta method), [17](#)
 meta (mzMLmeta attribute), [16](#)

meta_isa (mzml2isa.mzml.mzMLmeta attribute), 17
meta_json (mzml2isa.mzml.mzMLmeta attribute), 17
MS Assay Name, 10
mzMLmeta (class in mzml2isa.mzml), 16
mzrange() (mzml2isa.mzml.mzMLmeta method), 17

N

Native spectrum identifier format, 10
NMR Assay Name, 27
NMR Probe, 27
NMR tube type, 27
nmrMLmeta (class in nmrm12isa.nmrm1), 31
ns (mzMLmeta attribute), 16
Number of data points, 27
Number of scans, 10
Number of steady state scans, 28
Number of transients, 28

O

obo (mzMLmeta attribute), 16

P

Pixel size x, 13
Pixel size y, 14
polarity() (mzml2isa.mzml.mzMLmeta method), 17
Processing Parameter Data File, 28
Processing Parameter Data Format, 28
Pulse sequence, 29
Pulse Sequence Data File, 28
Pulse Sequence Data Format, 28
Pulse Width, 28

R

Raw data file format, 11
Raw Spectral Data File, 11
Relaxation Delay, 29

S

Sample Name, 11, 29
Sampling Strategy, 29
Scan m/z range, 11
Scan pattern, 14
Scan polarity, 11
Scan type, 14
scan_num() (mzml2isa.mzml.mzMLmeta method), 17
software() (mzml2isa.mzml.mzMLmeta method), 17
Solvent, 14
Solvent flowrate, 15
Spectral Denoising Method, 29
Spectrum representation, 15
spectrum_meta() (mzml2isa.mzml.mzMLmeta method), 17
Spinning Rate, 29

Study_contacts, 30
Sweep Width, 30

T

Target material, 15
Temperature, 30
Time range, 11
timerange() (mzml2isa.mzml.mzMLmeta method), 17
tree (mzMLmeta attribute), 16

U

Universally unique identifier, 15
unparameter() (mzml2isa.isa.ISA_Tab static method), 18
urlize() (mzml2isa.mzml.mzMLmeta method), 17
usermeta (ISA_Tab attribute), 18

W

Window Function Method, 30
write() (mzml2isa.isa.ISA_Tab method), 19

X

X axis range, 30

Y

Y axis type, 31

Z

Zero Value Phase Correction, 31