RDE構造化処理プログラム開発 手順書

～ **RDEToolKit**の各登録モードの利用事例 ～

リリース**1.3.0 (20250930)**

国立研究開発法人 物質・材料研究機構

目次

1. はじめに

1.1 構造化処理の4つのモード

RDEにデータを登録する構造化処理には、以下に示すような4つのモードが存在します。

- インボイスモード
- エクセルインボイスモード
- マルチデータタイルモード
- RDEフォーマットモード

本ドキュメントでは、"モード"それぞれの概要と、利用方法について説明します。

モード名	制御ファイル	起動条件
インボイスモード	なし	デフォルトで起動
エクセルインボイスモード	なし	入力ファイルに `*._excel_invoice.xlsx` を格納
マルチデータタイルモード	Config	`extended_mode="MultDataTile"` を指定し、データセットテンプレートを作成する。
RDEフォーマットモード	Config	`extended_mode="rdeformat"` を指定し、データセットテンプレートを作成する。

- Configに指定する方法については、後述します。

1.1.1 モードとは何か?

モードは何で、どういった違いがあるでしょうか?

RDEToolKitを用いて作成した構造化処理プログラムを実行する際の初期段階において、入力ファイルをどのように展開するかの種別を"モード"と言います。

入力ファイルに合わせて適切なモードを設定して構造化処理プログラムを作成することにより、zip圧縮された入力ファイルを適切なディレクトリに展開したり、本来であれば複数回のアップロードが必要な登録作業を一度に実行するようになります。その結果、複数回の構造化処理が必要な場合には、適切に展開、配置されたそれぞれのデータフォルダに対して、構造化処理プログラムが実行されるようになります。

つまり、`入力ファイルの展開` や、`Excelファイルの内容を読み込み、invoice.jsonを生成する` などの処理を、開発者がそれぞれ実装する必要はなく、RDEToolKitに"任せる"ことでできるようになります。

以下、それぞれのモードについて、その概要を示します。

1.1.2 インボイスモード

このモードは、通常のRDE登録画面でデータを登録するモードです。一番、基本的かつデフォルトのモードです。

以下のように、Entry画面から、データを投入するモードです。

1.1.3 エクセルインボイスモード

このモードは、一度に複数のデータを登録するためのモードです。通常のインボイスモードでは一件ずつしかデータの登録が実行できませんが、エクセルインボイスモードを使うと、一度に複数のデータ(→複数のデータタイル)を登録することができます。

入力ファイルに、`*_excel_invoice.xlsx` という命名規則を持つExcelファイルが存在する場合、自動的にエクセルインボイスモードに移行し、Excelファイルの内容を使ってデータが登録されます。

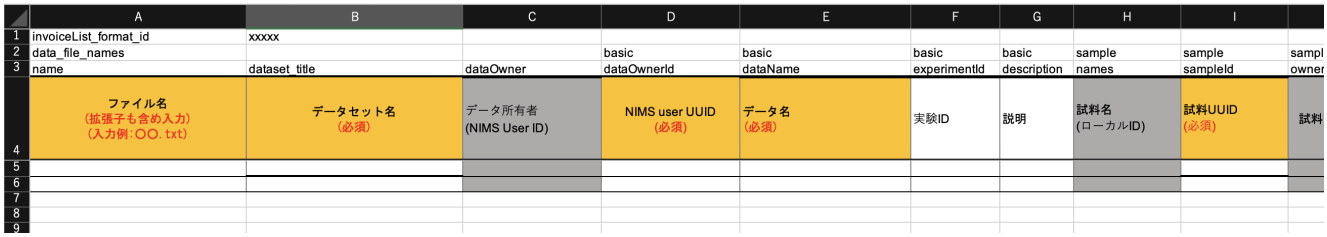
> いくつかのテンプレートで、上記命名規則を持たないExcelファイルをエクセルインボイスとして利用するように実装しているものがあります。これはtasksupportフォルダに、excelinvoice_flag.txtというファイルが存在し、かつ、入力ファイルとしてエクセルファイルが登録された場合に、そのエクセルファイルのファイル名末尾を、`*_excel_invoice.xlsx` に変更することで実現しています。すなわち、構造化処理プログラムの開発者がそれらの処理を自前で実装する必要があることに注意してください。

インボイスとして利用するExcelファイルは、以下の場所にテンプレートファイルがあり、そちらをダウンロードし変更して利用します。

- エクセルインボイステンプレート - rde-operation

> 上記サイトはNIMS内部からのみアクセスが可能であり、かつ有効なアカウントを保持しているユーザのみが利用可能です。他の環境で必要な場合は、関係者からファイルのコピーを受け取ってください。

エクセルインボイスとして構成したExcelファイルのフォーマット例を以下に示します。



	A	B	C	D	E	F	G	H	I	
1	invoiceList_format_id	xxxxx								
2	data_file_names			basic	basic	basic	basic	sample	sample	sampl
3	name	dataset_title	dataOwner	dataOwnerId	dataName	experimentId	description	names	sampleId	owner
4	ファイル名 （拡張子も含め入力） （入力例：〇〇.txt）	データセット名 （必須）	データ所有者 (NIMS User ID)	NIMS user UUID （必須）	データ名 （必須）	実験ID	説明	試料名 (ローカルID)	試料UUID （必須）	試料
5										
6										
7										
8										
9										

RDEToolKit v1.1.0より、エクセルインボイスで利用するExcelファイルのひな形を生成する機能が提供されています。詳細は後述します。

1.1.4 マルチデータタイルモード

マルチデータタイルモードは、一度のデータデータ受入(エントリー)で複数のデータを登録する処理をサポートするモードです。RDEのデーター覧画面におけるデータの表示単位をデータタイルと呼ぶことがあります。複数のデータタイルを同時に登録できるという意味で「マルチデータタイル」モードとしています。

> 複数のファイルを1つのデータタイルに登録する場合は、"インボイスモード(またはエクセルインボイスモード)"を使います。

1.1.5 RDEフォーマットモード

RDEフォーマットモードは、予めローカルPCなどで構造化処理処理を行い、その結果を構造化処理後のフォルダ構成(RDEフォーマット形式)でファイルを配置したファイル群をzipアーカイブしたものを登録するための機能を提供します。

入力データとして、以下のフォーマットを持つzip形式のファイルを投入する必要があります。

zipファイルの中に、invoice, main_image, other_image, structured、meta、raw、nonshared_raw、thumbnailといったフォルダがあり、その中にそれぞれのファイルを格納します。zipファイルは、登録前に構造化処理を実行し、そのファイルをそのまま格納します。

```
|- sample.zip
    |- invoice/
    |   |- invoice.json
    |- main_image/
    |   |- xxxx.png
    |- meta/
    |   |- metadata.json
    |- nonshared_raw/
    |- other_image/
    |   |- xxxxx.png
    |- raw/
    |   |- xxxxxx.xxx
    |- structured/
    |   |- xxxxxxx.csv
    |- thumbnail/
    |   |- xxxx.png
```

通常のインボイスモードと同様のフォルダ構成の"シングルデータタイル"書き込みと、エクセルインボイスモードと同様のフォルダ構成の"マルチデータタイル"書き込みの2つのパターンがあり、どちらでも構造化処理プログラムを変更することなく利用可能です。

> RDETooKitを利用した構造化処理プログラムでは、RDEフォーマットモードとして取り込んだファイルに対してさらに構造化処理を実行するようなプログラムを作成することも可能です。

1.2 使用するモードの選択基準

では、構造化処理プログラムを作成する際に、どのモードを利用すべきでしょうか?

以下の順に考慮して、使用するモードを選択してください。

1.2.1 RDEサーバでの構造化処理が不要な場合

RDEフォーマットモードは、すでに手元のPCなどで構造化処理が済んでいる等の理由でRDE上で構造化処理を実行せず、そのままRDEデータセットに取り込むモードです。

サーバに取り込んだ後の構造化処理が不要の場合は、RDEフォーマットモードの利用を考えてください。

- 生データや画像などのデータを、RDEが想定しているフォルダに配置する処理も"構造化処理"に含まれます。配置をRDEに任せる場合は、RDEフォーマットモードは利用出来ません。他のモードを利用してください。

1.2.2 構造化処理の負荷が高い場合

構造化処理の負荷が高い場合、つまり構造化処理に時間がかかる場合も、できればローカルPCなどでの"前処理"を実行し、RDEフォーマットモードでの取り込みを考慮してください。

> こちらは運用上の"お願い"となります。他のユーザへの影響を考慮してのお願いです。

1.2.3 登録すべきデータが複数個存在

- 登録すべき過去の実験データが数百件以上ある、と言った場合です。

ファイル1つ毎にデータ(データタイル)として登録し、かつ、すべてのデータで同じ送り状の内容で良い場合は、マルチデータタイルモードを利用します。

> ただし、セットする値に入力ファイル名をそのままセットすればよい場合は、マジック変数(${filename})が利用できます。マジック変数については本書では扱いません。他のマニュアルを参照ください。

1つのデータタイルが単一ファイルで構成される場合で、送り状やメタデータをデータごとに変更して登録する場合は、エクセルインボイスモードを利用します。

複数のファイルを1つのデータタイルとして格納する場合で、送り状の内容をデータタイルごとに変更する場合も、エクセルインボイスモードを利用します。

1.2.4 それ以外の場合

データ発生の都度、登録画面から登録することが可能な場合は、インボイスモードを利用します。

1.2.5 インボイスモードとエクセルインボイスモードの同時利用

入力データとして、所定の命名規則に則ったExcelファイルが存在する場合はエクセルインボイスモードとしてデータを登録し、存在しない場合はインボイスモードとして登録する、という構造化処理を構成可能です。

通常は1つのデータセットに"2つのモード"のどちらを使っても登録することができます。ただし、以下の様な構造化処理プログラムを作成している場合は、修正が必要となります。

- 入力ファイルやインボイスファイルのパスの求め方を、Excelインボイスモード実行時にも対応できる書き方をしていない。
- `構造化処理プログラム内でExcelファイルを指定して情報を読み込む` など、Excelインボイスファイルの存在を前提として処理が書かれている。

1.3 本書の構成

次章以降は、以下の様に構成していきます。

最初に、すべてのモードに共通する処理をまとめて記述します。

続いて、「マルチデータタイルモード」について示します。

次にRDEフォーマットモードについて示します。

最後に、エクセルインボイスモードについて示します。エクセルインボイス内では最初に通常のインボイスモードについて示し、次に同じ環境を利用したエクセルインボイスモードについて示します。

> 通常の"インボイスモード"を独立した章とはしていませんの注意してください。

2. 共通処理

次章以降で説明する各モード共通の処理をまとめます。

2.1 RDEToolKitのインストール

他の文書などを参考に、RDEToolKitをインストールします。

本文書では、Pythonの仮想環境 `venv` に対してRDEToolKitが導入されているものとして進めます。

```
$ source venv/bin/activate

(venv) $ python -m rdetoolkit version
1.3.4
```

> RDEToolKitは随時アップデートされますので上記とは異なる表記となる場合があります。

現時点(2025年09月)では、`v1.3.4` が最新となっています。それより前のバージョンをお使いの場合は、最新バージョンにアップデートすることができます。

```
pip install rdetoolkit --upgrade
```

> 新規開発の場合は、RDEToolKitの最新バージョンをお使いください。既存バージョンの修正の場合は、以前と同じバージョンでの利用でも問題ありませんが、RDEToolKitの最新バージョンの導入とそれに合わせたPythonスクリプトの修正を推奨します。

2.2 初期フォルダ構成作成

本書では、`$HOME/rde-sample/<モード名>` 下に構成すること想定します。

"<モード名>"の部分は、例えばマルチデータタイルモードでは"multidatatile"が、エクセルインボイスモードでは"excelinvoice"のようになります。

#	モード名	フォルダ名	備考
1	マルチデータタイルモード	$HOME/rde-sample/multidatatile	
2	エクセルインボイスモード	$HOME/rde-sample/excelinvoice	
3	RDEFormatモード	$HOME/rde-sample/rdeformat	
4	インボイスモード	$HOME/rde-sample/invoice	本書ではエクセルインボイスの一部として作成するので、明示的なフォルダは作成しません。

> RDEにて上記フォルダ名が決まっているわけではありません。ローカル開発においては、自由にフォルダ名を決めることができます。

以下、マルチデータタイルモードの場合の例を示します。

```
cd ${HOME}
mkdir -p rde-sample/multidatatile
```

当該フォルダに移動し、RDEToolKitの初期化コマンドを実行します。

```
(venv) $ cd rde-sample/multidatatile

(venv) $ python -m rdetoolkit init
Ready to develop a structured program for RDE.
Created: /home/devel/rde-sample/multidatatile/container/requirements.txt
Created: /home/devel/rde-sample/multidatatile/container/Dockerfile
Created: /home/devel/rde-sample/multidatatile/container/data/invoice/invoice.json
Created: /home/devel/rde-sample/multidatatile/container/data/tasksupport/invoice.schema.json
Created: /home/devel/rde-sample/multidatatile/container/data/tasksupport/metadata-def.json
Created: /home/devel/rde-sample/multidatatile/templates/tasksupport/invoice.schema.json
Created: /home/devel/rde-sample/multidatatile/templates/tasksupport/metadata-def.json
Created: /home/devel/rde-sample/multidatatile/input/invoice/invoice.json

Check the folder: /home/devel/rde-sample/multidatatile
Done!
```

> 上記例は、ユーザ `devel` にて実行した場合の例です。実行ユーザに関しては以下同様とします。

構造化処理プログラムは `container/` フォルダ下に作成しますので、移動します。

```
cd container/
```

2.3 モード指定方法およびRDEToolKit設定ファイルの利用方法

前述の様に、マルチデータタイルモードおよびRDEフォーマットモードを使用する場合は、その旨を指定する必要があります。

指定の仕方はいくつかありますが、大きくは以下の2つの方法になります。

1. rdetoolkit.workflows.run()実行時に、Configオブジェクトを指定し、その中で指定する方法
2. 設定ファイル(data/tasksupport/rdeconfig.yml または data/tasksupport/rdeconfig.yaml)に指定する方法

> 上記1.と2.を同時に使用することはできません。1.の方法を使った場合、2.の設定ファイルを設置しても無視されます。

なお、設定できる内容はモードの指定だけではなく、いくつかのRDEToolKitの振る舞いを変更することができます。本章では、モード指定の方法以外の `設定` の指定内容や、その方法についての記述も合わせて行います。

2.3.1 Config オブジェクトを指定する方法

main.py中で、rdetoolkit.workflows.run()を実行する前に、以下の様に設定します。

```
import rdetoolkit
from rdetoolkit.config import Config, SystemSettings

config = Config(
    system=SystemSettings(
        extended_mode="rdeformat",
        save_thumbnail_image=True
    ),
)
rdetoolkit.workflows.run(config=config)
```

> main.pyを上記の様に変更しただけでは、実行時に
> `rdetoolkit.exceptions.StructuredError: ERROR: no zipped input files` となります。RDEFormatモードを利用する場合の例については、次章以降で説明します。

このように、Configオブジェクトにはいくつかのプロパティを設定することができます。

`system` プロパティは、RDEToolKitの設定として利用されます。

`multidata_tile` プロパティは、`system` プロパティの `extended_mode` で"MultiDataTile"が指定されている場合に、その設定として利用されます。

その他のプロパティは、開発者が自由に設定、利用することができます。

`system` プロパティで設定できる内容は以下の通りです。

項目名	意味	フォーマット	規定値	備考
extended_mode	拡張モードの使用を宣言	文字列	指定なし	
save_raw	inputdataにあるファイルをrawフォルダにコピーするか?	Boolean	False	
save_nonshared_raw	inputdataにあるファイルをnonshared_rawフォルダにコピーするか?	Boolean	True	
save_thumbnail_image	main_imageにある画像ファイルをthumbnailとして保存するか?	Boolean	True	
magic_variable	マジック変数({$filename})をファイル名に置き換えるか?	Boolean	False	

> extended_modeで指定できるのは、'rdeformat' または 'MultiDataTile' (大文字小文字を区別)です。それ以外を指定した場合は、指定しない場合と同様に解釈されます。
>
> これらの項目は、すべて任意指定です。指定されない場合は、規定値が指定されたものとして実行されます。

`multidata_tile` プロパティで設定できる内容は以下の通りです。

項目名	意味	フォーマット	規定値	備考
ignore_errors	エラー発生時に処理を継続するか否か?	Boolean	False	デフォルトでは処理を止める

なお、Configオブジェクトは、何も設定しないと既定値を指定したのと同じ設定となりますので、以下の様にすることも出来ます。

```
import rdetoolkit

config = rdetoolkit.config.Config()
config.system.extended_mode = "rdeformat"
config.system.save_thumbnail_image = True

rdetoolkit.workflows.run(config=config)
```

> 最初の `config = rdetoolkit.config.Config()` で、すべて既定値がセットされたConfigオブジェクトを生成し、その下の2行で、既定値とは違う値をセットするもののみをセットします。

2.3.2 設定ファイルを使用する方法

上と同じ内容を、data/tasksupport/rdeconfig.yml または data/tasksupport/rdeconfig.yamlに記述することで、Configを指定します。

data/tasksupport/rdeconfig.yml

```
# コメント
system:
    save_raw: False
```

```
    # ↓ 空白行もOK

    save_nonshared_raw: False
```

`modules/datasets_process.py` を以下の様にして確認します。

```
from pprint import pprint

from rdetoolkit.models.rde2types import RdeInputDirPaths, RdeOutputResourcePath


def dataset(srcpaths: RdeInputDirPaths, resource_paths: RdeOutputResourcePath) -> None:
    config = srcpaths.config
    print(type(config))
    pprint(config)
```

実行してみます。

```
(venv) $ python main.py
<class 'rdetoolkit.models.config.Config'>
Config(system=SystemSettings(extended_mode=None, save_raw=False, save_nonshared_raw=False, save_thumbnail_image=False,
magic_variable=False), multidata_tile=MultiDataTileSettings(ignore_errors=False))
```

> 設定内容が反映されていることが確認できます。

2.3.3 ユーザ指定設定項目

上で示したプロパティ以外のプロパティは、"ユーザ指定設定"となります。上記とかぶらなければ任意のプロパティ名を構造化処理プログラムの中で利用することができます。

例えば、以下の様に設定できます。

data/tasksupport/rdeconfig.yml

```
# コメント
system:
    save_raw: False
    # ↓ 空白行もOK

    save_nonshared_raw: False

user:
    x-label: "X軸"
    y-label: "Y軸"
```

利用する場合は以下の様にします。

```
from pprint import pprint

from rdetoolkit.models.rde2types import RdeInputDirPaths, RdeOutputResourcePath


def dataset(srcpaths: RdeInputDirPaths, resource_paths: RdeOutputResourcePath) -> None:
    user_config = srcpaths.config.user
    print(type(user_config))
    pprint(user_config)
```

上記の結果は以下の様になります。

```
(venv) $ python main.py
<class 'dict'>
{'x-label': 'X軸', 'y-label': 'Y軸'}
```

> 取得した設定は、通常のDictとして利用できます。

3. RDEフォーマットモード

前述の様にRDEフォーマットモードは、入力データをRDEデータセットの形式にそのまま登録します。

シングルデータタイルとして登録する場合とマルチデータタイルとして登録する場合で、構造化処理プログラムに変更があるわけではありません。入力データ、つまりzipファイル内のフォルダ構成を変更することで、自動的に切り替わります。

3.1 RDEフォーマットとは

同一のフォルダ下に、以下のサブフォルダが存在し、その中にそれぞれが想定するファイルが格納されているものを"RDEフォーマット"と称します。

- invoice (インボイスファイルを格納)：必須
- main_image (主たるイメージを格納)：任意
- meta (抽出したメタデータファイルを格納)：任意
- nonshared_raw (展開した入力ファイルを格納)：任意 (入力ファイルを"非公開"にする場合に使用)
- other_image (その他のイメージを格納)：任意
- raw (展開した入力ファイルを格納)：任意 (入力ファイルを"公開"にする場合に使用)
- structured (構造化処理後の可読ファイルを格納)：任意
- thumbmail (サムネイル画像を格納)：任意

```
|- sample.zip
    |- invoice/
    |   |- invoice.json
    |- main_image/
    |   |- xxxx.png
    |- other_image/
    |   |- xxxxx.png
    |- structured/
    |   |- xxxxxx.csv
```

> 上記に記述のない名称のフォルダは取り込み処理にて無視されます。

複数のデータタイルに登録する場合は、divided/XXXX(XXXXは0001から始まる数字連番)フォルダに、上記と同じ構造を取ることで取り込みを実現出来ます。

```
|- sample.zip
    |- divided
    |   |- 0001
    |      |- invoice/
    |          |- invoice.json
    |      |- main_image/
    |          |- yyyy.png
    |      |- other_image/
    |          |- yyyyy.png
    |      |- structured/
    |          |- yyyyyy.csv
    |   |- 0002
```

```
    |      |- invoice/
    |          |- invoice.json
    |      |- main_image/
    |          |- zzzz.png
    |      |- other_image/
    |          |- zzzzz.png
    |      |- structured/
    |          |- zzzzzz.csv
    |- invoice/
    |   |- invoice.json
    |- main_image/
    |   |- xxxx.png
    |- other_image/
    |   |- xxxxx.png
    |- structured/
    |   |- xxxxxx.csv
```

> 上記例だと、3つのデータタイルとして取り込み処理が実施されます。
>
> いずれのフォルダにあるinvoice.jsonも、最上位tasksupportフォルダにあるinvoice.schema.jsonの定義に沿った内容である必要があります。

本書では、前者を"シングルデータタイル登録"、後者を"マルチデータタイル登録"と呼ぶことにします。

3.2 RDEToolKitのインストールと初期フォルダ構成作成

『共通処理』の章で示したように、RDEToolKitのインストールと、初期フォルダ構成の作成を実施します。

```
$ cd rde-sample
$ mkdir rdeformat
$ cd rdeformat/

$ source ~/venv/bin/activate
(venv) $ python -m rdetoolkit init
 :
(venv) $ cd container
```

以下、双方が完了しているものとして進めます。

3.3 シングルデータタイル登録

> 先に"シングルデータタイル"での実行例を示し、その後"マルチデータタイル"での実行例を示します。

3.3.1 テスト用データ配置

テスト用のデータとして、本書と同梱で配布されている、以下のファイルを配置します。

- structured.zip (370KB)

実際に格納されると、以下の様になります。

```
(venv) $ ls -l data/inputdata/
total 372
-rwxr-xr-x 1 devel devel 377909  9月 17 16:50 structured.zip
```

3.3.2 フォルダ初期化

開発においては、構造化処理プログラムを何度も実行することになります。

都度手動にて初期化するのは大変ですので、以下の内容で初期化用スクリプトを作成します。

reinit.sh

```
#!/bin/bash


# ./data/inputdata
#  -> 何もしない

#./data/job.failed
if [ -f ./data/job.failed ];then
    echo "./data/job.failed was removed"
    rm -f ./data/job.failed
fi

# ./data/invoice
#  -> 何もしない

#./data/tasksupport
#  -> 何もしない

#./modules
#  -> 何もしない

#./requirements.txt
#  -> 何もしない

D="
./data/devided
./data/logs
./data/divided
./data/attachment
./data/invoice_patch
./data/meta
./data/main_image
./data/other_image
./data/raw
./data/nonshared_raw
./data/structured
./data/temp
./data/thumbnail
"
for d in ${D};do
    echo "${d} was removed"
    rm -rf ${d}
done

# Python cache
rm -rf modules/__pycache__
```

実行権限を付与します。

```
chmod a+x reinit.sh
```

一度実行して問題がないことを確認します。

```
(venv) $ ./reinit.sh
./data/devided was removed
./data/logs was removed
./data/divided was removed
./data/attachment was removed
./data/invoice_patch was removed
./data/meta was removed
./data/main_image was removed
./data/other_image was removed
./data/raw was removed
./data/nonshared_raw was removed
./data/structured was removed
./data/temp was removed
./data/thumbnail was removed
```

3.3.3 構造化処理作成

ここから、実際に構造化処理プログラムを作成していきます。

> 以下"container/"フォルダからの相対パスとしてファイル名を記述します。

main.py

初期化にて作成されたmain.pyに以下を加えていきます。

1. RDEFormatモードを使用するように"設定"を追加
2. 同時にデフォルトではthumbnailフォルダへの画像ファイルコピーが実行されないので、実行するように設定を追加

> 別述のように、これらの設定内容は、main.py内に記述する方法の他、tasksupportフォルダにrdeconfig.ymlファイルを配置することでも実現できます。その方法を使う場合は、main.pyを変更する必要はありません。本章では、main.pyに記述する場合の例を示します。

main.py

```
import rdetoolkit
from rdetoolkit.config import Config, SystemSettings


def main() -> None:
    config = Config(
        system=SystemSettings(
            extended_mode="rdeformat",
            save_thumbnail_image=True
        ),
    )

    rdetoolkit.workflows.run(
        config=config
    )

if __name__ == '__main__':
    main()
```

この例では、上で示したように、RDEフォーマットモードの使用(extended_mode="rdeformat")と、サムネイル画像の自動保存(save_thumbnail_image=True)の2つを設定しています。

> 基本的に指定されたデータを取り込むだけですので、custom_dataset_function句の設定やmodulesフォルダへ構造化処理プログラムの配置は行いません。

3.3.4 実行

実行してみます。

まず実行前のdataフォルダは以下の様になっています。

```
(venv) $ tree data
data
├── inputdata
│   └── structured.zip
├── invoice
│   └── invoice.json
└── tasksupport
    ├── invoice.schema.json
    └── metadata-def.json

4 directories, 4 files
```

実行します。

```
python main.py
```

実行後のディレクトリ構成は、以下の様になっています。

```
(venv) $ tree data
data
├── attachment
├── inputdata
│   └── structured.zip
├── invoice
│   └── invoice.json
├── invoice_patch
├── logs
│   └── rdesys.log
├── main_image
│   └── gbp_rank2_numPeak2_shirley_mSG1504_result.png
├── meta
│   └── metadata.json
├── nonshared_raw
├── other_image
│   ├── BIC_vs_NumPeak.png
│   └── parameter_table.png
├── raw
│   └── Cs2O_O2s.zip
├── structured
│   ├── cmd_mrs.log
│   ├── gbp_rank2_numPeak2_shirley_mSG1504_parameters.csv
│   ├── settings_Linear.inp
│   └── settings_Shirley.inp
├── tasksupport
│   ├── invoice.schema.json
│   └── metadata-def.json
├── temp
│   ├── invoice_org.json
```

```
|   ├── main_image
|   |   └── gbp_rank2_numPeak2_shirley_mSG1504_result.png
|   ├── meta
|   |   └── metadata.json
|   ├── other_image
|   |   ├── BIC_vs_NumPeak.png
|   |   └── parameter_table.png
|   ├── raw
|   |   └── Cs2O_O2s.zip
|   ├── structured
|   |   ├── cmd_mrs.log
|   |   ├── gbp_rank2_numPeak2_shirley_mSG1504_parameters.csv
|   |   ├── settings_Linear.inp
|   |   └── settings_Shirley.inp
|   └── thumbnail
|       └── gbp_rank2_numPeak2_shirley_mSG1504_result.png
└── thumbnail
    └── gbp_rank2_numPeak2_shirley_mSG1504_result.png

21 directories, 26 files
```

> tempフォルダは、invoice_org.jsonを除き、zipファイルを展開したフォルダ/ファイルが展開されています。このフォルダは、RDE取り込み処理では無視され、取り込み対象とはならないことに注意してください。
>
> 上記のようにならず、nonshared_raw/ フォルダに structured.zip がそのままコピーされた状態となった場合は、rdeformatモードを使う設定が正しくなっていないことが考えられます。前章などを参考に、モード設定が正しく行われているかを確認してください。

3.4 マルチデータタイル登録

続いて"マルチデータタイル"での実行例を示します。

3.4.1 フォルダ初期化

フォルダを初期化します。

```
(venv) $ ./reinit.sh
./data/devided was removed
./data/logs was removed
./data/divided was removed
./data/attachment was removed
./data/invoice_patch was removed
./data/meta was removed
./data/main_image was removed
./data/other_image was removed
./data/raw was removed
./data/nonshared_raw was removed
./data/structured was removed
./data/temp was removed
./data/thumbnail was removed
```

3.4.2 テスト用データ配置

上で使用したstructured.zipを削除します。

```
rm data/inputdata/structured.zip
```

テスト用のデータとして、本書と同梱で配布されている、以下のファイルを配置します。

- data.zip (676KB)

実際に格納されると、以下の様になります。

```
(venv) $ ls -l data/inputdata/
total 676
-rwxrwxrwx 1 devel devel 691369  1月 31 13:36 data.zip
```

3.5 構造化処理作成

シングルデータタイル登録で使用したものから変更する必要はありません。

3.6 実行

実行してみます。

まず実行前のdataフォルダは以下の様になっています。

```
(venv) $ tree data
data
├── inputdata
│   └── data.zip
├── invoice
│   └── invoice.json
├── logs
└── tasksupport
    ├── invoice.schema.json
    └── metadata-def.json

5 directories, 4 files
```

実行します。

```
python main.py
```

実行後のディレクトリ構成は、以下の様になっています。

```
(venv) $ tree data
data
├── attachment
├── divided
│   └── 0001
│       ├── attachment
│       ├── invoice
│       │   └── invoice.json
│       ├── invoice_patch
│       ├── logs
│       ├── main_image
│       │   └── gbp_rank1_numPeak4_shirley_mSG0112_result.png
│       ├── meta
```

```
|       |   └── metadata.json
|       ├── nonshared_raw
|       ├── other_image
|       |   ├── BIC_vs_NumPeak.png
|       |   └── parameter_table.png
|       ├── raw
|       |   └── La2O3_O2s.zip
|       ├── structured
|       |   ├── cmd_mrs.log
|       |   ├── gbp_rank1_numPeak4_shirley_mSG0112_parameters.csv
|       |   ├── settings_Linear.inp
|       |   └── settings_Shirley.inp
|       ├── temp
|       └── thumbnail
|           └── gbp_rank1_numPeak4_shirley_mSG0112_result.png
├── inputdata
|   └── data.zip
├── invoice
|   └── invoice.json
├── invoice_patch
├── logs
|   └── rdesys.log
├── main_image
|   └── gbp_rank1_numPeak2_shirley_mSG0672_result.png
├── meta
|   └── metadata.json
├── nonshared_raw
├── other_image
|   ├── BIC_vs_NumPeak.png
|   └── parameter_table.png
├── raw
|   └── Cs2O_O1s.zip
├── structured
|   ├── cmd_mrs.log
|   ├── gbp_rank1_numPeak2_shirley_mSG0672_parameters.csv
|   ├── settings_Linear.inp
|   └── settings_Shirley.inp
├── tasksupport
|   ├── invoice.schema.json
|   └── metadata-def.json
├── temp
|   ├── data
|   |   ├── divided
|   |   |   └── 0001
|   |   |       ├── invoice
|   |   |       |   └── invoice.json
|   |   |       ├── main_image
|   |   |       |   └── gbp_rank1_numPeak4_shirley_mSG0112_result.png
|   |   |       ├── meta
|   |   |       |   └── metadata.json
|   |   |       ├── other_image
|   |   |       |   ├── BIC_vs_NumPeak.png
|   |   |       |   └── parameter_table.png
|   |   |       ├── raw
|   |   |       |   └── La2O3_O2s.zip
|   |   |       └── structured
|   |   |           ├── cmd_mrs.log
|   |   |           ├── gbp_rank1_numPeak4_shirley_mSG0112_parameters.csv
|   |   |           ├── settings_Linear.inp
|   |   |           └── settings_Shirley.inp
|   |   ├── invoice
|   |   |   └── invoice.json
|   |   ├── main_image
|   |   |   └── gbp_rank1_numPeak2_shirley_mSG0672_result.png
|   |   ├── meta
|   |   |   └── metadata.json
|   |   ├── other_image
|   |   |   ├── BIC_vs_NumPeak.png
|   |   |   └── parameter_table.png
|   |   ├── raw
```

```
|    |    |    └── Cs2O_O1s.zip
|    |    └── structured
|    |        ├── cmd_mrs.log
|    |        ├── gbp_rank1_numPeak2_shirley_mSGO672_parameters.csv
|    |        ├── settings_Linear.inp
|    |        └── settings_Shirley.inp
|    └── invoice_org.json
└── thumbnail
    └── gbp_rank1_numPeak2_shirley_mSGO672_result.png

44 directories, 47 files
```

> tempフォルダは、invoice_org.jsonを除き、zipファイルを展開したフォルダ/ファイルが展開されています。このフォルダは、RDE取り込み処理では無視され、取り込み対象とはならないことに注意してください。
>
> 上の例では、入力のzipファイル中の最上位フォルダは"data"になります。このフォルダは任意で、"data2/"でも、シングルデータタイルモード時と同様main_imageやstructuredフォルダが直接並ぶ形式でも構いません。つまり「zipファイルを展開したときに、なんらかのフォルダがあって、その下にraw、structuredといったフォルダがあるケースと、フォルダなしでraw、structuredといったファイルがあるケースは、いずれの場合も同じ扱い」となります。

4. マルチデータタイルモード

続いて、`マルチデータタイルモード` でのコード例を示します。

すでに示したように、"マルチデータタイルモード"は、以下のような処理を実行します。

- 複数のファイルを、それぞれ別のデータタイルとして登録する。
- そのとき利用される送り状(invoice.json)は、すべて同じものが利用される。
- ファイルは全く別のフォーマットの場合もあり、基本的にメタデータの抽出やデータの可視化など、必要最小限のものを除き"構造化処理"は行わない。

> 前章で行った、RDEフォーマットモードの、マルチデータタイル登録 とは異なり、マルチデータタイルモードでは、"1ファイル = 1データタイル"として登録されます。

4.1 RDEToolKitのインストールと初期フォルダ構成作成

『共通処理』の章で示したように、RDEToolKitのインストールと、初期フォルダ構成の作成を実施します。

```
$ cd ${HOME}/rde-sample
$ mkdir multidatatile
$ cd multidatatile/

$ source ~/venv/bin/activate
(venv) $ python -m rdetoolkit init
:
(venv) $ cd container
```

以下、双方が完了しているものとして進めます。

4.2 テスト用データ配置

テスト用のデータとして、以下を配置します。

- Mo50Ti15C-20230111-10mN-00.tif (894KB)
- f27.bmp (377KB)
- materials.svg (32KB)
- report.pdf (1.4MB)

実際に格納されると、以下の様になります。

```
(venv) $ cd container/
(venv) $ ls -l data/inputdata/
total 2720
-rw-rw-r-- 1 devel devel  894137  5月 16 07:10 Mo50Ti15C-20230111-10mN-00.tif
-rw-rw-r-- 1 devel devel  376734  5月 16 07:10 f27.bmp
```

```
-rw-rw-r-- 1 devel devel   31755  5月 16 07:10 materials.svg
-rw-rw-r-- 1 devel devel 1475968  5月 16 07:11 report.pdf
```

> 日付については、上記と異なる可能性があります。違っていても特に問題ありません。

4.2.1 フォルダ初期化

前章同様、以下の内容で初期化用スクリプトを作成します。

reinit.sh

```
#!/bin/bash


# ./data/inputdata
#  -> 何もしない

#./data/job.failed
if [ -f ./data/job.failed ];then
    echo "./data/job.failed was removed"
    rm -f ./data/job.failed
fi

# ./data/invoice
#  -> 何もしない

#./data/tasksupport
#  -> 何もしない

#./modules
#  -> 何もしない

#./requirements.txt
#  -> 何もしない

D="
./data/divided
./data/logs
./data/attachment
./data/invoice_patch
./data/meta
./data/main_image
./data/other_image
./data/raw
./data/nonshared_raw
./data/structured
./data/temp
./data/thumbnail
"
for d in ${D};do
    echo "${d} was removed"
    rm -rf ${d}
done

# Python cache
rm -rf modules/__pycache__
```

実行権限を付与します。

```
chmod a+x reinit.sh
```

4.3 invoice.schema.json、invoice.jsonファイルの配置

> 実際のところ、下記2つのファイルは、`python -m rdetoolkit init` にて作成された内容から変更していません。ここでは内容を確認すればOKです。

data/tasksupport/invoice.schema.jsonを、以下の内容で作成します。

```
{
    "version": "https://json-schema.org/draft/2020-12/schema",
    "schema_id": "https://rde.nims.go.jp/rde/dataset-templates/dataset_template_custom_sample/invoice.schema.json",
    "description": "RDEデータセットテンプレートテスト用ファイル",
    "value_type": "object",
    "required": [
        "custom",
        "sample"
    ],
    "properties": {
        "custom": {
            "type": "object",
            "label": {
                "ja": "固有情報",
                "en": "Custom Information"
            },
            "required": [
                "toy_data1"
            ],
            "properties": {
                "toy_data1": {
                    "label": {
                        "ja": "トイデータ１",
                        "en": "toy_data1"
                    },
                    "type": "string"
                },
                "sample2": {
                    "label": {
                        "ja": "トイデータ２",
                        "en": "toy_data2"
                    },
                    "type": "number"
                }
            }
        },
        "sample": {
            "type": "object",
            "label": {
                "ja": "試料情報",
                "en": "Sample Information"
            },
            "properties": {}
        }
    }
}
```

data/invoice/invoice.jsonを以下の内容で作成します。

```
{
    "datasetId": "3f976089-7b0b-4c66-a035-f48773b018e6",
    "basic": {
        "dateSubmitted": "",
        "dataOwnerId": "7e4792d1a8440bcfa08925d35e9d92b234a963449f03df441234569e",
        "dataName": "toy dataset",
        "instrumentId": null,
        "experimentId": null,
```

```
        "description": null
    },
    "custom": {
        "toy_data1": "2023-01-01",
        "toy_data2": 1.0
    },
    "sample": {
        "sampleId": "",
        "names": [
            "<Please enter a sample name>"
        ],
        "composition": null,
        "referenceUrl": null,
        "description": null,
        "generalAttributes": [],
        "specificAttributes": [],
        "ownerId": "1234567e4792d1a8440bcfa08925d35e9d92b234a963449f03df449e"
    }
}
```

4.4 構造化処理作成

ここから、実際に構造化処理プログラムを作成していきます。

> 以下"container/"フォルダからの相対パスとしてファイル名を記述します。

4.4.1 main.py

初期化にて作成されたmain.pyに以下を加えていきます。

1. "MultiDataTile"モードを使う指定
2. "modules/datasets_process.py"にある `dataset()` 関数を実行する。

> 詳細な処理については、この `dataset()` 関数から呼び出される関数にて実行することを想定します。

main.py

```python
import rdetoolkit
from rdetoolkit.config import Config, SystemSettings, MultiDataTileSettings

from modules.datasets_process import dataset

def main() -> None:
    config = Config(
        system=SystemSettings(
            extended_mode="MultiDataTile",
        ),
        multidata_tile=MultiDataTileSettings(
            ignore_errors=True
        ),
    )
    rdetoolkit.workflows.run(
        config=config,
        custom_dataset_function=dataset,
    )

if __name__ == "__main__":
    main()
```

4.4.2 modules/datasets_process.py

最初に'Hello RDE!'をプリントするだけの処理を実行し、何が起きるのかを確認します。

modules/datasets_process.py

```
from rdetoolkit.errors import catch_exception_with_message
from rdetoolkit.models.rde2types import RdeInputDirPaths, RdeOutputResourcePath


@catch_exception_with_message(error_message="ERROR: failed in data processing")
def dataset(srcpaths: RdeInputDirPaths, resource_paths: RdeOutputResourcePath) -> None:
    print('Hello RDE!')
```

実行します。

```
(venv) $ python main.py
Hello RDE!
Hello RDE!
Hello RDE!
Hello RDE!
```

"Hello RDE!"が4回出力されました。

どうして4回出力されたのでしょうか?

謎を解明するために、pprintを使って確認してみます。

> pprintは変数の構造を含めて表示することが出来ます。同様のことをデバッガを利用して実行できます。普段お使いのデバッガがある場合はそちらをご利用いただいても問題ありません。

modules/datasets_process.pyの、上の方でpprintをimportします。

```
:
from pprint import pprint
:
```

次に、`dataset()` 関数内で、srcpathsの設定内容を表示するようにします。

```
:
    print('Hello RDE!')
    pprint(srcpaths)
:
```

実行してみます。

```
(venv) $ python main.py
Hello RDE!
RdeInputDirPaths(inputdata=PosixPath('data/inputdata'),
                 invoice=PosixPath('data/invoice'),
                 tasksupport=PosixPath('data/tasksupport'),
                 config=Config(system=SystemSettings(extended_mode='MultiDataTile', save_raw=False, save_nonshared_raw=True,
save_thumbnail_image=False, magic_variable=False), multidata_tile=MultiDataTileSettings(ignore_errors=True), smarttable=None))
Hello RDE!
RdeInputDirPaths(inputdata=PosixPath('data/inputdata'),
                 invoice=PosixPath('data/invoice'),
                 tasksupport=PosixPath('data/tasksupport'),
```

```
                config=Config(system=SystemSettings(extended_mode='MultiDataTile', save_raw=False, save_nonshared_raw=True,
save_thumbnail_image=False, magic_variable=False), multidata_tile=MultiDataTileSettings(ignore_errors=True), smarttable=None))
Hello RDE!
RdeInputDirPaths(inputdata=PosixPath('data/inputdata'),
                 invoice=PosixPath('data/invoice'),
                 tasksupport=PosixPath('data/tasksupport'),
                 config=Config(system=SystemSettings(extended_mode='MultiDataTile', save_raw=False, save_nonshared_raw=True,
save_thumbnail_image=False, magic_variable=False), multidata_tile=MultiDataTileSettings(ignore_errors=True), smarttable=None))
Hello RDE!
RdeInputDirPaths(inputdata=PosixPath('data/inputdata'),
                 invoice=PosixPath('data/invoice'),
                 tasksupport=PosixPath('data/tasksupport'),
                 config=Config(system=SystemSettings(extended_mode='MultiDataTile', save_raw=False, save_nonshared_raw=True,
save_thumbnail_image=False, magic_variable=False), multidata_tile=MultiDataTileSettings(ignore_errors=True), smarttable=None))
```

同じものが4回出力されました。

まだ"なぜ4回出力されるのか?"が不明です。今度はsrcpathsの代わりに、もう1つの引数であるresource_pathsを表示してみます。

```
:
    print('Hello RDE!')
    #pprint(srcpaths) #この行をコメントアウト
    pprint(resource_paths) #この行を追加
:
```

実行してみます。

```
(venv) $ python main.py
Hello RDE!
RdeOutputResourcePath(raw=PosixPath('data/raw'),
                      nonshared_raw=PosixPath('data/nonshared_raw'),
                      rawfiles=(PosixPath('data/inputdata/Mo5OTi15C-20230111-10mN-00.tif'),),
                      struct=PosixPath('data/structured'),
                      main_image=PosixPath('data/main_image'),
                      other_image=PosixPath('data/other_image'),
                      meta=PosixPath('data/meta'),
                      thumbnail=PosixPath('data/thumbnail'),
                      logs=PosixPath('data/logs'),
                      invoice=PosixPath('data/invoice'),
                      invoice_schema_json=PosixPath('data/tasksupport/invoice.schema.json'),
                      invoice_org=PosixPath('data/temp/invoice_org.json'),
                      temp=PosixPath('data/temp'),
                      invoice_patch=PosixPath('data/invoice_patch'),
                      attachment=PosixPath('data/attachment'))
Hello RDE!
RdeOutputResourcePath(raw=PosixPath('data/divided/0001/raw'),
                      nonshared_raw=PosixPath('data/divided/0001/nonshared_raw'),
                      rawfiles=(PosixPath('data/inputdata/f27.bmp'),),
                      struct=PosixPath('data/divided/0001/structured'),
                      main_image=PosixPath('data/divided/0001/main_image'),
                      other_image=PosixPath('data/divided/0001/other_image'),
                      meta=PosixPath('data/divided/0001/meta'),
                      thumbnail=PosixPath('data/divided/0001/thumbnail'),
                      logs=PosixPath('data/divided/0001/logs'),
                      invoice=PosixPath('data/divided/0001/invoice'),
                      invoice_schema_json=PosixPath('data/tasksupport/invoice.schema.json'),
                      invoice_org=PosixPath('data/temp/invoice_org.json'),
                      temp=PosixPath('data/divided/0001/temp'),
                      invoice_patch=PosixPath('data/divided/0001/invoice_patch'),
                      attachment=PosixPath('data/divided/0001/attachment'))
Hello RDE!
RdeOutputResourcePath(raw=PosixPath('data/divided/0002/raw'),
                      nonshared_raw=PosixPath('data/divided/0002/nonshared_raw'),
                      rawfiles=(PosixPath('data/inputdata/materials.svg'),),
```

```
                        struct=PosixPath('data/divided/0002/structured'),
                        main_image=PosixPath('data/divided/0002/main_image'),
                        other_image=PosixPath('data/divided/0002/other_image'),
                        meta=PosixPath('data/divided/0002/meta'),
                        thumbnail=PosixPath('data/divided/0002/thumbnail'),
                        logs=PosixPath('data/divided/0002/logs'),
                        invoice=PosixPath('data/divided/0002/invoice'),
                        invoice_schema_json=PosixPath('data/tasksupport/invoice.schema.json'),
                        invoice_org=PosixPath('data/temp/invoice_org.json'),
                        temp=PosixPath('data/divided/0002/temp'),
                        invoice_patch=PosixPath('data/divided/0002/invoice_patch'),
                        attachment=PosixPath('data/divided/0002/attachment'))
Hello RDE!
RdeOutputResourcePath(raw=PosixPath('data/divided/0003/raw'),
                        nonshared_raw=PosixPath('data/divided/0003/nonshared_raw'),
                        rawfiles=(PosixPath('data/inputdata/report.pdf'),),
                        struct=PosixPath('data/divided/0003/structured'),
                        main_image=PosixPath('data/divided/0003/main_image'),
                        other_image=PosixPath('data/divided/0003/other_image'),
                        meta=PosixPath('data/divided/0003/meta'),
                        thumbnail=PosixPath('data/divided/0003/thumbnail'),
                        logs=PosixPath('data/divided/0003/logs'),
                        invoice=PosixPath('data/divided/0003/invoice'),
                        invoice_schema_json=PosixPath('data/tasksupport/invoice.schema.json'),
                        invoice_org=PosixPath('data/temp/invoice_org.json'),
                        temp=PosixPath('data/divided/0003/temp'),
                        invoice_patch=PosixPath('data/divided/0003/invoice_patch'),
                        attachment=PosixPath('data/divided/0003/attachment'))
```

resource_pathsには、出力されるフォルダや出力されるファイルが入っています。

少しわかりにくいので、この中からさらに"rawfiles"の値だけを取り出してみます。

```
:
    #pprint(resource_paths)  #この行はコメントアウト
    pprint(resource_paths.rawfiles)
:
```

実行すると以下の様になります。

```
(venv) $ python main.py
Hello RDE!
(PosixPath('data/inputdata/Mo50Ti15C-20230111-10mN-00.tif'),)
Hello RDE!
(PosixPath('data/inputdata/f27.bmp'),)
Hello RDE!
(PosixPath('data/inputdata/materials.svg'),)
Hello RDE!
(PosixPath('data/inputdata/report.pdf'),)
```

そうです、"Hello RDE!"が4回表示されたのは、入力ファイル、つまり `data/inputdata/` フォルダ内に存在するファイルが4つ有り、そのそれぞれに対して構造化処理プログラムが実行されるため、なのです。

> `resource_paths.rawfiles` は、上記の様に入力ファイルが1個の場合でも配列として提供されます。構造化処理プログラム内で利用する場合は注意してください。

この"単純な"プログラムの結果どうなったかを確認してみます。

```
(venv) $ tree data
data
├── attachment
```

```
├── divided
│   ├── 0001
│   │   ├── attachment
│   │   ├── invoice
│   │   │   └── invoice.json
│   │   ├── invoice_patch
│   │   ├── logs
│   │   ├── main_image
│   │   ├── meta
│   │   ├── nonshared_raw
│   │   │   └── f27.bmp
│   │   ├── other_image
│   │   ├── raw
│   │   ├── structured
│   │   ├── temp
│   │   └── thumbnail
│   ├── 0002
│   │   ├── attachment
│   │   ├── invoice
│   │   │   └── invoice.json
│   │   ├── invoice_patch
│   │   ├── logs
│   │   ├── main_image
│   │   ├── meta
│   │   ├── nonshared_raw
│   │   │   └── materials.svg
│   │   ├── other_image
│   │   ├── raw
│   │   ├── structured
│   │   ├── temp
│   │   └── thumbnail
│   └── 0003
│       ├── attachment
│       ├── invoice
│       │   └── invoice.json
│       ├── invoice_patch
│       ├── logs
│       ├── main_image
│       ├── meta
│       ├── nonshared_raw
│       │   └── report.pdf
│       ├── other_image
│       ├── raw
│       ├── structured
│       ├── temp
│       └── thumbnail
├── inputdata
│   ├── Mo50Ti15C-20230111-10mN-00.tif
│   ├── f27.bmp
│   ├── materials.svg
│   └── report.pdf
├── invoice
│   └── invoice.json
├── invoice_patch
├── logs
│   └── rdesys.log
├── main_image
├── meta
├── nonshared_raw
│   └── Mo50Ti15C-20230111-10mN-00.tif
├── other_image
├── raw
├── structured
├── tasksupport
│   ├── invoice.schema.json
│   └── metadata-def.json
├── temp
│   └── invoice_org.json
└── thumbnail
```

```
55 directories, 16 files
```

`data/inputdata/` フォルダには、先に用意した4つのファイルがあることが分かります。

```
:
├── inputdata
│   ├── Mo50Ti15C-20230111-10mN-00.tif
│   ├── f27.bmp
│   ├── materials.svg
│   └── report.pdf
```

主な出力先の1つである"nonshared_raw/"フォルダについて確認すると * data/nonshared_raw

の他に

- data/divided/0001/nonshared_raw
- data/divided/0002/nonshared_raw
- data/divided/0003/nonshared_raw

の3つがあることが分かります。

そして、そのそれぞれに、入力ファイルが1つずつ格納されています。

> 入力ファイルを、`nonshared_raw/` フォルダに出力する機能は、RDEToolKitにより標準提供されている機能です。
> `nonshared_raw/` フォルダではなく、`raw/` フォルダに出力したい場合など、デフォルトの挙動を変更するには、(main.pyで与えるConfig()内の指定などの)設定内容を変更する必要があります。

4.5 イメージ処理

上述のように、この例で使われる入力ファイルはtif形式、bmp形式、svg形式およびpdf形式とそれぞれ異なっています。

tif形式(= tiff形式)やbmp形式(BitMap形式)、jpg形式(=JPEG形式)などの画像イメージは問題なくRDE内で利用可能である一方、SVG形式では、そのまま画像用のフォルダ(`main_image/` あるいは `other_image/`)に格納した場合、格納は可能であっても、登録されたデータのダウンロードに支障が発生してしまいます。

> 現時点(2025.05)のRDEの制限となっています。RDEの将来バージョンでSVG形式がサポートされた場合はこの限りではありません。

またpdf形式は画像形式ではないので、何らかの方法で画像化する必要があります。

> 画像化しなくても"No Image"という画像が表示されるだけですので、厳密には必須ではありませんが、画像が表示された方がわかりやすいなどの理由から画像化することを考えることにします。

マルチデータタイルモードでは、1ファイルが1データタイルとして扱われるので、それぞれのファイルから `main_image/` フォルダに格納する"画像"ファイルを生成します。

4.5.1 入力ファイル形式判断

入力ファイルの形式を判断するために、python-magic モジュールを利用します。

> 同様のモジュールとして、Python標準モジュールのmimetypesがあります。mimetypesは、ファイル名拡張子によって形式を返します。つまりファイル名に拡張子が付いていないファイルについて、その形式が判断できません。そのため、ここではpython-magicモジュールを使うことにします。

python-magicモジュールの実行には、libmagicが必要となります。

インストール状況を確認します。テストしたUbuntu 24.04の場合、以下のようになります。

```
(venv) $ sudo apt list --installed | grep libmagic*
:
libmagic-mgc/noble,now 1:5.45-3build1 amd64 [インストール済み、自動]
libmagic1t64/noble,now 1:5.45-3build1 amd64 [インストール済み、自動]
```

つまり、libmagicパッケージが既にインストールされた状態でした。インストールされていることが確認できない場合は、それぞれの環境に合わせてlibmagic(または相当する)パッケージをインストールしてください。

続いて、Pythonの `python-magic` モジュールをインストールします。

```
pip install python-magic
```

確認します。

```
(venv) $ pip list
Package                   Version
------------------------- -----------------
:
python-magic              0.4.27
:
```

> 上記のように `python-magic` が表示されればインストールされていることが確認できます。

スクリプト(modules/datasets_process.py)の先頭で、`magic` をインポートします。

> パッケージ名は `python-magic` ですが、importする際は `magic` だけとなります。

modules/datasets_process.py

```
import magic
```

ファイル出力先を表示するようにします。以下、全文を提示します。

modules/datasets_process.py

```
from pprint import pprint

import magic
from rdetoolkit.errors import catch_exception_with_message
from rdetoolkit.models.rde2types import RdeInputDirPaths, RdeOutputResourcePath
```

```
@catch_exception_with_message(error_message="ERROR: failed in data processing")
def dataset(srcpaths: RdeInputDirPaths, resource_paths: RdeOutputResourcePath) -> None:
    image_to = resource_paths.main_image
    pprint(image_to)
```

実行します。

```
(venv) $ python main.py
PosixPath('data/main_image')
PosixPath('data/divided/0001/main_image')
PosixPath('data/divided/0002/main_image')
PosixPath('data/divided/0003/main_image')
```

このように、今回の場合、image_toは、ファイルそれぞれで以下の様になります。

- data/main_image
- data/divided/0001/main_image
- data/divided/0002/main_image
- data/divided/0003/main_image

> 2個目以降のフォルダ名を構造化処理プログラム内で都度生成するのは大変なので、上の様にRDEToolKitが提供している機能を利用します。

次に、それぞれのファイル名を取得し、python-magicモジュールによる判定を行います。

modules/datasets_process.py

```
from pprint import pprint

import magic
from rdetoolkit.errors import catch_exception_with_message
from rdetoolkit.models.rde2types import RdeInputDirPaths, RdeOutputResourcePath


@catch_exception_with_message(error_message="ERROR: failed in data processing")
def dataset(srcpaths: RdeInputDirPaths, resource_paths: RdeOutputResourcePath) -> None:
    image_to = resource_paths.main_image
    for inputfile in resource_paths.rawfiles:
        mymime = magic.from_file(inputfile, mime=True)
        pprint(mymime)
```

実行します。

```
(venv) $ python main.py
'image/tiff'
'image/bmp'
'image/svg+xml'
'application/pdf'
```

それぞれのファイルの形式が正しく判断されていることが分かります。

4.5.2 ファイルコピー または 画像ファイルへの変換

続いて、判定した形式(=mymime)の値に応じて処理を実行します。RDEが画像としてサポートしている形式であればそのままコピーします。それ以外の形式であれば、何らかの変換処理を実施し、画像として登録します。

"image/jpeg"、**"image/png"** または **"image/gif"**の場合

変換操作など実行せずに、main_imageフォルダにコピーする。コピーにはshutilパッケージを利用します。

modules/datasets_process.py

```
import shutil
:

        if mymime == "image/jpeg" or mymime == "image/png" or mymime == "image/gif":
            shutil.copy(inputfile, image_to)
```

> 今回は、この3つの形式のファイルではないので、このif文では何も実行されません。

"application/pdf"の場合

pdfファイルの1ページ目の見た目を画像として生成します。

様々な方法がありますが、PyMuPDFを使うことにします。

Pythonモジュールをインストールします。

```
pip install PyMuPDF
```

インポートします。このモジュールは、インストール時に指定するパッケージ名とインポートする名称が大きく異なります(→"fitz")ので注意してください。またpdfファイル名から拡張子を取り除き、新たな拡張子(.png)を付けて保存するため、pathlibもインポートしておきます。

modules/datasets_process.py

```
from pathlib import Path
import fitz
:
```

変換処理は、上のif文に続く形とします。

```
:
        elif mymime == "application/pdf":
            base_file_name = inputfile.stem

            # PDFファイルを開く
            pdf = fitz.open(inputfile)

            # 1ページずつ画像ファイルとして保存する
            for i, page in enumerate(pdf):
                page_num = i + 1 # iは0から始まるため、1足してページ数とする。
                pix = page.get_pixmap()
                image_f_name = f'{base_file_name}_{page_num:02}.png'
                image_f_path = image_to / image_f_name
```

```
                pix.save(image_f_path)
                # 1 ページだけでよいので、ここで抜ける
                break
```

> 上の例では、pdfのファイル名から拡張子を除き、"_"(アンダースコア)に続きページ番号を前ゼロ埋め2桁(つまり"01")、さらに拡張子(.png)を付加して画像ファイル名としています。

"image/svg+xml"の場合

前述の様に、現時点のRDEは、画像形式としてのsvg形式をサポートしていません。そのため、png形式などRDEがサポートしている形式に変換する必要があります。

ここでは、svg形式をpng形式に変換することとし、変換に `cairosvg` を利用することにします。

Pythonモジュールをインストールします。

```
pip install cairosvg
```

OSの必要パッケージもインストールします。

```
sudo apt install libcairo2
```

インポートします。

modules/datasets_process.py

```
import os
import cairosvg
:
```

> 前者、つまりosパッケージのインポートは、Pythonスクリプト内で 環境変数 を設定するために必要となります。

変換処理は、上のif文に続く形とします。

modules/datasets_process.py

```
:
        elif mymime == "image/svg+xml":
            base_file_name = inputfile.stem
            image_f_name = base_file_name + '.png'
            os.environ['LC_CTYPE'] = "ja_JP.UTF-8"
            cairosvg.svg2png(url=str(inputfile),write_to=str(image_to / image_f_name))
```

> プログラム中で環境変数 `$LC_CTYPE` の設定を行っています。この設定は必須ではありませんが、英語環境で日本語文字列を含むsvg形式のファイルを変換する際には文字化けを避けるために設定します。

その他の画像形式の場合 (tif形式、bmp形式を含む)

その他の形式の場合は、PILを用いてpng画像への変換を試みます。PILサポート外の画像形式や画像以外のファイル形式(例えばCSV形式など)の場合は、何もしないようにします。

変換処理は、上のif文に続く形とします。

```
:
from PIL import Image
:
        else:
            try:
                base_file_name = inputfile.stem
                image_f_name = base_file_name + '.png'
                img = Image.open(inputfile)
                img.save(str(image_to / image_f_name))
            except Exception:
                pass
```

> 場合によっては、png画像に変換出来なかった場合に raise StructuredError(.....) を発行することを考慮してください。

以上をまとめると、以下のようになります。

modules/datasets_process.py

```
import magic
import shutil
import os
from pathlib import Path

import fitz
import cairosvg
from PIL import Image
from rdetoolkit.errors import catch_exception_with_message
from rdetoolkit.models.rde2types import RdeInputDirPaths, RdeOutputResourcePath


@catch_exception_with_message(error_message="ERROR: failed in data processing")
def dataset(srcpaths: RdeInputDirPaths, resource_paths: RdeOutputResourcePath) -> None:
    image_to = resource_paths.main_image
    for inputfile in resource_paths.rawfiles:
        mymime = magic.from_file(inputfile, mime=True)
        if mymime == "image/jpeg" or mymime == "image/png" or mymime == "image/gif":
            shutil.copy(inputfile, image_to)
        elif mymime == "application/pdf":
            base_file_name = inputfile.stem

            # PDFファイルを開く
            pdf = fitz.open(inputfile)

            # 1ページずつ画像ファイルとして保存する
            for i, page in enumerate(pdf):
                page_num = i + 1 # iは0から始まるため、1足してページ数とする。
                pix = page.get_pixmap()
                image_f_name = f'{base_file_name}_{page_num:02}.png'
                image_f_path = image_to / image_f_name
                pix.save(image_f_path)
                # 1 ページだけでよいので、ここで抜ける
                break
        elif mymime == "image/svg+xml":
            base_file_name = inputfile.stem
            image_f_name = base_file_name + '.png'
            os.environ['LC_CTYPE'] = "ja_JP.UTF-8"
            cairosvg.svg2png(url=str(inputfile),write_to=str(image_to / image_f_name))
        else:
            try:
                base_file_name = inputfile.stem
                image_f_name = base_file_name + '.png'
                img = Image.open(inputfile)
```

```
                img.save(str(image_to / image_f_name))
            except Exception:
                pass
```

実行します。

```
python main.py
```

出来上がったフォルダ構成を確認してみます。

```
(venv) $ tree data
data
├── attachment
├── divided
│   ├── 0001
│   │   ├── attachment
│   │   ├── invoice
│   │   │   └── invoice.json
│   │   ├── invoice_patch
│   │   ├── logs
│   │   ├── main_image
│   │   │   └── f27.png
│   │   ├── meta
│   │   ├── nonshared_raw
│   │   │   └── f27.bmp
│   │   ├── other_image
│   │   ├── raw
│   │   ├── structured
│   │   ├── temp
│   │   └── thumbnail
│   ├── 0002
│   │   ├── attachment
│   │   ├── invoice
│   │   │   └── invoice.json
│   │   ├── invoice_patch
│   │   ├── logs
│   │   ├── main_image
│   │   │   └── materials.png
│   │   ├── meta
│   │   ├── nonshared_raw
│   │   │   └── materials.svg
│   │   ├── other_image
│   │   ├── raw
│   │   ├── structured
│   │   ├── temp
│   │   └── thumbnail
│   └── 0003
│       ├── attachment
│       ├── invoice
│       │   └── invoice.json
│       ├── invoice_patch
│       ├── logs
│       ├── main_image
│       │   └── report_01.png
│       ├── meta
│       ├── nonshared_raw
│       │   └── report.pdf
│       ├── other_image
│       ├── raw
│       ├── structured
│       ├── temp
│       └── thumbnail
├── inputdata
│   ├── Mo50Ti15C-20230111-10mN-00.tif
│   ├── f27.bmp
│   ├── materials.svg
```

```
│   └── report.pdf
├── invoice
│   └── invoice.json
├── invoice_patch
├── logs
│   └── rdesys.log
├── main_image
│   └── Mo50Ti15C-20230111-10mN-00.png
├── meta
├── nonshared_raw
│   └── Mo50Ti15C-20230111-10mN-00.tif
├── other_image
├── raw
├── structured
├── tasksupport
│   ├── invoice.schema.json
│   └── metadata-def.json
├── temp
│   └── invoice_org.json
└── thumbnail

55 directories, 20 files
```

> inputdata/ フォルダにある、tiff形式、bmp形式、svg形式およびpdf形式ファイルは、それぞれpng画像に変換され、main_image/ フォルダに格納されます。本章の例としては扱っていませんがjpeg形式、gif形式およびpng形式は(変換などの作業を行わず)そのまま main_image/ フォルダに複製が格納されます。

4.6 非共有フォルダ格納と共有フォルダへの格納

RDEToolKitのデフォルト設定では、入力ファイルは非共有フォルダ(→ `nonshared_raw/`)へのコピー処理が実行されます。そのため非共有フォルダへの格納のために処理を追加する必要は有りません。

逆に共有フォルダ(raw/フォルダ)へ書き込み対場合は、設定を変更する必要があります。

以下に、非共有フォルダではなく、共有フォルダにコピーする場合の `main.py` の例を示します。

```python
import rdetoolkit
from rdetoolkit.config import Config, SystemSettings, MultiDataTileSettings

from modules.datasets_process import dataset

def main() -> None:
    config = Config(
        system=SystemSettings(
            extended_mode="MultiDataTile",
            save_nonshared_raw=False,
            save_raw=True,
        ),
        multidata_tile=MultiDataTileSettings(
            ignore_errors=True
        ),
    )
    rdetoolkit.workflows.run(
        config=config,
        custom_dataset_function=dataset,
    )
```

```
if __name__ == "__main__":
    main()
```

save_nonshared_raw に False をセットすることで、nonshared_raw/フォルダへのコピーを抑止し、save_raw に True をセットすることでraw/フォルダへのコピーを実施するようにしています。

また別述のように、RDEToolKitの設定は外部ファイル(tasksupport/rdeconfig.yml など)で指定することも可能です。詳しくは関連する章や別途公開されているオンラインマニュアルなどを参照ください。

5. エクセルインボイスモード

"エクセルインボイスモード"は、以下のような処理を実行します。

- 送り状(invoice.json)の代わりに、同等の内容を含むExcel形式ファイルを使って、複数のデータを一度に登録する。
- 1つのデータタイルに登録されるファイルは、1個でも複数でも構わない。ただし、複数個の場合は、zip圧縮して1つのファイルにしておく必要がある。
- 構造化処理は、通常の"インボイスモード"の場合と同様に実装される。
- 1つのデータタイルに登録される送り状データは、Excelファイルの1行に記述される。
- 画面から入力された送り状データ(invoice.json)は、Excelファイルの内容で上書きされる。Excelファイル内で指定がない(空欄の)項目は画面入力値が利用される。

5.1 エクセルインボイスモード利用時の開発概要

エクセルインボイスモードを利用した構造化処理プログラムの開発は以下のような流れで実施します。

1. (通常の)インボイスモードとして構造化処理プログラムを開発する。
2. 上で作成した送り状設定ファイル(invoice.schema.json)の形式に合わせて、エクセルインボイスファイル(xlsx形式)を作成する。必要であればinvoice.jsonの内容を、このExcelファイルに転記する。
3. 上記エクセルインボイスで指定したファイル(入力ファイル)を作成する。
4. エクセルインボイスモードとしての動作確認

> 構造化処理プログラムとしては、"インボイスモード"と"エクセルインボイスモード"での違いはありません。
> RDEToolKitが自動的に判別し、適切に前処理を実行してくれるためです。

5.2 準備

5.2.1 RDEToolKitのインストールと初期フォルダ構成作成

『共通処理』の章で示したように、RDEToolKitのインストールと、初期フォルダ構成の作成を実施します。

```
$ cd ${HOME}/rde-sample
$ mkdir excelinvoice
$ cd excelinvoice/

$ source ~/venv/bin/activate

(venv) $ python -m rdetoolkit init
:
(venv) $ cd container
```

以下、双方が完了しているものとして進めます。

5.2.2 フォルダ初期化

前章同様、以下の内容で初期化用スクリプトを作成します。

reinit.sh

```
#!/bin/bash


# ./data/inputdata
#  -> 何もしない

#./data/job.failed
if [ -f ./data/job.failed ];then
    echo "./data/job.failed was removed"
    rm -f ./data/job.failed
fi

# ./data/invoice
#  -> 何もしない

#./data/tasksupport
#  -> 何もしない

#./modules
#  -> 何もしない

#./requirements.txt
#  -> 何もしない

D="
./data/divided
./data/logs
./data/attachment
./data/invoice_patch
./data/meta
./data/main_image
./data/other_image
./data/raw
./data/nonshared_raw
./data/structured
./data/temp
./data/thumbnail
"
for d in ${D};do
    echo "${d} was removed"
    rm -rf ${d}
done

# Python cache
rm -rf modules/__pycache__
```

実行権限を付与します。

```
chmod a+x reinit.sh
```

5.3 インボイスモードの開発 (その1)

"その1"として、'Hello RDE!'を表示するだけの構造化処理プログラムを作成実行することにより、入力ファイルが非公開(または公開)フォルダにコピーされるところまでを実装、確認します。

5.3.1 テスト用データ配置

テスト用のデータとして、以下を配置します。

- sample-data.txt (0KB)

0バイトのファイルですので、`touch`コマンドにより設置します。

```
(venv) $ touch data/inputdata/sample-data.txt
```

実際に格納されると、以下の様になります。

```
(venv) $ ls -l data/inputdata/
total 0
-rw-r--r-- 1 devel devel 0  9月 29 09:55 sample-data.txt
```

> 上記サンプルデータは"0バイト"のファイルです。この例ではファイルの読み込み処理を行わないダミー実装のためです。nonshared_raw/ フォルダへのコピーのみ実行されます。

5.3.2 構造化処理作成

ここから、実際に構造化処理プログラムを作成していきます。

main.py

初期化にて作成されたmain.pyを以下のように変更します。

main.py

```
import rdetoolkit

from modules import datasets_process


def main() -> None:
    rdetoolkit.workflows.run(
        custom_dataset_function=datasets_process.dataset,
    )

if __name__ == "__main__":
    main()
```

5.3.3 modules/datasets_process.py

modules/datasets_process.py

```
from rdetoolkit.errors import catch_exception_with_message
from rdetoolkit.models.rde2types import RdeInputDirPaths, RdeOutputResourcePath


@catch_exception_with_message(error_message="ERROR: failed in data processing")
def dataset(srcpaths: RdeInputDirPaths, resource_paths: RdeOutputResourcePath) -> None:
    print('Hello RDE!')
```

実行してみます。

```
(venv) $ python main.py
Hello RDE!
```

想定通りに表示されました。

出力データを確認します。

```
(venv) $ tree data
data
├── attachment
├── inputdata
│   └── sample-data.txt
├── invoice
│   └── invoice.json
├── invoice_patch
├── logs
│   └── rdesys.log
├── main_image
├── meta
├── nonshared_raw
│   └── sample-data.txt
├── other_image
├── raw
├── structured
├── tasksupport
│   ├── invoice.schema.json
│   └── metadata-def.json
├── temp
└── thumbnail

15 directories, 6 files
```

`sample-data.txt` が、想定通り `nonshared_raw/` フォルダにコピーされていることが確認できます。

5.4 インボイスモードの開発 (その2)

次に、実働する構造化処理プログラムを作成し、送り状ファイル(invoice.json)などの関連するファイルを作成していきます。

先に関連するファイル群を作成していきます。

5.4.1 invoice.json

送り状ファイルを用意します。

data/invoice/invoice.json

```
{
    "datasetId": "e751fcc4-b926-4747-b236-cab40316fc49",
    "basic": {
        "dateSubmitted": "2023-03-14",
        "dataOwnerId": "16b51e451d6be669bb93b551420ec005b1df55b83530316137643764",
        "dataName": "test_spe1",
        "instrumentId": null,
```

```
        "experimentId": "test_230606_1",
        "description": "test_today"
    },
    "custom": {
        "key1": "test1",
        "key2": "test2",
        "key3": "test3",
        "key4": "test4",
        "key5": "test5"
    },
    "sample": {
        "sampleId": "8ea50153-7a87-44b7-9d6e-10179c507039",
        "names": [
            "testdev01_1"
        ],
        "composition": "AB2C3",
        "referenceUrl": "test_ref",
        "description": null,
        "generalAttributes": [
            {
                "termId": "3adf9874-7bcb-e5f8-99cb-3d6fd9d7b55e",
                "value": "test_ABC"
            },
            {
                "termId": "e2d20d02-2e38-2cd3-b1b3-66fdb8a11057",
                "value": "b"
            },
            {
                "termId": "efcf34e7-4308-c195-6691-6f4d28ffc9bb",
                "value": "c"
            },
            {
                "termId": "7cc57dfb-8b70-4b3a-5315-fbce4cbf73d0",
                "value": "d"
            },
            {
                "termId": "1e70d11d-cbdd-bfd1-9301-9612c29b4060",
                "value": "e"
            },
            {
                "termId": "5e166ac4-bfcd-457a-84bc-8626abe9188f",
                "value": "f"
            },
            {
                "termId": "0d0417a3-3c3b-496a-b0fb-5a26f8a74166",
                "value": "g"
            }
        ],
        "ownerId": "de17c7b3f0ff5126831c2d519f481055ba466ddb6238666132316439"
    }
}
```

一部の項目は、`invoice.schema.json` ではないファイルにて定義されています。例えば `"ownerId": "test_owner"` のような指定をすると、バリデーションエラーとなります。

これは `(RDEToolKitのインストールフォルダ)/static/invoice_basic_and_sample.schema_.json` で指定している、以下の内容に合致していないために出力されます。

```
:
                "ownerId": {
                    "description": "sample owner id",
                    "type": "string",
                    "pattern": "^([0-9a-zA-Z]{56})$"
                },
:
```

`invoice.schema.json` の指定と比べて問題がないのにエラーが発生する場合は、上記ファイルの設定と合致していない可能性がありますので、そちらも合わせて確認してください。

5.4.2 invoice.schema.json

> 別途提供されている"Excelファイルからjson生成"するツールで作成した `invoice.schema.json` は"description"部分が入らない(→ "None"が指定される)ので、出来上がったjsonファイルを手動にて修正する必要があります(下記は修正済み)。

data/tasksupport/invoice.schema.json

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "NIMS_ELN_Process_Pulsed_laser_deposition_Magnetic_thin_film",
  "description": "固有情報と試料情報のスキーマ",
  "type": "object",
  "required": [
      "custom",
      "sample"
  ],
  "properties": {
      "custom": {
          "type": "object",
          "label": {
              "ja": "固有情報",
              "en": "Custom Information"
          },
          "required": [],
          "properties": {
              "key1": {
                  "label": {
                      "ja": "キー1",
                      "en": "key1"
                  },
                  "type": "string"
              },
              "key2": {
                  "label": {
                      "ja": "キー2",
                      "en": "key2"
                  },
                  "type": "string"
              },
              "key3": {
                  "label": {
                      "ja": "キー3",
                      "en": "key3"
                  },
                  "type": "string"
              },
              "key4": {
                  "label": {
                      "ja": "キー4",
                      "en": "key4"
                  },
                  "type": "string"
              },
              "key5": {
                  "label": {
                      "ja": "キー5",
                      "en": "key5"
                  },
                  "type": "string"
```

```
            }
        }
    },
    "sample": {
      "type": "object",
      "label": {
          "ja": "試料情報",
          "en": "Sample Information"
      },
      "required": [
      ],
      "properties": {
          "generalAttributes": {
              "type": "array",
              "items": [
                  {
                      "type": "object",
                      "required": [
                          "termId"
                      ],
                      "properties": {
                          "termId": {
                              "const": "3adf9874-7bcb-e5f8-99cb-3d6fd9d7b55e"
                          }
                      }
                  },
                  {
                      "type": "object",
                      "required": [
                          "termId"
                      ],
                      "properties": {
                          "termId": {
                              "const": "e2d20d02-2e38-2cd3-b1b3-66fdb8a11057"
                          }
                      }
                  },
                  {
                      "type": "object",
                      "required": [
                          "termId"
                      ],
                      "properties": {
                          "termId": {
                              "const": "efcf34e7-4308-c195-6691-6f4d28ffc9bb"
                          }
                      }
                  },
                  {
                      "type": "object",
                      "required": [
                          "termId"
                      ],
                      "properties": {
                          "termId": {
                              "const": "7cc57dfb-8b70-4b3a-5315-fbce4cbf73d0"
                          }
                      }
                  },
                  {
                      "type": "object",
                      "required": [
                          "termId"
                      ],
                      "properties": {
                          "termId": {
                              "const": "1e70d11d-cbdd-bfd1-9301-9612c29b4060"
                          }
                      }
                  },
```

```
                    {
                        "type": "object",
                        "required": [
                            "termId"
                        ],
                        "properties": {
                            "termId": {
                                "const": "5e166ac4-bfcd-457a-84bc-8626abe9188f"
                            }
                        }
                    },
                    {
                        "type": "object",
                        "required": [
                            "termId"
                        ],
                        "properties": {
                            "termId": {
                                "const": "0d0417a3-3c3b-496a-b0fb-5a26f8a74166"
                            }
                        }
                    }
                ]
            }
        }
    }
  }
}
```

5.4.3 metadata-def.json

metadata-def.jsonを以下の内容で作成します。

data/tasksupport/metadata-def.json

```
{
    "file_size": {
        "name": {
            "ja": "ファイルサイズ",
            "en": "file size"
        },
        "schema": {
            "type": "number"
        },
        "description": "ファイルサイズ",
        "variable":1,
        "unit" : "byte"
    },
    "file_created": {
        "name": {
            "ja": "ファイル作成日",
            "en": "file created at"
        },
        "schema": {
            "type": "string",
            "format": "date"
        },
        "description": "ファイル作成日"
    },
    "file_name": {
        "name": {
            "ja": "ファイル名",
            "en": "file name"
        },
        "schema": {
```

```
            "type": "string"
        },
        "description": "ファイル名"
    }
}
```

> "file_size"を繰り返し有りメタデータ項目(→ "variable":1)として定義しています。入力ファイルは1つですので本来は通常のメタデータ項目ですが、ここでは例を示す意味で"繰り返し有り"としています。繰り返しのないメタデータ項目として扱う場合は"variable: 1"の行を削除します。

続いて、構造化処理を実行するようにプログラムを作成していきます。

> ここではサンプルを表示することを主題にしていますので、一部の処理が意味の無い処理のままとなっています。

5.4.4 modules/interfaces.py

```python
from __future__ import annotations

from abc import ABC, abstractmethod
from pathlib import Path
from typing import Any, Generic, TypeVar

import pandas as pd
from rdetoolkit.models.rde2types import MetaType, RepeatedMetaType
from rdetoolkit.rde2util import Meta

T = TypeVar("T")


class IInputFileParser(ABC):
    @abstractmethod
    def read(self, path: Path) -> Any:  # noqa: D102
        raise NotImplementedError


class IStructuredDataProcessor(ABC):
    @abstractmethod
    def to_csv(self, dataframe: pd.DataFrame, save_path: Path, *, header: list[str] | None = None) -> Any:  # noqa: D102
        raise NotImplementedError


class IMetaParser(Generic[T], ABC):
    @abstractmethod
    def parse(self, data: T) -> Any:  # noqa: D102
        raise NotImplementedError

    @abstractmethod
    def save_meta(
        self, save_path: Path, meta: Meta, *, const_meta_info: MetaType | None = None, repeated_meta_info: RepeatedMetaType | None =
None,
    ) -> Any:
        raise NotImplementedError


class IGraphPlotter(Generic[T], ABC):
    @abstractmethod
    def plot(self, data: T, save_path: Path, *, title: str | None = None, xlabel: str | None = None, ylabel: str | None = None) ->
Any:  # noqa: D102
        raise NotImplementedError
```

5.4.5 modules/inputfile_handler.py

```python
from __future__ import annotations

from pathlib import Path
from typing import Any
from datetime import datetime

import pandas as pd
from rdetoolkit.models.rde2types import MetaType

from modules.interfaces import IInputFileParser


class FileReader(IInputFileParser):
    def read(self, srcpath: Path) -> tuple[MetaType, pd.DataFrame]:
        file_size = srcpath.stat().st_size
        file_created = datetime.fromtimestamp(srcpath.stat().st_mtime).strftime("%Y-%m-%d")
        self.data = pd.DataFrame([[1, 11], [2, 22], [3, 33]]) # Caution! dummy data
        self.meta: dict[str, str | int | float | list[Any] | bool] = {
            "file_size": file_size,
            "file_created": file_created,
            "file_name": srcpath.name
        }

        return self.meta, self.data
```

上記例では、ソース中のコメントにもあるように、返却されるdataは(固定の)ダミーデータです。

5.4.6 modules/meta_handler.py

```python
from __future__ import annotations

from pathlib import Path
from typing import Any

from rdetoolkit import rde2util
from rdetoolkit.models.rde2types import MetaType, RepeatedMetaType

from modules.interfaces import IMetaParser


class MetaParser(IMetaParser[MetaType]):
    def parse(self, data: MetaType) -> tuple[MetaType, RepeatedMetaType]:
        """Parse and extract constant and repeated metadata from the provided data."""
        # dummy return
        self.const_meta_info: MetaType = data
        self.repeated_meta_info: RepeatedMetaType = {"key": ["key_value1", "key_value2"], "sample": ["sample_value1",
"sample_value2"]}
        return self.const_meta_info, self.repeated_meta_info

    def save_meta(  # noqa: ANN201
        self,
        save_path: Path,
        metaobj: rde2util.Meta,
        *,
        const_meta_info: MetaType | None = None,
        repeated_meta_info: RepeatedMetaType | None = None,
    ) -> Any:
        """Save parsed metadata to a file using the provided Meta object."""
        if const_meta_info is None:
            const_meta_info = self.const_meta_info
        if repeated_meta_info is None:
            repeated_meta_info = self.repeated_meta_info
```

```
        metaobj.assign_vals(const_meta_info)
        metaobj.assign_vals(repeated_meta_info)

        # dummy return
        return metaobj.writefile(str(save_path))
```

> 上記例では、parseメソッドは全くのダミーです。

5.4.7 modules/graph_handler.py

```
from __future__ import annotations

from pathlib import Path

import pandas as pd

from modules.interfaces import IGraphPlotter


class GraphPlotter(IGraphPlotter[pd.DataFrame]):
    def plot(self, data: pd.DataFrame, save_path: Path, *, title: str | None = None, xlabel: str | None = None, ylabel: str | None =
None) -> pd.DataFrame:
        return data  # dummy return value for testing purposes
```

> 上記例では、plotメソッドはダミー実装であり、実際には何も実行されません。

5.4.8 modules/structured_handler.py

```
from __future__ import annotations

from pathlib import Path

import pandas as pd

from modules.interfaces import IStructuredDataProcessor


class StructuredDataProcessor(IStructuredDataProcessor):
    def to_csv(self, dataframe: pd.DataFrame, save_path: Path, *, header: list[str] | None = None) -> None:
        if header is not None:
            dataframe.to_csv(save_path, header=header, index=False)
        else:
            dataframe.to_csv(save_path, index=False)
```

5.4.9 modules/datasets_process.py

最後に、上で記述した処理を使うようにdatasetメソッドを記述します。

> 元からある内容を、下記内容で上書きします。

```
from rdetoolkit.errors import catch_exception_with_message
from rdetoolkit.models.rde2types import RdeInputDirPaths, RdeOutputResourcePath
from rdetoolkit.rde2util import Meta

from modules.inputfile_handler import FileReader
from modules.meta_handler import MetaParser
from modules.graph_handler import GraphPlotter
from modules.structured_handler import StructuredDataProcessor
```

```
class CustomProcessingCoordinator:
    def __init__(
        self,
        file_reader: FileReader,
        meta_parser: MetaParser,
        graph_plotter: GraphPlotter,
        structured_processer: StructuredDataProcessor
    ):
        self.file_reader = file_reader
        self.meta_parser = meta_parser
        self.graph_plotter = graph_plotter
        self.structured_processer = structured_processer


def custom_module(srcpaths: RdeInputDirPaths, resource_paths: RdeOutputResourcePath) -> None:
    co = CustomProcessingCoordinator(
             FileReader(),
             MetaParser(),
             GraphPlotter(),
             StructuredDataProcessor()
         )

    # Read Input File
    inputfile = resource_paths.rawfiles[0] # Only one input file is expected
    meta, df_data = co.file_reader.read(inputfile)

    # Save csv
    co.structured_processer.to_csv(
        df_data,
        resource_paths.struct / "sample.csv"
    )

    # Meta
    co.meta_parser.parse(meta)
    co.meta_parser.save_meta(
        resource_paths.meta / "metadata.json",
        Meta(srcpaths.tasksupport / "metadata-def.json")
    )

    # Graph
    co.graph_plotter.plot(
        df_data,
        resource_paths.main_image / "sample.png"
    )


@catch_exception_with_message(error_message="ERROR: failed in data processing")
def dataset(srcpaths: RdeInputDirPaths, resource_paths: RdeOutputResourcePath) -> None:
    custom_module(srcpaths, resource_paths)
```

> 上記例では、ファイル名をプログラム内で固定で記述するなど、実際の構築の際にはそうするべきでない実装が含まれています。

この状態で実行すると、以下の様になります。

```
(venv) $ python main.py
(venv) $
```

> Excelインボイスファイルを入力としていませんので、この時点では"インボイスモード"となっています。

結果、以下の様に入力ファイルをrawフォルダにコピーする処理だけが実施されます。グラフ作成処理がダミーなので、イメージ系のファイル作成はありません。

```
(venv) $ tree data
data
├── attachment
├── inputdata
│   └── sample-data.txt
├── invoice
│   └── invoice.json
├── invoice_patch
├── logs
│   └── rdesys.log
├── main_image
├── meta
│   └── metadata.json
├── nonshared_raw
│   └── sample-data.txt
├── other_image
├── raw
├── structured
│   └── sample.csv
├── tasksupport
│   ├── invoice.schema.json
│   └── metadata-def.json
├── temp
└── thumbnail

15 directories, 8 files
```

5.5 エクセルインボイスモードの開発

> data/inputdata/フォルダに、条件を満たすExcel形式ファイルが存在する場合、自動的に"エクセルインボイスモード"となります。従って、エクセルインボイスモード を使うために、main.pyを修正してConfig()を使ったモード指定するといった追加指定は不要です。

5.5.1 テスト用データ配置

通常のインボイスモードでの開発が済んでいるので、エクセルインボイスでの確認を進めます。

フォルダ構成を初期化します。

```
(venv) $ ./reinit.sh
./data/divided was removed
./data/logs was removed
./data/attachment was removed
./data/invoice_patch was removed
./data/meta was removed
./data/main_image was removed
./data/other_image was removed
./data/raw was removed
./data/nonshared_raw was removed
./data/structured was removed
./data/temp was removed
./data/thumbnail was removed
```

上で使用したデータファイル(`sample-data.txt`)は、不要ですので削除します。

```
rm data/inputdata/sample-data.txt
```

また、invoice.jsonファイルは、以下の構造化処理プログラム実行で上書きされますので、念のためバックアップを取っておきます。

```
cp data/invoice/invoice.json data/invoice/invoice.json.orig
```

エクセルインボイスモードのテスト用のデータを作成します。

```
(venv) $ echo '0000' > data0000.dat
(venv) $ echo '0001' > data0001.dat
(venv) $ echo '0002' > data0002.dat

(venv) $ zip data.zip ./data0000.dat ./data0001.dat ./data0002.dat
  adding: data0000.dat (stored 0%)
  adding: data0001.dat (stored 0%)
  adding: data0002.dat (stored 0%)

(venv) $ ls -l data.zip
-rw-rw-r-- 1 devel devel 493  9月 29 10:13 data.zip
```

> zipコマンドがない場合は、`sudo apt install zip` にてインストールしてから作業してください。

所定の場所に配置します。

```
cp data.zip data/inputdata/
```

以下の様になります。

```
(venv) $ ls -l data/inputdata/
total 4
-rw-rw-r-- 1 devel devel 493  9月 29 10:13 data.zip
```

構造化処理プログラムが、`インボイスモード` か `エクセルインボイスモード` かのいずで実行されるかを判断するポイントは、`data/inputdata` フォルダに、ファイル名の末尾が `_excel_invoice.xlsx` となっているExcelファイルが存在するかどうか、です。当該ファイルが存在すればエクセルインボイスモードとして稼働しますし、存在しなければ通常のインボイスモードとして稼働します。

先に、エクセルインボイスモードで使用するExcelファイルについて説明し、その後作成します。

5.5.2 ファイルモードとフォルダモード

エクセルインボイスを使用する場合、入力ファイルはzipファイル(zip形式の圧縮ファイル)である必要があります。

zipファイルを展開した時に、ファイル1つ1つを別々のデータタイルとして登録するモードを"ファイルモード"、展開したときにデータタイルごとにフォルダが別になっているものを"フォルダモード"といいます。

ファイルモードとフォルダモードでは、登録シートを以下の様に設定します。* ファイルモード：A2セルに"data_file_names"、A3セルに"name"をそれぞれ指定 * フォルダモード：A2セルに"data_folder"を指定(A3セルは任意)

5行目以降のA列に、ファイルモードの場合は展開後のファイル名を、フォルダモードの場合はフォルダ名をそれぞれ指定します。

5行目以降の、B列以降はインボイス(送り状)の内容を記述します。

> エクセルインボイスの登録シートの書き方については、"https://dice.nims.go.jp/services/RDE/service_menu/catalog/manuals/excel_invoice/excel_invoice/" を参照してください。
>
> 上で作成したdata.zipは、展開後ファイルとなりますので、ファイルモード を利用することになります。

5.5.3 エクセルインボイスモードで使用するExcelファイルの条件

エクセルインボイスモードで使用するExcelファイルには、いくつかの条件があります。

- xslx形式のファイルであること。
- ファイル名の末尾が `_excel_invoice.xlsx` (拡張子を含む)であること
- シート名には制限はないが、登録に使用するシート(以下"登録シート"といいます)は、左上端 (A1セル)の値が" `invoiceList_format_id` "であること。
- 左上端 (A1セル)の値が" `invoiceList_format_id` "であるシートが1個だけ存在していること。

以下に、具体的に説明します。

xslx形式のファイルであること

拡張子が `xls` のExcelファイル(古い形式のExcelファイル)はサポートしていません。

`xsl` 形式のExcelファイルをinputfileとして指定した場合、以下のようなエラーとなります。

```
(venv) $ ls -l data/inputdata/
:
-rwxrwxr-x 1 devel devel 94720  9月  3 17:54 sample_excel_invoice.xls

(venv) $ python main.py
 :
Exception Type: ImportError
Error: Missing optional dependency 'xlrd'. Install xlrd >= 2.0.1 for xls Excel support Use pip or conda to install xlrd.
```

エラーメッセージのように、`xlrd` パッケージを導入すればうまく稼働する可能性はありますが、RDEToolKitのサポート外となります。

ファイル名の末尾が **`_excel_invoice.xlsx`** **(**拡張子を含む**)**であること

ファイル名の末尾が `_excel_invoice.xlsx` (拡張子を含む)でないExcelファイルを入力ファイルとした場合、エクセルインボイスモードでの実行はできません。

上記形式に則っていないファイル名のExcelファイルを `data/inputdata` フォルダに配置して、`python main.py` を実行した場合、実行自体はエラーにならない可能性があります(入力ファイルのチェック内容次第)。

しかし、構造化処理プログラムからは、当該Excelファイルが `単なる入力ファイルの1つ` として扱われます。つまり、`エクセルインボイスモード` ではなく、通常の `インボイスモード` での登録となります。

また、ファイル名の末尾が `_excel_invoice.xlsx` (拡張子を含む)のExcelファイルは、`data/inputdata/` フォルダに1個だけである必要があります。当該条件を満たすExcelファイルが2個(以上)ある場合は、以下のようなエラーとなります。

```
(venv) $ ls -l data/inputdata/
:
-rwxrwxr-x 1 devel devel 56883  2月  7 11:26 ample_excel_invoice.xlsx
-rwxr-xr-x 1 devel devel 56883  2月  7 11:31 sample_excel_invoice.xlsx

(venv) $ python main.py

(→ ここでエラーメッセージは表示されません)

(venv) $ cat data/job.failed
ErrorCode=1
ErrorMessage=ERROR: more than 1 excelinvoice file list. file num: 2
```

5.5.4 シートの制限

Excelファイルが複数のシートを含んでいる場合、インボイスとして登録に使用するのは、`シート内のA1セルの値が"invoiceList_format_id"である` シート(以下"登録シート"と呼びます)となります。

登録シートが複数ある場合は、以下のようなエラーとなります。

```
(venv) $ python main.py

(venv) $ cat data/job.failed
ErrorCode=1
ErrorMessage=ERROR: multiple sheet in invoiceList files
```

> デフォルト設定では、ターミナルにはエラーメッセージが表示されません。`data/job.failed` ファイルや、`data/logs/rdesys.log` (存在している場合)などを確認し、必要な対処を実施してください。

登録シートの"シート名"には特に制限はありません。Excelが許容する範囲で自由にシート名を決定することが出来ます。新規ファイル作成時の `Sheet 1` や `Sheet1` でも問題ありません。

5.5.5 Excelファイルの作成(編集)

上記を踏まえ、エクセルインボイスモードで使用するExcelファイルを作成していきます。

RDEToolKitには、エクセルインボイスモードで使用するExcelファイルのひな形を作成する機能がありますので、これを使います。

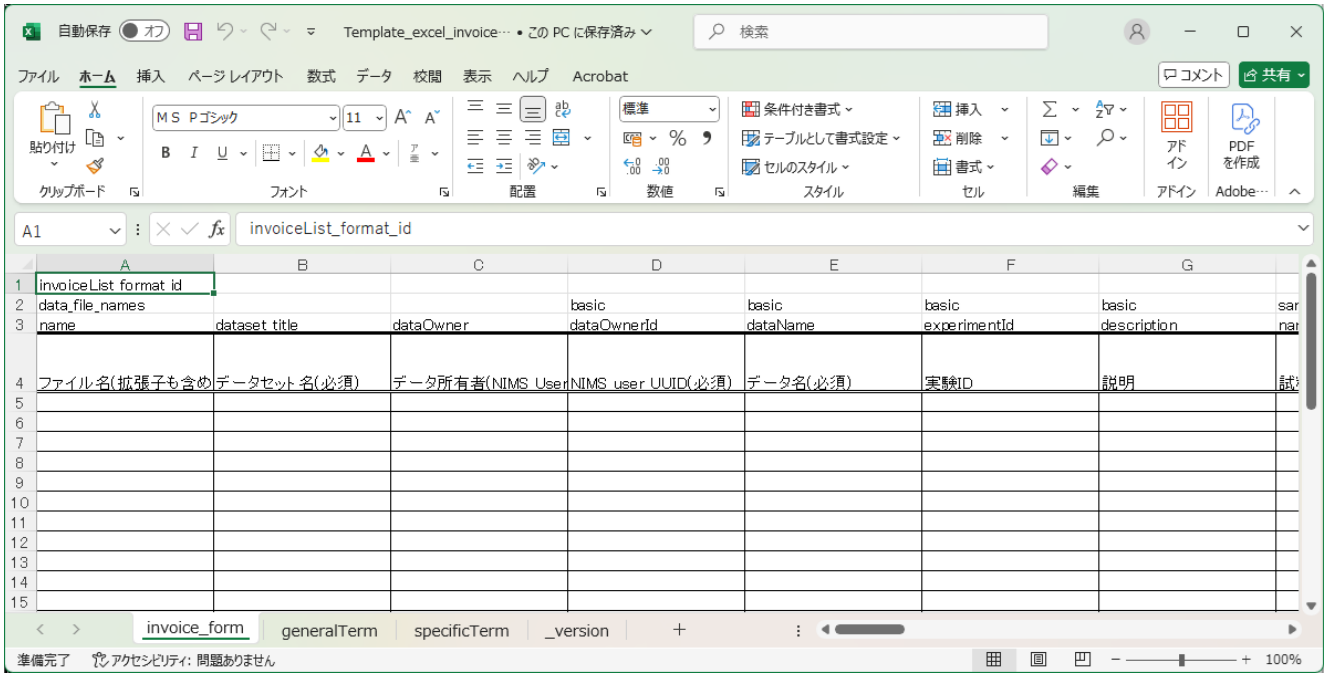
ひな形を作成するには、`make-excelinvoice` サブコマンドを使用します。

```
(venv) $ python -m rdetoolkit make-excelinvoice data/tasksupport/invoice.schema.json
  Generating ExcelInvoice template...
```

```
:
  ExcelInvoice template generated successfully! : /home/devel/rde-sample/excelinvoice/container/template_excel_invoice.xlsx
```

> カレントディレクトリに、`template_excel_invoice.xlsx` という名前で作成されました。
>
> フォルダモードを使用する場合は、(末尾に) `-m folder` を追加して実行してください。

このファイルを、MS Excelが稼働するPC(Windows、Mac)にコピー(もしくは移動)します。



> `generalTerm` および `specificTerm` のシートは、空のシートとしてでも必要なシートです。これら2つのシートは必要な情報がセットされているはずですので、修正不要です。
>
> `_version` のシートには、生成したRDEToolKitのバージョン番号がセットされています。当該シートは存在していても問題ありませんので、通常はそのまま保持してください。

A1セル

`invoice_form` のシートを選択します。

> 前述の様にシート名は任意です。ここでは変更せずにそのまま利用することにします。

A1セル(シートの左上端)の値が `invoiceList_format_id` であることを確認します。

> そうなっているはずなので、確認だけでOKです。

A2セル

今回は、"ファイルモード"ですので、`data_file_names` であることを確認します。

"フォルダモード"を使う場合は、`data_folder` に変更してください。

2行目**(A2**セルを除く**)**および**3**行目

インボイスの項目名が入りますので、基本的に変更しないでください。

4行目

2、3行目で示した項目名の、日本語名が入ります。ここも修正不要です。

> この行は、入力する担当者向けにある行です。構造化処理プログラム実行には影響を与えません。別の項目名に変更するなどしても影響はないですが、混乱のもととなりますので、修正しないでください。
>
> またセルの中に改行を含むものがありますが、うまく改行されずに表示されることがあります。カーソルを持っていき、いったん編集モードにしてから確定すると正しく改行されます。気になる場合はお試しください。

5.5.6 5行目以降

送り状データを、1行ずつ書いていきます。

- 4行目の日本語項目名に (必須) と書かれた項目は、一部の例外を除き必ず入力してください。入力が漏れていた場合は、構造化処理プログラム実行時にエラーになる場合があります。

> 一部の必須項目は、管理上必須としているものもあります。例えば データセット名 が空欄だと、あとで当該Excelファイルを見た場合に、どのデータセット向けのExcelインボイスファイルかが判別できなくなる可能性があります。そのため、データセット名は必須と表記されています。
>
> また状況に応じて必須となるものに"(必須)"と表記されている場合もあります。例えば"試料UUID(必須)"とある場合は、既存試料と紐付ける場合は入力必須 ということであり、試料新規作成の場合は、当該セルは空欄でもエラーとなりません。こういった項目の場合は、必須と明記されていても空欄で問題ありません。

テストデータとして、それぞれのセルに以下を入力します。

> サンプル値ですので、変更して入力しても構いません。サンプルですので、多くの列で、まったく意味がないデータとなっています。
>
> 下記例では、"入力値"欄に3つ例示が有る場合は行毎の、1個しかない場合は、3行とも同じ値を使います。

列名	列番号	入力値	備考
ファイル名 (拡張子も含め入力)	A列	data0000.dat、data0001.dat、data0002.dat	展開後のファイル名を拡張子込みでセット
データセット名	B列	Dummy	必須だが、ダミーでよい
データ所有者	C列	(空欄)	
NIMS user UUID	D列	97e05f8b9ed6b4b5dd6fd50411a9c163a2d4e38d62646236666383669	全部同じでよい
データ名	E列	データ0、データ1、データ2	
実験ID	F列	(空欄)	
説明	G列	(空欄)	
試料名(ローカルID)	H列	Inu、Neko、(空欄)	7行目だけ空欄にします。
試料UUID	I列	(空欄)	
試料管理者UUID	J列	de17c7b3f0ff5126831c2d519f481055ba466ddb6238666132316439	全部同じでよい
化学式・組成式・分子式など	K列	(空欄)	
参考URL	L列	(空欄)	
試料の説明	M列	(空欄)	
一般名称	N列	A1、B1、C1	
CAS番号	O列	A2、B2、C2	
結晶構造	P列	A3、B3、C3	
試料形状	Q列	A4、B4、C4	
試料購入日	R列	(空欄)	
購入元	S列	A6、B6、C6	
ロット番号、製造番号など	T列	A7、B7、C7	
キー1	U列	あ1、あ2、あ3	
キー2	V列	い1、い2、い3	
キー3	W列	う1、う2、う3	
キー4	X列	え1、え2、え3	
キー5	Y列	お1、お2、お3	

> 前述の様に、データセット名は管理上必要な項目なので開発時点では"Dummy"等意味の無い値でも問題ありません。実際の登録に用いるExcelファイルにおいては、どのデータセット向けのExcelインボイスファイルかがわかるように データセット名 を正しくセットしてください。

5.5.7 その他の注意点

2列目以降に 空の列 を追加しても、項目名(2行目と3行目)が空の場合は影響ありません。

5行目以降は、途中の行に 空の行 があってはいけません。

途中に空の行のあるExcelファイルを用いて構造化処理プログラムを実行した場合は、以下のようなエラーとなります。

```
(venv) $ cat data/job.failed
ErrorCode=1
ErrorMessage=Error! Blank lines exist between lines
```

5.6 エクセルインボイスモードとして実行

上記内容を入力→保存したExcelファイルを、`data/inputdata` フォルダに配置します。

```
(venv) $ tree data/inputdata/
data/inputdata/
├── template_excel_invoice.xlsx
└── data.zip

1 directory, 2 files
```

> 上記例では、ひな形作成時のファイル名のまま利用していますが、運用時は適切にファイル名を変更してください。

続いて、(通常の)インボイスモードでの実行結果をクリアします。

```
./reinit.sh
```

この状態でエクセルインボイスモードとして実行します。

```
python main.py
```

出来上がったフォルダ構成は以下の様になります。

```
(venv) $ tree data
data
├── attachment
├── divided
│   ├── 0001
│   │   ├── attachment
│   │   ├── invoice
│   │   │   └── invoice.json
│   │   ├── invoice_patch
│   │   ├── logs
│   │   ├── main_image
│   │   ├── meta
│   │   │   └── metadata.json
│   │   ├── nonshared_raw
```

```
│   │   │   └── data0001.dat
│   │   ├── other_image
│   │   ├── raw
│   │   ├── structured
│   │   │   └── sample.csv
│   │   ├── temp
│   │   └── thumbnail
│   └── 0002
│       ├── attachment
│       ├── invoice
│       │   └── invoice.json
│       ├── invoice_patch
│       ├── logs
│       ├── main_image
│       ├── meta
│       │   └── metadata.json
│       ├── nonshared_raw
│       │   └── data0002.dat
│       ├── other_image
│       ├── raw
│       ├── structured
│       │   └── sample.csv
│       ├── temp
│       └── thumbnail
├── inputdata
│   ├── template_excel_invoice.xlsx
│   └── data.zip
├── invoice
│   ├── invoice.json
│   └── invoice.json.orig
├── invoice_patch
├── logs
│   └── rdesys.log
├── main_image
├── meta
│   └── metadata.json
├── nonshared_raw
│   └── data0000.dat
├── other_image
├── raw
├── structured
│   └── sample.csv
├── tasksupport
│   ├── invoice.schema.json
│   └── metadata-def.json
├── temp
│   ├── data0000.dat
│   ├── data0001.dat
│   ├── data0002.dat
│   └── invoice_org.json
└── thumbnail

42 directories, 22 files
```

前述のように、構造化処理プログラムは修正不要です。`data/inputdata` フォルダ上に、単独の入力ファイルを置くのではなく、zip圧縮された入力ファイルとExcelインボイスファイル(xlsxファイル)を配置するだけで、通常のインボイスモードがエクセルインボイスモードに変わります。

複数のデータタイルに対応するため、dividedフォルダが新規に作成され、さらに4桁数字連番のサブフォルダが作成されます。もとの `data` フォルダに1行目のデータ処理結果が、`data/divided/0001` に2行目のデータが、`data/divided/0002` に3行目のデータがそれぞれ格納されます。

例えば、生成されたinvoice.jsonのキー1～キー5までの部分を以下に示します。

data/invoice/invoice.json

```
:
    "custom": {
        "key1": "あ1",
        "key2": "い1",
        "key3": "う1",
        "key4": "え1",
        "key5": "お1",
:
```

data/divided/0001/invoice/invoice.json

```
:
    "custom": {
        "key1": "あ2",
        "key2": "い2",
        "key3": "う2",
        "key4": "え2",
        "key5": "お2",
:
```

data/divided/0002/invoice/invoice.json

```
:
    "custom": {
        "key1": "あ3",
        "key2": "い3",
        "key3": "う3",
        "key4": "え3",
        "key5": "お3",
:
```

> 上の様に、key1～key5が順番に並ぶとは限りません。

また上の例では、3つ目(シート上は7行目)の `試料名(ローカルID)` を空欄にしました。構造化処理プログラム実行の結果を確認すると以下の様になっています。

data/divided/0002/invoice/invoice.json

```
:
    "sample": {
        "sampleId": null,
        "names": [
            "testdev01_1"
        ],
:
```

この"testdev01_1"は、もとのinvoice.jsonの `sample.names` の値が転記されています。

```
(venv) $ vi data/invoice/invoice.json.orig
:
:
    "sample": {
        "sampleId": "8ea50153-7a87-44b7-9d6e-10179c507039",
        "names": [
            "testdev01_1"  (★)
        ],
        "composition": "AB2C3",
```

```
        "referenceUrl": "test_ref",
:
```

> Excelファイルに指定が無かったので、上記の(★)の行のデータが利用された、ということです。

data.zipは `data/temp/` フォルダ上で展開され、それぞれの `nonshared_raw/` フォルダに展開されます。

```
(venv) $ tree -f data | grep \\.dat
|   |   |   └── data/divided/0001/nonshared_raw/data0001.dat
|       |   └── data/divided/0002/nonshared_raw/data0002.dat
|   └── data/nonshared_raw/data0000.dat
:
|   ├── data/temp/data0000.dat
|   ├── data/temp/data0001.dat
|   ├── data/temp/data0002.dat
```

以上

6. 変更履歴

1.3.0 (2025.09.30)

- 対応RDEToolKit : v1.3.4
- マルチデータタイルモード指定時のキーワードが、大文字小文字を区別するようになった
- 4章のPIL(Image)を利用した処理がうまく動いていなかった部分を修正
- 実行結果がわかりやすいように、Excelインボイスの設定内容を変更

1.2.0 (2025.05.22)

- 対応RDEToolKit : v1.2.0

1.1.1 (2025.02.14)

- 対応RDEToolKit : v1.1.1