MatInf Documentation

Here is the main entry point to documentation for the MatInf project (Research Data Management in Materials Science: https://matinf.pro). Documentation contains multiple sections (every section is a dedicated document/link).

Source Code

For those who reasonably suggest that code is the best documentation, the source code is available at https://gitlab.ruhr-uni-bochum.de/vic/infproject.

Presentations

Some MatInf presentations from MDI group meetings:
https://mdi.matinf.pro/rubric/inf-presentations

Many presentations are available publicly in a CRC247 tenant:
https://crc247.mdi.ruhr-uni-bochum.de/rubric/how-to-do-wiki

API

An Application Programming Interface (API) enables external software to interact with MatInf. Details are available at
https://docs.google.com/document/d/1TDtHwtjP4RSXJw6m7jBvtmlSZ4qUBqWD/edit

User Manual

The user manual for RDMS as a collective effort of the MDI chair is available here:

https://docs.google.com/document/d/1w7lS24EE4cfZStLAaS9jSbV21GOODall/edit

Small How-to document, related to **publication management**:

https://docs.google.com/document/d/18ZPVxlymcqi03MUkTRCLPjOdg2aEkPya/edit

System settings

RDMS system settings are stored in the appsettings.json file and contain the following data:

SmtpConfiguration - to be done

…

Tenant settings

Tenant settings are stored in the **SettingsJson** attribute of the **Tenant** table and adjust tenant behavior. So far in Settings JSON there could be several parameters

- FinaliseOnObjectInsertUpdate
  controls action performed on an object creation or modification. If set to "SampleNamePrefix" then for all samples (objects of a type Sample (TypeId=6)) a sequential integer number is added to the object name as a prefix
- RedirectByObjectId_NotFound
  controls if a search request to /object/id/<ObjectId> gives no results. If set to "FallbackExternalId", then the next search by ExternalId is to be performed automatically.

```
{
   "FinaliseOnObjectInsertUpdate": "SampleNamePrefix",
   "RedirectByObjectId_NotFound": "FallbackExternalId"
}
```

It is also worth noting that GUI customisation is supported without modifying the server code through JS/CSS injections, allowing the default UI to be completely replaced with a customised version if required.

Apart from this, at the end of each page generated by RDMS is added content from `<appsettings.json:PathToFileStorage>\tenant<TenantId>\EndOfAllDocuments.html` file, if it is defined. It enables client-side HTML DOM and application state (stored in the window.app object) analysis together with its modification.

User-Defined Type System

Object Type Configuration

The flexible type system is the heart of the RDMS. The system allows user-defined types (UDT) that are fully configured according to the following attributes:

Type Name – user-defined type name

Table Name (Data Structure) – defines one of the core object types as a base entity. One of:

- **ObjectInfo** – the simplest (bare) object
- **Sample** – chemical system (set of chemical elements), for example, Co-O
- **Composition** – chemical composition (set of elements and quantities), for example, Co_3O_4
- **Reference** – literature reference (Authors, Title, Year, DOI, etc.)
- **Handover** – to trace object locations (sender, recipient, timestamps, amount, confirmations, notifications, etc…)

Validation Schema – only if you have external Web Services ready to validate data from files (see details below)

Data Schema – only if you have external Web Services ready to extract data from files (see details below)

File Required – Specifies whether the file is mandatory

Settings (JSON) – customization of type for RDMS that is not covered by other attributes. A JSON object with an extensible structure. The document could contain the following properties:

AllowedExtensions – an array object ([]), that contains a list of file extensions, that are considered to be valid for the type, for example: [".png", ".gif", ".jpg", ".jpeg"],

ApplyForTypeIds – an array that contains a list of type identifiers (TypeId) for which this type is applicable, for example (so far it is user to show handover object only for a given object types): [6, 83, 89],

CustomEditPath – a URL (relative or absolute) that indicates that objects of this type have a customised edit form (used for samples, although the same functionality is available within templates), for example: "/custom/editsample",

UrlGetVisualizers – an array of objects of the following structure: {"Url":"<url>", "Name":"<name>", "BiIcon":"<icon_class>"}, where:
<url> - URL (relative or absolute) that specifies that for objects there is a custom visualisation developed, that is available under the link (the specified endpoint here should be ready to get the data for visualisation using the query parameters appended to the URL:
- oid - ObjectId of the document for visualization;
- url - full URL to download the document for visualization.
The app can use either of these parameters to access the document for processing and generate the web page accordingly.
<name> - text of the link;
<icon_class> - class of the Bootstrap 5 icon, for example "bi-graph-up-arrow".
If several objects are defined, then a list of links is formed for visualization.

UrlPostVisualizer – a URL (relative or absolute) that specifies, that for objects there is a custom visualisation developed, that is available under the link (the specified endpoint here should be ready to get the data for visualisation on the request POST body and generate as an output a web page), for example: "https://domain.org/visualiser",

IncludePropertiesForm – specifies whether the template-based form should be included in the generic create/update form: 1 - indicates that the generic form should include template properties (0 - otherwise), for example: 1

IncludeTypeSettingsJsonInAPICalls - specifies whether to include TypeInfo.SettingsJson configuration document to the API call for data validation/extraction: 1 - include, 0 - do not include (default). Used in generic validators/data extractors in order to extract the expected schema from the TypeInfo.SettingsJson document.

ValidationContext – specifies that the validator and data extractor should become the context from the parent objects in order to validate/extract data. Context matters! One document could be considered valid or invalid in various contexts or provide different outputs depending on the context. Currently, the following context types are supported:
- "ChemicalSystem" - it provides a chemical system as a union of chemical systems from all direct parents (used in “EDX CSV” and “XPS CSV” to know the chemical system of interest and to filter out all other chemical elements, e.g., coming from substrate).
- "VROConnectionString” - adds to the Dictionary available within context object two parameters: “TenantId” and “VROConnectionString” (TenantId of the current tenant and Connection String to the database “vro” schema, where a set of read-only views is available). It is used in CRC 247, where cumulative documents are uploaded, which may contain properties for different samples.

- “TenantInfo” - adds to the Dictionary available within the context object a parameter “TenantUrl” that contains the URL address of the tenant, which is the source of the request.

CompositionTypeId – specifies the TypeId identifier used to identify the one of Composition-based object types on data upload, when compositions are created or updated Use cases:
a) in “EDX CSV” and in “XPS CSV” types to identify volume and surface compositions and to distinguish between them;
b) in “Resistance”, “Thickness”, etc… types to identify the composition type to search for according to the predicate specified (to exclude ambiguity).

OnObjectCreated – specifies that a configuration object for notifications should be in the object creation form. So far, the configuration object can contain only "EmailNotification" property (describes how to make an e-mail notification) with the values:

- "user" - an active user should be specified from a drop-down menu as a notification recipient.

- "userAndPI" - an active user should be specified from a drop-down menu as a notification recipient (project PI(s) is/are included in CC).

- "project" - a group of users (having a common "project" claim)

An example of the JSON:


```
{  "AllowedExtensions": [".txt", ".csv"],

   "ApplyForTypeIds": [6, 83, 89],

   "CustomEditPath": "/custom/editsample",

   "UrlPostVisualizer": "https://domain.org/visualiser",

   "IncludePropertiesForm": 1,

   "IncludeTypeSettingsJsonInAPICalls": 1,
```

```
  "ValidationContext": "ChemicalSystem",
  "CompositionTypeId": 125,
  "OnObjectCreated": { "EmailNotification": "user" }
}
```

Description – a user-friendly commentary explaining the type designation.

UDT Support API

Since RDMS supports user-defined types it's essential to provide a data format that fits a particular object type and even more - to enforce this via document validation and further benefit from data extraction. Validation and data extraction can be implemented in an external web service and the URL to this service should be configured in a user-defined type setting.

Documentation that covers User-Defined Types API support to enable validation, data extraction, and data visualisation from proprietary documents is described here (https://docs.google.com/document/d/1zg_inYhPL8OsbxmpFB0sofB_Qd5kOYip/edit?usp=sharing&ouid=117346421118620353373&rtpof=true&sd=true).
If you develop your own data type, you should develop a Web service to provide deep integration of the data type in the RDMS according to the mentioned instructions.

UDT: Supported Formats

Here you can find a list (or a catalog) of already supported data formats through UDT Support API (as described above).

EDX (C# implementation, 3 formats)

EDX is used to measure the **volume composition** of the sample surface. Currently, it supports CSV and TXT formats coming from two measurement devices (format slightly deviates in terms of element names, see the source code).
The measurement result is a single composition: for 342-grid standard Materials Library the result is 342 compositions (one for every MA).
The overall functionality could be tested via the Swagger test environment, available at https://validation.matinf.pro/
Remarks: "EDX CSV" type supports the context of EDX measurement (which should be linked to the parent sample). It means that the chemical system (elements of interest) is propagated from the parent object (which is the sample). It provides an upload of raw TXT files from the EDX device to RDMS in "EDX CSV" type, which will automatically exclude

substrate elements (everything except the system-defined at the parent sample level) and create measurement areas accordingly.
Question: What about the “EDX Raw (txt)” type from now on?
Will a CSV file be created from the raw TXT file?
If not, will we have two different file types (txt, csv) declared EDX CSV?
Answer: “EDX Raw (txt)” can be used if you do not want to create MAs, but just want to upload raw data (and after that, probably make CSV for upload to EDX CSV manually.
CSV file is not created as a separate object (I guess there is no need for that), but a dataset (and subsequent export to CSV) containing all compositions with their properties is on the way, so it should be even better.
"EDX CSV" always allows importing TXT and CSV, in principle, we could split this into "EDX CSV (csv)" and "EDX CSV (txt)". This could be a topic for discussion, but so far, I don't see a reason for it. "EDX CSV" leads to the creation of MAs, and "EDX TXT" does not - this is the crucial difference. Maybe the type names are a bit misleading; we can think about how to rename them to better reflect the difference in meaning.
All suggestions are welcome (but keep in mind that renaming will affect automatic type detection by file name).

XPS (C# implementation, 3 formats)

XPS is used to measure the **surface composition** of the sample. Currently, it supports CSV and TXT formats. The file formats are the same as in EDX case. The only difference is in the created composition types (**surface composition** instead of **volume composition**).

XRD Phase Overview CSV (C# implementation, 1 format)

Supports upload of qualitative phase analysis with the header:

```
Index;x;y;Crystal Structure; IDs
```

SDC Processed CSV (C# implementation, 1 format)

Processed Scanning Droplet Cell (SDC) measurement results in CSV (one normalized value per measurement area). CSV header example:

```
MA,x,y,SDC Area [cm²],Current Density [mA/cm²]
```

Implemented via **type:TypeValidationLibrary.TypeValidator_Generic_CSV** with the SettingsJson:

```
{ "AllowedExtensions":[".csv"], "CompositionTypeId":8,
"IncludeTypeSettingsJsonInAPICalls":1,
"DocumentFormat": {
   "IndexColumnName": ["Index", "Spectrum", "Measurement Area",
"MA"],
```

```
   "Output": [{"ColumnName":"SDC Area [cm²]",
"PropertyName":"SDC Area", "PropertyType":"Float",
"PropertyComment":"cm²"},{"ColumnName":"Current Density
[mA/cm²]", "PropertyName":"SDC Current Density",
"PropertyType":"Float", "PropertyComment":"mA/cm²"}] } }
```


Resistance (Python implementation, REST Web Service, 3 formats)

Resistance data are supported in two types, covering two formats (all accessible within a single service available at https://htts.matinf.pro/ with Swagger):

- **HTTS Resistance CSV** - files with **.csv** extensions that are validated according to the settings in the MDI tenant through https://htts.matinf.pro/resistance/csv/validation/body and data extracted through https://htts.matinf.pro/resistance/csv/data/databasevaluesbody) web service.
- **HTTS Resistance TXT** - files with **.txt** extension that are validated according to the settings in the MDI tenant through https://htts.matinf.pro/resistance/txt/validation/body and data extracted through https://htts.matinf.pro/resistance/txt/data/databasevaluesbody web service.

The CSV and TXT data formats, together with validation messages, are described and available here:

https://docs.google.com/document/d/1Gnprouy3_YMbhx9O13xjrNP_CBOYWqy7/edit?usp=sharing&ouid=117346421118620353373&rtpof=true&sd=true

Thickness (Python implementation, REST Web Service, 2 formats)

Thickness data are supported in two types, covering two formats (all accessible within a single service available at https://thickness.matinf.pro/ with Swagger):

- **Thickness Excel** - files with **.xls** and **.xlsx** extensions that are validated according to the settings in MDI tenant through https://thickness.matinf.pro/thickness/xlsx/validation (/body) and data extracted through https://thickness.matinf.pro/thickness/xlsx/data (/databasevaluesbody) web service.
- **Thickness TXT** - files with **.txt** extension that are validated according to the settings in MDI tenant through https://thickness.matinf.pro/thickness/txt/validation (/body) and data extracted through https://thickness.matinf.pro/thickness/txt/data (/databasevaluesbody) web service.

The CSV and TXT data formats, together with validation messages, are described and available here:

https://docs.google.com/document/d/1MEAMadA-nUmW5i_AN_7-0MgQs_ScvahH/edit?usp=sharing&ouid=117346421118620353373&rtpof=true&sd=true

BET adsorption report (pdf, txt) (C# implementation, 3 formats)

BET surface area measurement for nanoparticle powder samples is supported. The file format is either PDF (which contains EITHER “Anton Paar Kaomi for Nova” title and a set of parameters OR data format from MicroActive device) or TXT (which is a standardized output from a measurement device). The data extraction procedure produces two properties:

- **BET_R** - Correlation coefficient
- **BET_Area** - Surface area, m²/g

Properties are added to the “BET adsorption report (pdf)” object and additionally **BET_Area** property is added to all parents (samples).

BandGap Processed CSV (C# implementation, 1 format)

Processed bandgap measurement results in CSV (one bandgap value per measurement area). CSV header example:

```
MA,x,y,bandgap
```

Implemented via **type:TypeValidationLibrary.TypeValidator_Generic_CSV** with the SettingsJson:

```
{ "AllowedExtensions":[".csv"], "CompositionTypeId":8,
"IncludeTypeSettingsJsonInAPICalls":1, "DocumentFormat": {
"IndexColumnName": ["Index", "Spectrum", "Measurement Area",
"MA"], "Output": [{"ColumnName":"bandgap",
"PropertyName":"Bandgap", "PropertyType":"Float",
"PropertyComment":"eV"}] } }
```

Phase Overview Prediction CSV (C# implementation, 1 format)

Phase Overview Prediction CSV is implemented via **type:TypeValidationLibrary.TypeValidator_PhasePrediction_CSV**:

```
MA,x,y,Ni,Pd,Pt,Ru,Temp,Number of Phases,Phase Names,Phase Fractions

--// "['FCC_A1#1', 'HCP_A3#1']",[0.91686304 0.08313696]
```

The SettingsJson:

```
{ "AllowedExtensions":[".csv"], "CompositionTypeId":8,
"IncludeTypeSettingsJsonInAPICalls":1, "Visualization": {
```

"FilterParameters":[{"ColumnName":"Temp", "Type": "Float",
"FilterCaptionHtml": "Select Temperature (K)"}] } }

Catalysis Results (xlsx) (C# implementation, 1 format + 2 pending…)

LSV (xlsx, csv, txt, <empty>) (JavaScript implementation for the Tafel slope analysis, 5 formats supported)

Formats (LSV & CV with manual cycle selection) with headers:

- `WE(1).Potential (V)  WE(1).Current (A)  Time (s)`
- `time/s  control/V  Ewe/V<I>/mA  cycle number  (Q-Qo)/C  I Range`
- `cycle number  time/s  Ewe/V<I>/mA`
- `Number  Time/s  Potential/V  Current/A`
- `Potential  Current`

Special use case to support publications and provide the baseline Tafel analysis for the community!

UDT: Other Considerations

Some general common sense rules for documents, characterizing materials libraries and data formats:

Data files should be generated in .csv (better) or .xlsx format.

- The file contains only measurement data regarding category, with no extra explanation.
- The data should have (when possible) a table structure: if you have 5 measurement results, then you should have 5 rows with corresponding columns.
- Each column must have a name (the first row must contain column names).
- The measurement areas must start at 1 and end at 342 (please do not start with 0, as this would make the data inconsistent unless it's taken into account by the extractor, which is possible but not recommended).
- The suggested table to create is as follows:

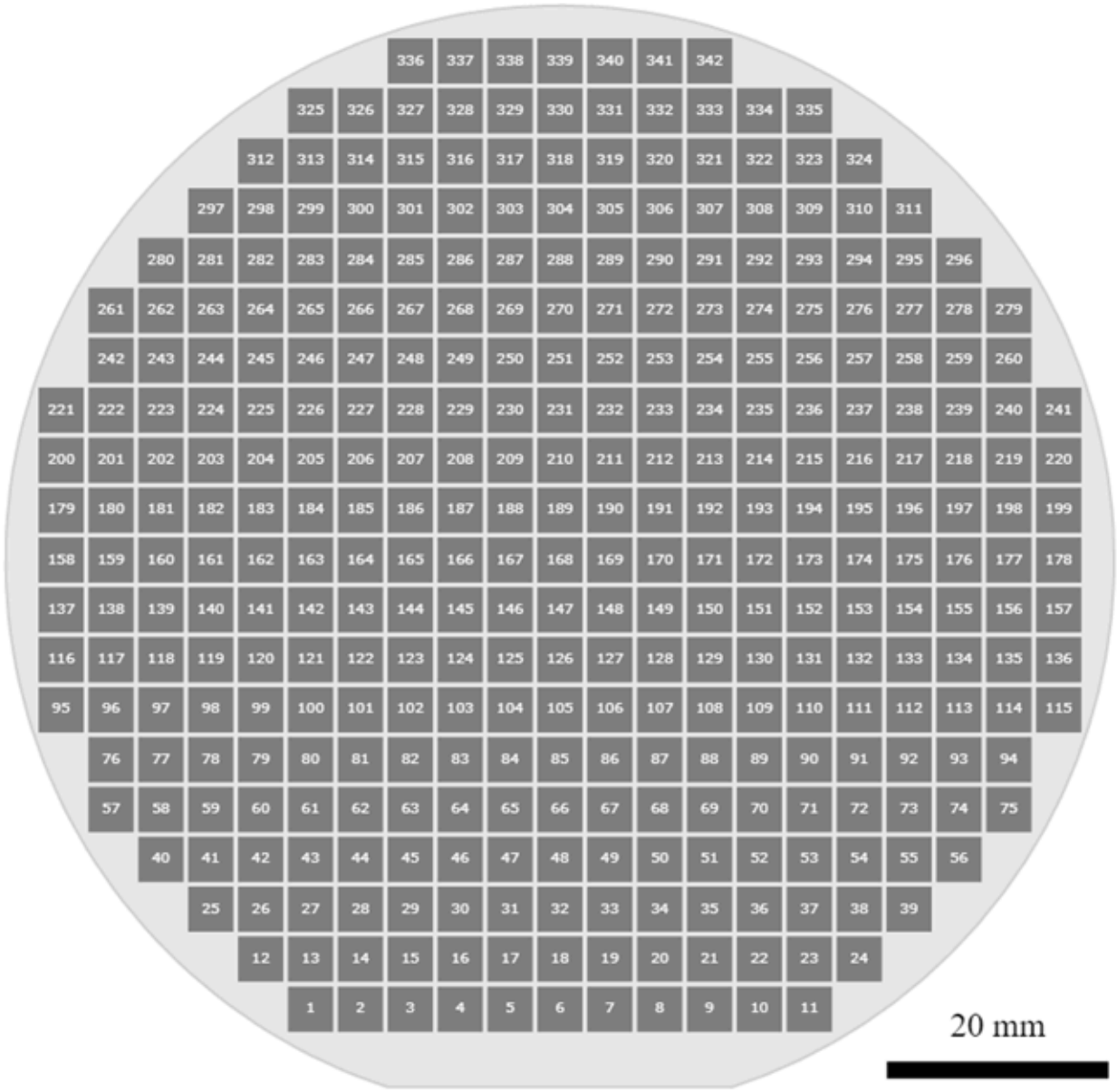
x	y	MA	T	T-set	Measured parameter 1 (e.g. R1 or Thickness1)	Measured parameter 2 (e.g. R2 or Thickness2)	Measured parameter 3 (e.g. R3 or Thickness3)

		1					
		2					
		…					
		342					

Legend:

x, y - coordinates of the MA (device related)

MA - “Measurement Area” (4.5x4.5 mm) Index on the Materials Library (for example, according to the MDI Measurement Area Grid):



T - actual measured Temperature (for temperature-dependent measurements), in °C

T-set - Temperature as set by the device, in °C