NOMAD is based on a *bottom-up* approach to data management. Instead of only supporting data in a specific
predefined format, we process files to extract data from an extendable variety of data formats.
Converting heterogenous files into homogeneous machine actionable processed data is the
basis to make data FAIR. It allows us to build search interfaces, APIs, visualization, and analysis tools independent from specific file formats.

!datamodel (images/datamodel.png)

NOMAD's datamodel and processing

Uploads

Users create **uploads** to organize files. Think of an upload like a project:
many files can be put into a single upload and an upload can be structured with directories.
You can collaborate on uploads, share uploads, and publish uploads. The files in an upload are called **raw files**.
Raw files are managed by users and they are never changed by NOMAD.

!!! note

As a *rule*, **raw files** are not changed during processing (or otherwise). However,

to achieve certain functionality, a parser, normalizer, or schema developer might decide to bend

this rule. Use-cases include the generation of more mainfiles (and entries) and updating

of related mainfiles to automatize ELNs, or

generating additional files to convert a mainfile into a standardized format like nexus or cif.

Files

We already said that all uploaded files are **raw files**. Recognized files that have an entry are called **mainfiles**. Only the mainfile of the entry is

passed to the parser during processing. However, a parser can call other tools or read other files.

Therefore, we consider all files in the same directory of the mainfile as **auxillary files**,

even though there is not necessarily a formal relationship with the entry. If

formal relationships with aux files are established, e.g. via a reference to the file within the processed data, is up to the parser.

Entries

All uploaded **raw files** are analysed to find files with a recognized format. Each file that follows a recognized format is a **mainfile**. For each mainfile, NOMAD will create a database **entry**. The entry is eternally matched to the mainfile. The entry id, for example,

is a hash over the upload id and the mainfile path (and an optional key) within the upload.

This **matching** process is automatic, and users cannot create entries

manually.

!!! note

We say that raw files are not changed by NOMAD and that users cannot create entries,

but what about ELNs? There is a *create entry* button in the UI?

However,

NOMAD will simply create an editable **mainfile** that indirectly creates an entry.

The user might use NOMAD as an editor to change the file, but the content is

determined by the users. Contrary to the processed data that is created from raw files by NOMAD.

Datasets

Users can build collections of entries to form **datasets**. You can imagine datasets like tags or albums in other systems. Each entry can be contain in many datasets and a dataset can hold many entries. Datasets can also overlap. Datasets are only indirectly related to files. The main purpose of **datasets** in NOMAD is to have citable collections of data. Users can get a DOI for their datasets. Datasets have no influence on the processing of data.

Processing

The processing of entries is automatic. Initially and on each mainfile change, the entry corresponding to the mainfile, will be processed. Processing consist of **parsing**, **normalizing**, and **persisting** the created data, as explained in more detail in the Processing section (processing.md).

Parsing

Parsers are small programs that transform data from a recognized *mainfile* into a structured machine processable tree of data that we call the *archive* or **processed data** (data.md)
of the entry. Only one parser is used for each entry. The used parser is determined during matching and depends on the file format. A dedicated guide (../howto/plugins/parsers.md#match-your-raw-file) shows how to match a specific file from your parser. Parsers can be added to NOMAD as plugins (../howto/plugins/parsers.md); this is a list of all built-in parsers (../reference/parsers.md).

!!! note

A special case is the parsing of NOMAD archive files. Usually a parser converts a file

from a source format into NOMAD's *archive* format for processed data. But users can

also create files following this format themselves. They can be uploaded either as `.json` or `.yaml` files

by using the `.archive.json` or `.archive.yaml` extension. In these cases, we also considering

these files as mainfiles and they are also going to be processed. Here the parsing

is a simple syntax check and basically just copying the data, but normalization might

still modify and augment the data substantially. One use-case for these *archive* files,

are ELNs. Here the NOMAD UI acts as an editor for a respective `.json` file, but on each save, the

corresponding file is going through all the regular processing steps. This allows

ELN schema developers to add all kinds of functionality such as updating referenced

entries, parsing linked files, or creating new entries for automation.

Normalizing

While parsing converts a mainfile into processed data, normalizing is only working on the

processed data. Learn more about why to normalize in the documentation on structured data (./data.md).

There are two principle ways to implement normalization in NOMAD:

normalizers and **normalize** functions.

Normalizers (../howto/plugins/normalizers.md) are small programs that take processed data as input.

There is a list of normalizers registered in the NOMAD configuration (../reference/config.md#normalize).

In the future, normalizers might be
added as plugins as well. They run in the configured order. Every normalizer is run on all entries and the normalizer might decide to do something or not, depending on what
it sees in the processed data.

Normalize functions are special functions implemented as part of section definitions in Python schemas (../howto/plugins/schema_packages.md#schema-packages-python-vs-yaml).

There is a special normalizer that will go through all processed data and execute these function if they are defined. Normalize functions get the respective section instance as input. This allows schema plugin (../howto/plugins/schema_packages.md) developers to add normalizing to their sections.

Read about our structured data (./data.md) to learn more about the different sections.

Storing and indexing

As a last technical step, the processed data is stored and some information is passed into the search index. The store for processed data is internal to NOMAD and processed data cannot be accessed directly and only via the archive API (../howto/programmatic/api.md#access-processed-data-archives) or ArchiveQuery (../howto/programmatic/archive_query.md) Python library functionality.

What information is stored in the search index is determined
by the *metadata* and *results* sections and cannot be changed by users or plugins. However, all scalar values in the processed data are also index as key-values pairs.

!!! warning "Attention"
    This part of the documentation should be more substantiated. There will be a learn section
    about the search soon.