User Manual of Automatic Data Curation Tool(ADCT): A bulk data curator software in Library and Information Science

Version 3.0

Arunavo Banerjee
Senior Project Officer (NDLI),
Master of Science, Department of Computer Science & Engineering,
Indian Institute of Technology Kharagpur, India
banerjee.arunavo@iitkgp.ac.in

April 2024

Contents

1 Introduction

The `Automatic Data Curation Tool (ADCT)` is to guide and facilitate a Digital Library Data Curator to do batch data curation and transformation at ease without the exclusive need of a computer programmer. The system has two fundamental components,

1) A **JSON/CSV/XLSX** file written in the `MetaCur` language for defining the curation logic to apply curation at a Data Collection or Item/Handle_ID level, called as a **Logic File**, and

2) A Run Parameter Configuration file called `Run Properties (*.run.properties)` to select the input/output data and logic file and set up the necessary configuration parameters whenever required.

These two files, along with the data and the required supplementary configuration files, are put inside a folder named `Curation Bundle` to run the curation logic onto the required data and generate the output. In this user manual, we explain the details of the application and its usage, data curation in the digital library in a nutshell, and show examples of how to apply MetaCur syntax for various curation requirements applicable to the Digital Library domain. The following description provides the language details and constructs to show how to perform the required tasks.

1.1 What is Data Curation

Data curation is maintaining and adding value to a trusted body of digital research data for current and future use; it encompasses the active management of data throughout the research life cycle.

1.2 Why Curate

Data are evidence supporting research and scholarship; better research is based on verifiable data, which may, in turn, lead to new knowledge. Observational, environmental and other data are unique and cannot be recaptured or reproduced. Data may represent records and have associated legal requirements; curation will allow us to protect the data for the future and manage risks.

Where data is created in the course of research that is publicly funded, a duty to manage is implied, including the provision of access to data and data reuse. Meeting this obligation will be enabled by good data stewardship.

1.3 What Data

Curation requires effort and resources. In principle, any digital object or database may be perceived as likely to be of sufficient value for the effort and expense of curation. Data may be curated in the short-term but may not require long-term preservation. There are cost barriers to both curation and preservation, so an effective appraisal and selection process is essential. It is important to build an appropriate, robust, distributed infrastructure to support curation. Components may include laboratory repositories, institutional repositories, subject or discipline repositories, databases and data centres. Coordinated strategies and policies at the research funder's level are required, together with sufficient investment for the future.

1.4 When is the Need to Perform Curation

Curation applies throughout the research life-cycle, from before or at the point of data creation, through primary use to eventual disposal. The ADCT Curation Life-cycle Model describes the processes involved in curation. The curation life-cycle may continue indefinitely, and curation cannot be left to the end of primary use, for example, at the end of a funded project.

1.5 Who should have responsibility

Several roles and responsibilities are involved in curation and preservation practice within the Curation Life-cycle. Curation should start with the individual or group that creates or captures the information. Curation requires a significant amount of domain knowledge; data scientists and data curators may add value to the original data. Users, custodians, and re-users of the data and the funding bodies have curation responsibilities. There is currently a shortage of experienced data scientists and curators with digital preservation experience.

1.6 How will curation be achieved

The key is to follow good practice, including domain, national, and international standards in the capture, management, and archiving of data. Processes and tools to ensure easy discovery, control access, and facilitate data sharing and reuse are required. An infrastructure of data centres and trusted repositories, together with methodologies to demonstrate provenance and assure authenticity, are essential. Curation practice in detail will depend on the domain or discipline. Data structure, scale, and ownership must also be taken into account, as well as the diversity of cultures and research methodologies. Curation can build on and fit in with current practices, for example, researchers' informal sharing of ongoing research with colleagues, their training, or the need to comply with formal regulatory and ethical procedures.

One particular method for standardization of curation can be done with the reusability of existing logic across data sources for a field or similar fields that follow the same semantics. This means in the Digital Library domain; data behaviours can be identified using a schema by defining its characteristics and relationships amongst themselves. The complete process of Data Curation in a Digital Library domain is described below as how a Library Science Domain Expert can utilise the ADCT software and its native language; MetaCur is also mentioned in the Appendix of this User Manual.

2 The ADCT Language (MetaCur)

2.1 Overview

The ADCT language, named as `MetaCur`, follows a JSON[1] syntax to perform various instructions related to batch data curation in a Digital Library and Information Systems. The philosophy behind the language is to provide the data curation instructions on each field as key-value pair information. The syntax of the language follows a context-sensitive grammar; that is, both the *Left Hand Side (LHS)* and the *Right Hand Side (RHS)* of a production rule is a string that can have a combination of multiple Terminal (Σ) and Non-Terminal ($\mathcal{NT}$) symbols that defines the context of the grammar. The grammar is expressed in the form as

$$\alpha\mathcal{A}\beta \Rrightarrow \alpha\gamma\beta \text{ where:}$$

$$\mathcal{A} \in \mathcal{NT} \text{ and } |\mathcal{A}| = 1\,,$$

$$\alpha, \beta \in (\mathcal{NT} \cup \Sigma)^{*} \text{ and}$$

$$\gamma \in (\mathcal{NT} \cup \Sigma)^{+}$$

In the above expression, $\mathcal{NT}$ denotes the set of non-terminals, $|\mathcal{A}|$ denotes the size of the non-terminal variable $\mathcal{A}$ and Σ denotes the set of terminal symbols. The equivalence of CSL in `MetaCur` is Field names, Actions are denoted as the non-terminals and the final target values or expressions specified for fields are denoted as the terminals. The start symbol $\mathcal{S}$ is denoted by the key `Fields` in the JSON script, which generates the subsequent logics as productions.

The way this is implemented in `MetaCur` is that field names are mentioned as the JSON keys and the curation logics which are to be applied for those fields are enclosed as a JSON Object value. Each JSON Object Value for a particular field Key is termed as `Field Translation Block (FTB)`. The set of instructions that are to be applied for a field to perform some data transformation or curation is to be enclosed inside a JSON Array construct named `action` inside the Field Translation Block. The Field-Field Translation Block (FTB) Pairs are then enclosed as a JSON Object within another Key `Fields`, inside a root JSON construct. The CSL grammar expression for the `MetaCur` language after applying the above substitutions are as follows:

$$\mathtt{Fields} \Rrightarrow \mathtt{A\ B}$$

$$\mathtt{A} \Rrightarrow \mathtt{fieldName\ :\ FTB|}\epsilon$$

$$\mathtt{B} \Rrightarrow \mathtt{,\ fieldNname\ :\ FTB\ B|}\epsilon$$

$$\mathtt{FTB} \Rrightarrow \mathtt{FP\ C|ACTION\ D}$$

$$\mathtt{C} \Rrightarrow \mathtt{,\ FP\ C|}\epsilon$$

$$\mathtt{FP} \Rrightarrow \mathtt{dataType|multivalued|controlled|}$$

$$\mathtt{validation|sourcePriority}$$

$$\mathtt{D} \Rrightarrow \mathtt{,ACTION\ D|}\epsilon$$

$$\mathtt{ACTION} \Rrightarrow \mathtt{ACTION\ E}$$

$$\mathtt{ACTION} \Rrightarrow \mathtt{useMap|moveField|}$$

[1] `https://www.json.org/json-en.html`

$$lookUp|add|deleteField|$$

$$attach|curate$$

$$E \Rightarrow AD\ FILTER$$

$$FIELD \Rightarrow FILTER\ E|\epsilon$$

$$E \Rightarrow ,\ FILTER\ E|\epsilon$$

In the above grammar `Fields` is the start symbol, **`A, B, Field-FTB Pair, FTB, FP, C, ACTION, D, E, AD, FILTER`** are the set of non-terminals and **`fieldName, dataType, multiValued, controlled, validation, sourcePriority, useMap, moveField, lookUp, add, deleteField, attach, curate`** are the set of terminals.

Following the above grammar, below is a detailed indicative example of the logic script. The script starts with the symbol `Fields`, which then expands to zero, one or many Field-Field Translation Block pairs. Each pair is marked with an indicative name ***`fieldName1, fieldName2`*** and the corresponding FTB contains the respective Field Properties and the actions that are to be applied to the fields for its fieldValue curation. Each action would then contain its action descriptor and its subsequent features.

```
{
  "Fields": {
    "<fieldName1>": {
      "<FP 1>": "<FP1 Values>",
      "<FP 2>": "<FP2 Values>",
      "action": [
        "<Ins 1>",
        "<Ins 2>"
      ],
      "Ins 1": {
        "<Ins 1 Prop 1>": "Ins 1 Prop1 Value(s)",
        "<Ins 1 Prop 2>": "Ins 1 Prop 2 Value(s)",
        "filter": [
          "<filter1>",
          "<filter2>"
        ],
        "filter1": {
          "fields": [
            "<fields1>",
            "<fields2>"
          ],
          "<fields1>": {
            "fieldName": "<name_of_the_field>",
            "criteria": "filter_criteria",
            "fieldValue": "field_value_to_filter"
          },
          "<fields2>": {
            "fieldName": "<name_of_the_field>",
            "criteria": "filter_criteria",
            "fieldValue": "field_value_to_filter"
          },
          "expr": "<boolean_expression>",
          "<Ins 1 Prop 1>": "Ins 1 Prop1 Value(s)",
          "<Ins 1 Prop 2>": "Ins 1 Prop 2 Value(s)",
          "Fields": {}
        }
      },
      "Ins 2": {},
      "<fieldName2>": {}
    }
  }
}
```

Figure 1: Indicative Example of a Logic Script as followed from the above grammar

The instructions and the details in the above example Field Translation Block are as follows: A `Field Translation Block(FTB)` constitutes three components,

1. **`Field Properties:`**
   A set of Field Properties to describe the behaviour of the field during a curated value assignment. The Field Properties parameters are set at a Schema Level. Each field under the same schema is characterized by the same set of Field Properties. If nothing is mentioned, then a default behaviour is assumed for the field. In order to override the default behaviour, specific property and their values are to be mentioned as key-value pair information within the FTB. We shall speak more about the Field Properties keys and their constructs in detail in its specific section.

2. **`Action:`**
   An Action block comprising a set of actions to be applied on each of the field values as per their specifications. The key action inside the `FTB` is a JSON Array, which holds the curation instructions. The instructions are applied in the order they are mentioned in the JsonArray action. Actions have their own Action descriptor, which has properties specific to a particular action. Actions can also be nested within another action. We shall describe the Action and its details in its respective section.

3. **`Action Descriptor:`**
   Each Action is defined by its own Action Descriptor, which varies across different actions. The Filter action is generally applicable across all action descriptors. A common example is the parameter `inputFile`, which is applicable across actions `useMap`, `lookUp`, and `moveField` since these actions require a configuration file to be associated with them. For other actions such as `add` and `copyData`, the parameter `targetValue` becomes relevant due to their nature of application in the data. Action Descriptors are the containers for nested actions. The output of one action can be sent to another action by nesting the second action within the first. We shall describe the details of Action Descriptors in the specific section in Action.

In such an FTB, among all three of them, either an Acton block or a Field Properties Block needs to be present. The Action descriptor is optional, i.e., if no default behaviour of an action is overridden, the Action Descriptor block is not necessary. Let us see a Field Translation Block(FTB) below, and we will dissect it in further Sections and explain the components step by step.

```json
{
  "_comment": "FTB Root Construct",
  "Fields": {
    "_comment": "All Field-Field Translation logic pairs are to be enclosed within
↪   the Fields Key as a JSON Object. An FTB execution does not necessarily maintain
↪   the order in which it has been provided inside the Fields key value.",
    "dc.subject": {
      "_comment": "A Field Translation Block(FTB) of dc.subject",
      "action": [
        {
          "_comment": "Action JSONArray holds the list of actions to be performed
↪   on the source field dc.subject. In the action list, the action lookUp is to be
↪   performed on each value of the dc.subject. Subsequently, action copyData is to
↪   be performed. The application of the next action is determined by the
↪   fall-through behaviour, which is discussed later."
        },
        "lookUp",
        "copyData"
      ],
      "lookUp": {
       "_comment": "The Action Description(AD) Block for lookUp. Each AD block for
↪   a particular Action can have varied parameters. For lookUp the parameter
↪   inputFile identifies the location to fetch the lookUp configuration details.",
        "inputFile": "lookUp.xlsx"
      }
    },
    "dc.subject.ddc": {
      "_comment": "Multiple FTBs can be specified separated by comma as per JSON
↪   syntax. Below is an example of FTB for dc.subject.ddc. A Field Translation
↪   Property named validation for dc.subject.ddc",
      "validation": true,
      "action": [
        {
          "_comment": "Action JSONArray holds the list of actions to be performed
↪   on dc.subject.ddc. In the below example, the action add is to be performed on
↪   each value of dc.subject.ddc. Then, if subsequent action copyData is to be
↪   performed, would depend on the fall-through behaviour, which is discussed
↪   later."
        },
        "add",
        "copyData"
      ],
      "add": {
        "_comment": "Action Descriptor(AD) for the action add. The set of
↪   parameters for add AD are targetValue and mode.",
        "targetValue": "091"
      }
    }
  }
}
```

Figure 2: Field Translation Block Details

Finally, a Curation task on a data source can be performed in three ways: 1) **Collection Level Curation**, as mentioned above using MetaCur, Data Curation logic can be applied across the whole collection of data in this method of curation. 2) **Item ID Level Curation**, In this way, the data curation logic is applied for specific item IDs instead of the complete collection. Since Handle_ID is frequently defined as the Item ID for Digital Library Data Archival systems, in this scenario the default specification of logic is considered at a Handle_ID level. 3) Lastly, the third level of the Data Curation task is called **Data Stitching Curation**. Data stitching is a very non trivial activity performed to establish the relationship across the elements of a collection of data. The details of such curation types, methods and the associated files are described in the following Section 3.

2.2 Keywords

Data Curation, Extraction Transformation and Load, Data Extraction, MetaCur, Digital Library.

2.3 Benefits and Value

A few salient features of the `Automatic Data Curation Tool` are :

- Providing data accuracy through this software.
  A library field professional can analyse and organize bulk data through this software without the help of technicians.
- Quick turn around time for data collection to data dissemination
  It takes very little time to organize bulk data with this software as the need for some technician or computer programmer is removed from the operational process. Also, while organizing the bulk data through the software, any errors in the data can be detected, which can then be quickly corrected without any back-and-forth movement of the data.
- Standardization and Internationalization of data repositories
  This software has been developed keeping in mind the future, which is all international standard supported tools in the library field and so on. Through this, we can present any repository data in a standard format to users through organizations.

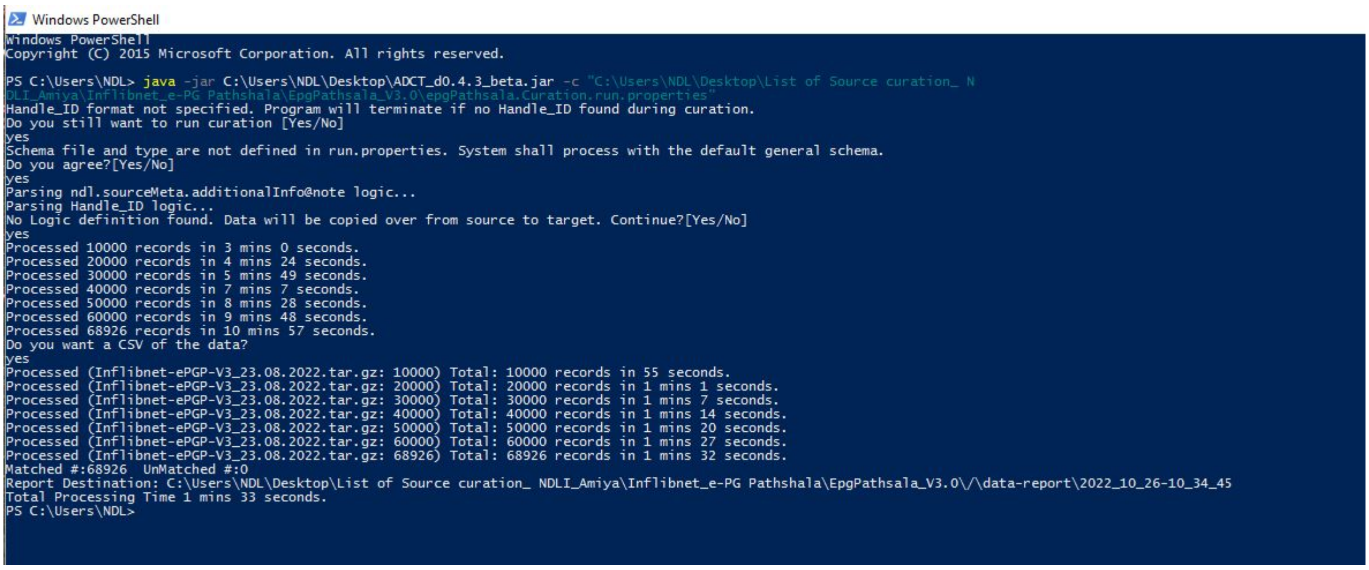
2.4 Platform Requirements

The software can be run on machines with either Windows or Linux operating systems. The machine is required to have a minimum of 8GB of RAM and 8 cores with 2GHz of computing capacity. Though the minimum requirement seems to be within range, however, for large volumes of data, having adequate computing power is essential.

2.5 Execution Process

The `Automatic Data Curation Tool` software is currently distributed in the form of an executable Java ARchive `JAR (.jar)` file, which needs to be executed using a `Command Line Input (CLI)` syntax. For Windows operating systems the CLI syntax can either run in the Command Prompt or in Powershell system utility. For Unix/Linux machines, using the terminal utility would be adequate to run the `ADCT` software. The syntax to run the software is nothing different from running self-executing JAR files. It is denoted as `Java -jar ADCT_v{current version no}.jar -<Flag>[c|h|s] {*}.run.properties <absolute path to run parameter file>` where the first keyword is the application program `JAVA`, which is called to execute the

`ADCT .jar` application. The next keyword is the flag `-jar` identifying the nature of the application; the third parameter is the `ADCT` software application with the appropriate version number. The following two parameters are the direct input to the `ADCT` system as mentioned in Section 1. The first parameter is the run flag, which determines the type of curation operation to perform in the data, as mentioned in Section 3.1. The second parameter is the run properties file containing structural and environmental information of data and the other important factors of the process. This is also referred to in Section 3.2. An example of an ADCT running instance is shown in the figure below.



```
Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Users\NDL> java -jar C:\Users\NDL\Desktop\ADCT_d0.4.3_beta.jar -c "C:\Users\NDL\Desktop\List of Source curation_ N
DLI_Amiya\Inflibnet_e-PG Pathshala\EpgPathsala_V3.0\epgPathsala.Curation.run.properties"
Handle_ID format not specified. Program will terminate if no Handle_ID found during curation.
Do you still want to run curation [Yes/No]
yes
Schema file and type are not defined in run.properties. System shall process with the default general schema.
Do you agree?[Yes/No]
yes
Parsing ndl.sourceMeta.additionalInfo@note logic...
Parsing Handle_ID logic...
No Logic definition found. Data will be copied over from source to target. Continue?[Yes/No]
yes
Processed 10000 records in 3 mins 0 seconds.
Processed 20000 records in 4 mins 24 seconds.
Processed 30000 records in 5 mins 49 seconds.
Processed 40000 records in 7 mins 7 seconds.
Processed 50000 records in 8 mins 28 seconds.
Processed 60000 records in 9 mins 48 seconds.
Processed 68926 records in 10 mins 57 seconds.
Do you want a CSV of the data?
yes
Processed (Inflibnet-ePGP-V3_23.08.2022.tar.gz: 10000) Total: 10000 records in 55 seconds.
Processed (Inflibnet-ePGP-V3_23.08.2022.tar.gz: 20000) Total: 20000 records in 1 mins 1 seconds.
Processed (Inflibnet-ePGP-V3_23.08.2022.tar.gz: 30000) Total: 30000 records in 1 mins 7 seconds.
Processed (Inflibnet-ePGP-V3_23.08.2022.tar.gz: 40000) Total: 40000 records in 1 mins 14 seconds.
Processed (Inflibnet-ePGP-V3_23.08.2022.tar.gz: 50000) Total: 50000 records in 1 mins 20 seconds.
Processed (Inflibnet-ePGP-V3_23.08.2022.tar.gz: 60000) Total: 60000 records in 1 mins 27 seconds.
Processed (Inflibnet-ePGP-V3_23.08.2022.tar.gz: 68926) Total: 68926 records in 1 mins 32 seconds.
Matched #:68926  UnMatched #:0
Report Destination: C:\Users\NDL\Desktop\List of Source curation_ NDLI_Amiya\Inflibnet_e-PG Pathshala\EpgPathsala_V3.0\/\data-report\2022_10_26-10_34_45
Total Processing Time 1 mins 33 seconds.
PS C:\Users\NDL>
```

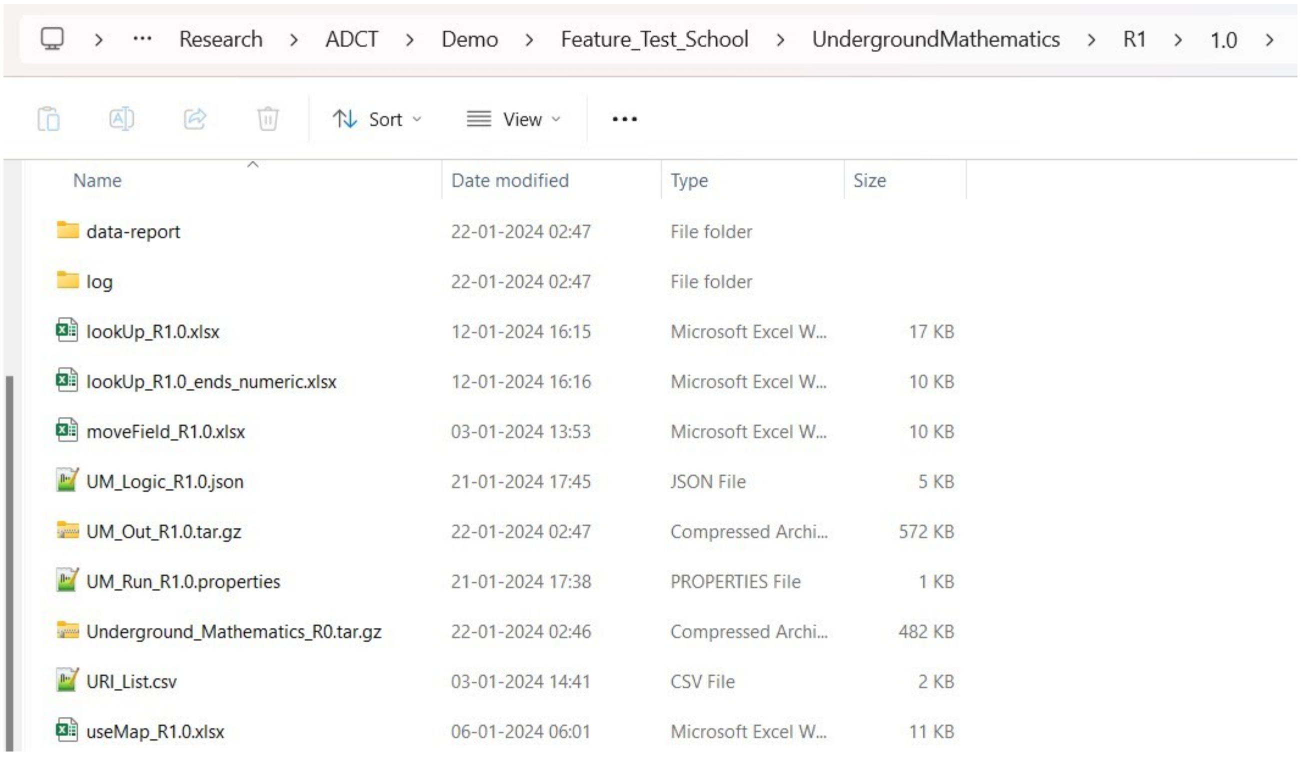
Figure 3: An ADCT Running Instance

3 Curation Bundle

3.1 Overview

The `Automatic Data Curation Tool(ADCT)` is a locally executable software requiring access to the physical files stored in the user's own computer. Since a curation can stretch to multiple phases and rounds it becomes important that the configuration files used in the curation process are well organized. Otherwise, the logic would frequently get lost and the user would have immense pain to correlate these files with the curated data version. Hence, the concept of `Curation Bundle` is introduced, which is a collection of such configuration files, data files, logs and data reports that identifies a round of curation. As will be explained later in Section 3.4, Major and Minor versions are version identifiers that map a `Curation Bundle` with verbose requirements naturally identified by a Subject Matter Expert herein a Library Science professional in the context. It has already been introduced in the previous sections that Data Curation on source data can be applied in three ways. Let us now introduce these below and see what are the specific requirements for each type of curation method and what differences are there between them.

1. **`Collection Level Curation`**:
   The `Collection Level` Curation is the fundamental way of curation where a curation logic is applied to the whole collection of the data. The complete data is processed as a batch and the logic uniformly gets applied to each and every item irrespective of their characteristics and any object class they might belong to in the underlying dataset. According to the internal logic definition defined in the logic script, if a filter logic for a particular field exists, then respective filters are applied to the selected groups of items for that particular field on which the curation actions apply. The rest of the metadata for those items is processed as it is. In case of no filter, the logic applies to the complete collection of the data. The execution flag for a collection level curation mode is **`-c`**, which stands for collection, and the logic script file extension is **`JSON(.json)`**.

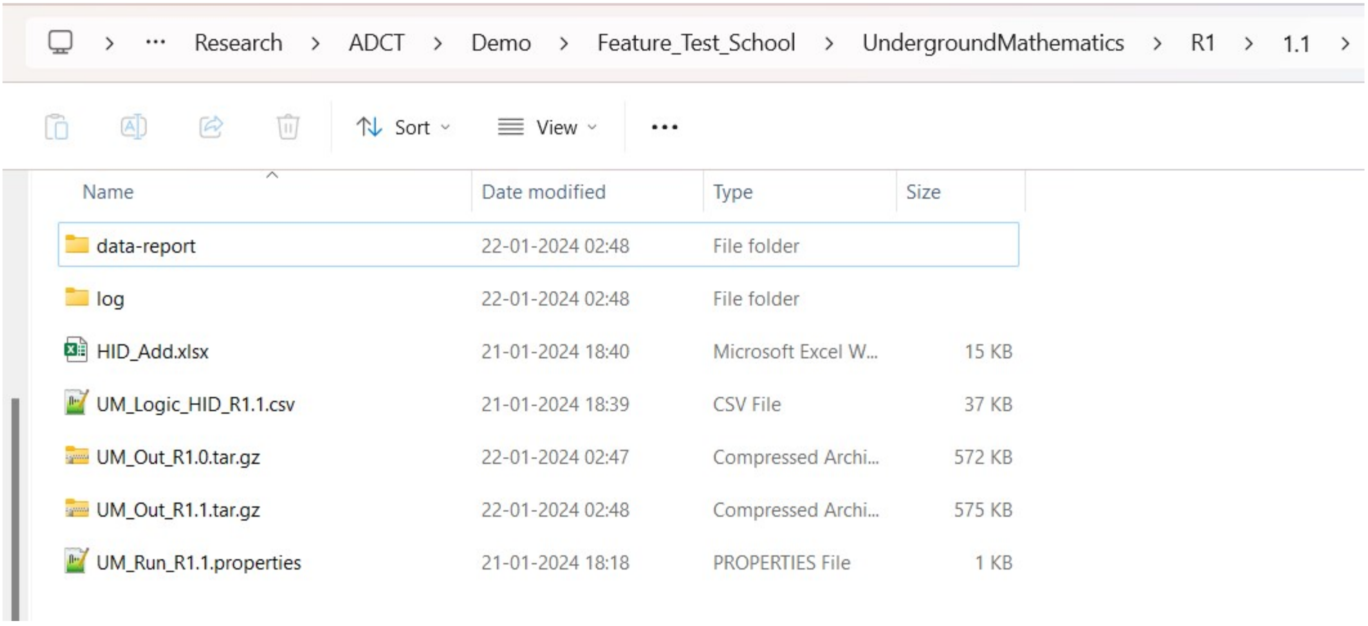


··· > Research > ADCT > Demo > Feature_Test_School > UndergroundMathematics > R1 > 1.0 >
Sort
View

Name	Date modified	Type	Size
data-report	22-01-2024 02:47	File folder	
log	22-01-2024 02:47	File folder	
lookUp_R1.0.xlsx	12-01-2024 16:15	Microsoft Excel W...	17 KB
lookUp_R1.0_ends_numeric.xlsx	12-01-2024 16:16	Microsoft Excel W...	10 KB
moveField_R1.0.xlsx	03-01-2024 13:53	Microsoft Excel W...	10 KB
UM_Logic_R1.0.json	21-01-2024 17:45	JSON File	5 KB
UM_Out_R1.0.tar.gz	22-01-2024 02:47	Compressed Archi...	572 KB
UM_Run_R1.0.properties	21-01-2024 17:38	PROPERTIES File	1 KB
Underground_Mathematics_R0.tar.gz	22-01-2024 02:46	Compressed Archi...	482 KB
URI_List.csv	03-01-2024 14:41	CSV File	2 KB
useMap_R1.0.xlsx	06-01-2024 06:01	Microsoft Excel W...	11 KB

Figure 4: An Example of a Collection Level Curation Bundle

2. **`Item ID(Handle_ID Level Curation`**:
   The `Handle_ID Level Curation` method is the crudest way to curate a dataset where for each respective Handle_ID, individual logic is defined and applied through the data. This process can be presumed to be equivalent to manual creation only it is being executed through ADCT. For an ID level curation, curation logic is applied at an individual identity level of items, and then individual items are processed one after another. Herein we have chosen to show the ID as Handle_ID which is a common digital library document identifier, however, any identity column can be used when pre-defined within the system. For an ID (herein Handle_ID) level curation, the complexity is kept simple, and the provisions for filters and nestings are not provided in the logic script application. Since the implementation of the logic is at the granular level where the exact value is set for a particular field and grouping of items is not applicable hence, simple versions of actions such as basic `useMap, lookUp, copyData,` add and `deleteField` have been provided with this mode of execution. The execution flag for an Item ID level mode curation is **`-h`** identified after `Handle_ID` for common digital library document identifier, and the logic script file extension is **`CSV(.csv)`** provided in the logic parameter of the curation run properties `.run.properties` file.



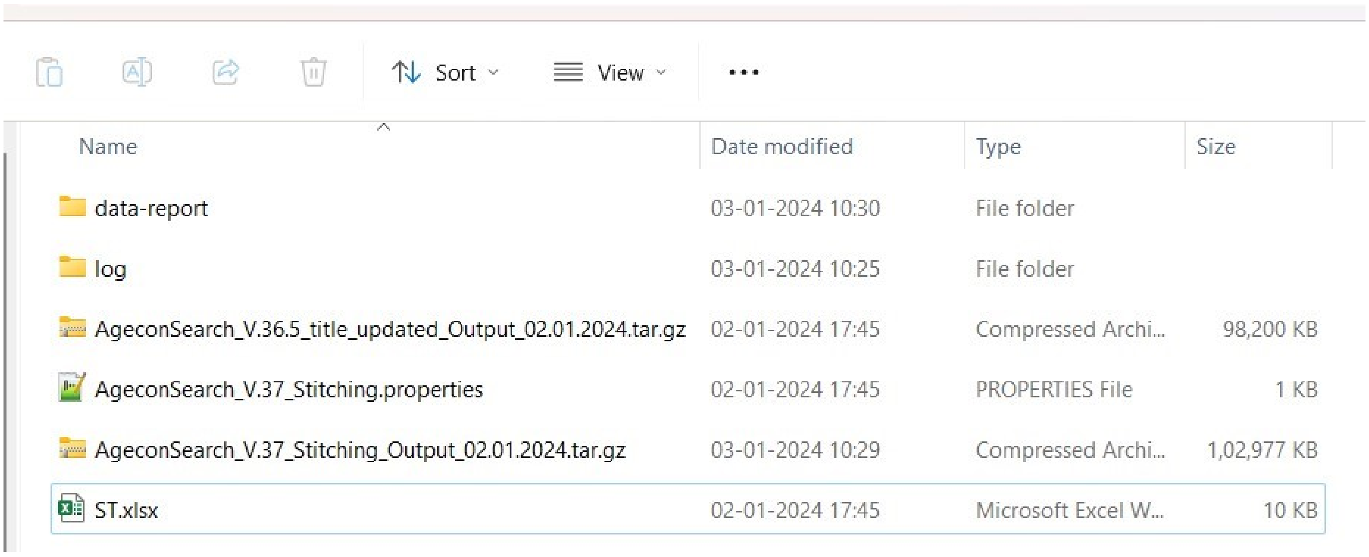
Name	Date modified	Type	Size
data-report	22-01-2024 02:48	File folder	
log	22-01-2024 02:48	File folder	
HID_Add.xlsx	21-01-2024 18:40	Microsoft Excel W...	15 KB
UM_Logic_HID_R1.1.csv	21-01-2024 18:39	CSV File	37 KB
UM_Out_R1.0.tar.gz	22-01-2024 02:47	Compressed Archi...	572 KB
UM_Out_R1.1.tar.gz	22-01-2024 02:48	Compressed Archi...	575 KB
UM_Run_R1.1.properties	21-01-2024 18:18	PROPERTIES File	1 KB

Figure 5: An Example of a Handle_ID Level Curation Bundle

3. **`Stitching Level Curation`**:
   The process of establishing a relationship among the items inside a collection is called Content Stitching. Although the concept looks intuitive from the standpoint of a source system hosting the data specific to its own context, for large aggregated digital document repository, it is first introduced in the NMEICT[2] project NDLI[3]. The `Content Stitching` method is used to establish the relationship across objects within the metadata management framework. In this way, the integrity and dependency of the related objects are maintained while displaying the desired content. The `Stitching Level Curation` is the most sophisticated way of curating the data amongst all the three methods. Although details about the Content Stitching procedure and its subsequent details have been explained in Section 6.1, we shall also briefly touch on the contents of the `Curation Bundle` when a data Stitching requirement is necessary to apply on the data. The Content Stitching template is a spreadsheet file written

[2] `https://nmeict.ac.in/`
[3] `http://project.ndl.gov.in/`

in `Microsoft Excel (.xlsx)`. Hence, the logic file parameter in the `*.run.properties` file becomes an Excel spreadsheet file pointing to the stitching template in the `Curation Bundle`. The execution flag for a `Data Stitching Level` mode curation is **-s**, identified after the name stitching module.



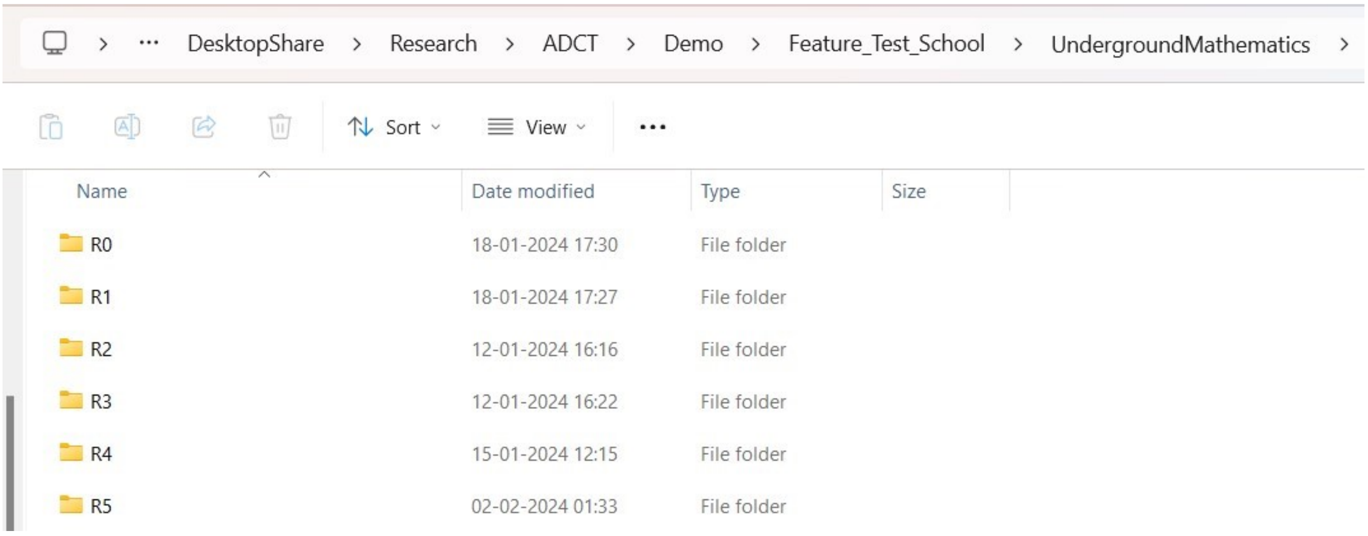
Name	Date modified	Type	Size
data-report	03-01-2024 10:30	File folder	
log	03-01-2024 10:25	File folder	
AgeconSearch_V.36.5_title_updated_Output_02.01.2024.tar.gz	02-01-2024 17:45	Compressed Archi...	98,200 KB
AgeconSearch_V.37_Stitching.properties	02-01-2024 17:45	PROPERTIES File	1 KB
AgeconSearch_V.37_Stitching_Output_02.01.2024.tar.gz	03-01-2024 10:29	Compressed Archi...	1,02,977 KB
ST.xlsx	02-01-2024 17:45	Microsoft Excel W...	10 KB

Figure 6: An Example of a Stitching Level Curation Bundle

Nonetheless, a `Curation Bundle` is a collection of files that are configured and provided for the ADCT systems to run curation logic onto a `sourceData`. It consists of the necessary data, configurations, and transformation files referred to in Run Properties `*.run.properties` and Logic Script `logic.json` file and to be enclosed inside the `Curation Bundle` folder. The physical name of the folder is necessary to be different for each iteration of curation as having the same folder name in the root directory of the source system in the process is not permitted by any operating systems. Hence, the concept of `Curation Bundle` is a logical one, and the physical names are being assigned by the user running and applying the various rounds of curation logic against the `sourceData`. Amongst the files, only the `run.properties` file is passed onto the ADCT system as a direct input, and the rest of the files are loaded as per the defined curation logic. Hence, all the necessary files must be kept inside the respective `Curation Bundle`, and no files are referred to either in the Logic Script or in the Run Configuration file, which does not exist inside the bundle. Though there are no technical limitations to running multiple curation logics from a single `Curation Bundle`, it is always advisable to make the curation bundle atomic to the curation operation that is being applied to the data. In the following section, we will see the benefits of maintaining the atomicity of a curation bundle concerning the applicable logic.

3.2 Run Configuration parameter file

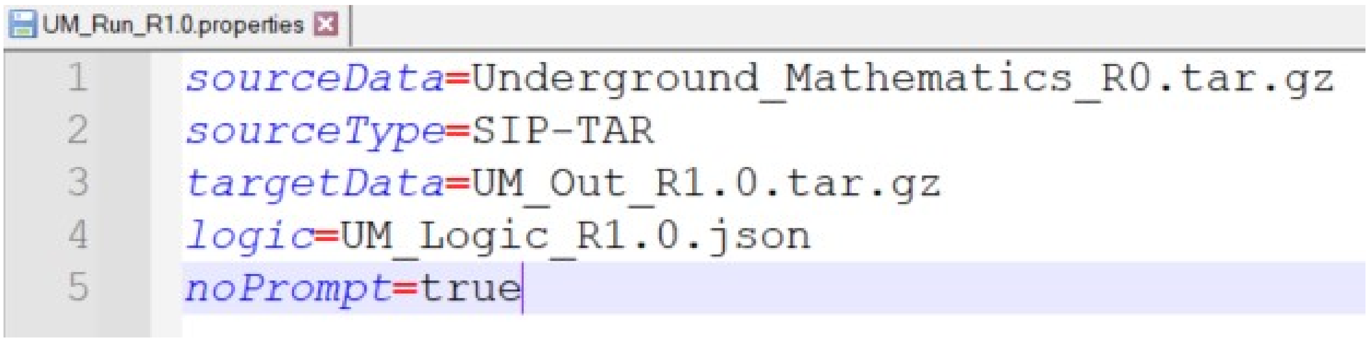
The Run Configuration Parameter File {**`.run.properties`**} is the second parameter of the CLI syntax of the ADCT system. It is, hence, a direct input to ADCT as it contains the information on the structural inputs to the ADCT systems. A few common structural parameters which the `*.run.properties` files consist of are, **`sourceData`** that defines which data to curate, **`sourceType`**, i.e., the type of the sourceData, whether it is a gzipped tar file or a folder containing SIP-XMLs or any other type of digital library archive files which are supported by the ADCT system. The parameter file also contains the name of output **`targetData`** after the transformation application, the name of the curation logic file and many other parameters. For the Collection Level run, it accepts a **`JSON(.json)`** file as the logic file for curation definition; for the Handle_ID Level run, it accepts a


DesktopShare > Research > ADCT > Demo > Feature_Test_School > UndergroundMathematics >

Name	Date modified	Type	Size
R0	18-01-2024 17:30	File folder	
R1	18-01-2024 17:27	File folder	
R2	12-01-2024 16:16	File folder	
R3	12-01-2024 16:22	File folder	
R4	15-01-2024 12:15	File folder	
R5	02-02-2024 01:33	File folder	

Figure 7: An Example of Multiple Curation Bundle in a data source Root.

`CSV(.csv)` file as the logic input file and for Data Stitching Curation the logic file passed to the ADCT system is a **`MS-EXCEL`** template with file extension as **`(.xlsx)`**. It is to be kept in mind that the file extension of the Run Configuration input is necessitated to `.properties` file extension as per the specification of ADCT systems. There are multiple other parameters that are part of the Run Properties configurations. The parameters definitions and their properties have been discussed in detail in the Subsection 4.1


UM_Run_R1.0.properties

```
sourceData=Underground_Mathematics_R0.tar.gz
sourceType=SIP-TAR
targetData=UM_Out_R1.0.tar.gz
logic=UM_Logic_R1.0.json
noPrompt=true
```

Figure 8: An Example of Run Properties file

3.3 Logic File

As mentioned above in Section 3.1, data curation can be achieved in three ways,

1. Collection Level Curation,
2. Item ID Level Curation here in as Handle_ID in the context of Digital Library data Archives and
3. Data Stitching Curation

For each method of data curation, the logic file is different and represents the underlying principle the curation method for which it has been applied. For example, in the **`Collection Level Curation`** process the Logic File, as mentioned in Section 4.2 is a `JSON(.json)` file. It is where all the curation logics for the required fields are written in one place of the particular `sourceData` in which the data transformation actions are applied, The logic file is written within the syntax of JSON to use an existing framework for the easier transition of the user to write logic and faster execution of curation instruction translation by ADCT as it can leverage standardized, current frameworks for parsing and any syntax correction. The following are the major components constituting a logic file for `Collection Level Curation`,

1. A mandatory **Field Translation Block** to write the transformation logic for the respective fields required to be curated.
2. An optional **Field Properties Block** to define the target field properties dynamically in case there is a requirement to override the default Field Properties as defined in the schema file.
3. An optional **Action Sequence** array to define the actions applicable to the Field value
4. The optional **Action Descriptors Block** respective to each action defined in the sequence

Amongst the above 4 options, any of points 2. or 3. needs to be present in a Field Translation Block (`FTB`), i.e., a `FTB` can not be left empty in a curation logic script for ADCT. Although a logic script as a whole can be left blank by only mentioning the root structure of it, i.e., the starting { and ending } brace to being the construct of the logic script, the `Fields` parameter and an empty JSON object.

```
{
  "Fields": {
    "ndl.sourceMeta.additionalInfo@note": {
      "action": [
        "lookUp",
        "copyData"
      ],
      "lookUp": {
        "inputFile": "lookUp_v2.xlsx"
      }
    },
    "asset.Handle_ID": {
      "sourcePriority": [
        "asset.Handle_ID",
        "ndl.sourceMeta.additionalInfo@note"
      ]
    }
  }
}
```

Figure 9: A Sample Curation Logic for Collection Level Data Curation

A blank logic script defines a default/blank run of the data, which performs housekeeping tasks such as conforming to the defined standards of the data, removing beginning and trailing spaces, removing multiple spaces from a text object and many more. Details of the default operation have been defined in Section 4.2.

There is no separate requirement for an IDE to write the curation logic for ADCT. It can be simply written on any text editor with the extension to save as a JSON file. Here we use the editor NotePad++[4] to write the JSON Logic Scripts(**`.json`**) and the Run Properties (**`*.run.properties`**)

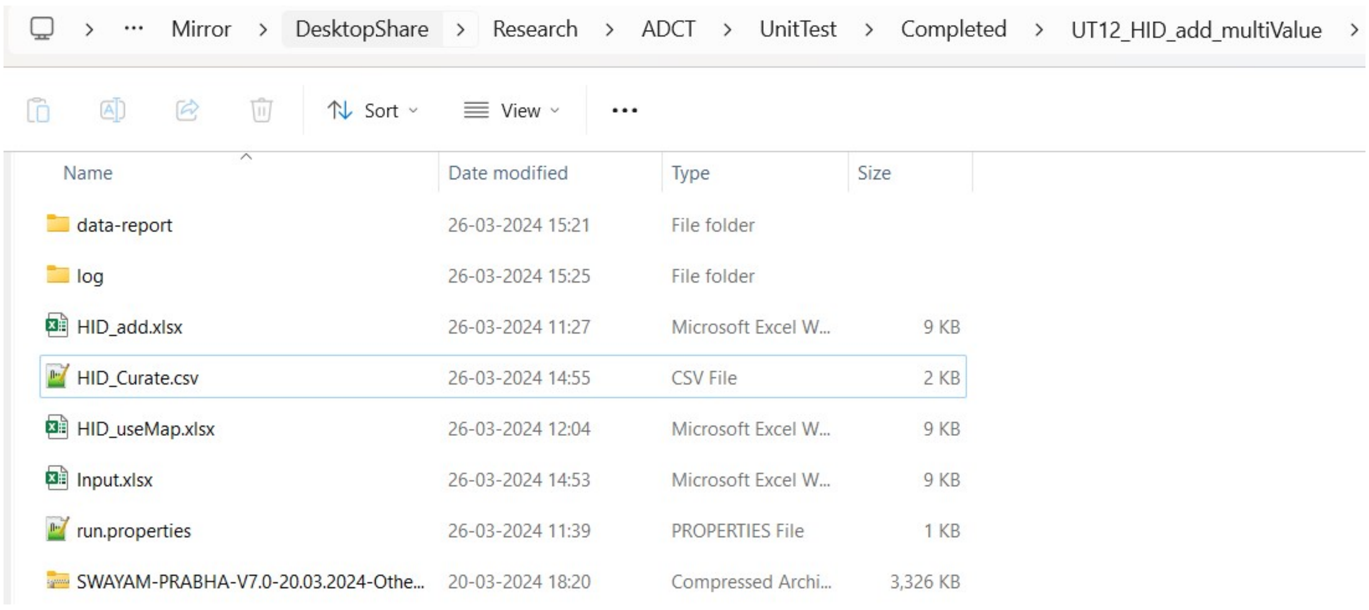
[4]`https://notepad.plus/`

```
{
  "Fields": { }
}
```

Figure 10: Blank Logic script used to perform default curations

files. The detailed definitions and elaborations of the parameters are described in the earlier Section 4.2

In the case of a `Item Id Level Curation` mode, the logic file processed is a Comma Separated Values `CSV(.csv)` file, which contains JSON scripts for individual items listed one after another in multiple rows. The CSV file contains two columns: the ID of the data by which individual items would be identified to apply the logic and the JSON script that is going to be applied to the data. In the current context of Digital Library Data Archival and Management Systems, the identifier commonly used is Handle_ID. However, apart from the common identifier, other IDs can also be used based on the configuration changes made to the ADCT System. The JSON script associated with each identifier is the same as the syntax and semantics used in the collection level curation; however, the only difference between them is that the curation logic, which can be applied in the case of an ID-based curation mode, is simple and straightforward. There are a few limitations that are applied in the case of an ID-based curation module, such as the absence of a filter action, column nesting inside a Fields Translation Block etc. Also, not all the actions are applicable in the case of an ID-based curation module since the transformation is straightforward and trivial. The actions that are available for the ID-based, and in the current context Handle_ID based curation system are useMap, copyData, deleteField and add. Complex actions such as moveField, lookUp, and attach are not applicable to the ID based curation module as the expectation is to have a straightforward transformation either depending on the value or to directly add values based on ID. Below is an example of a `ID/HID Level Curation` which depicts the above explanation.



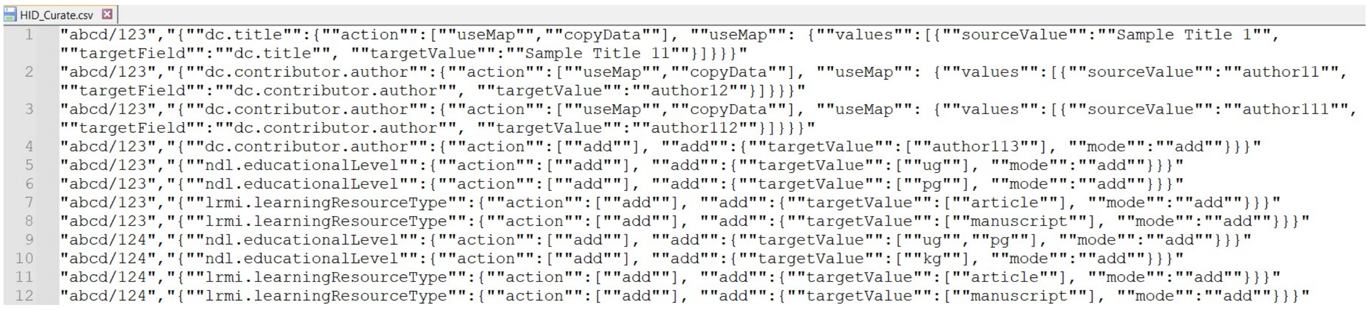
› ··· Mirror › DesktopShare › Research › ADCT › UnitTest › Completed › UT12_HID_add_multiValue ›

Name	Date modified	Type	Size
data-report	26-03-2024 15:21	File folder	
log	26-03-2024 15:25	File folder	
HID_add.xlsx	26-03-2024 11:27	Microsoft Excel W...	9 KB
HID_Curate.csv	26-03-2024 14:55	CSV File	2 KB
HID_useMap.xlsx	26-03-2024 12:04	Microsoft Excel W...	9 KB
Input.xlsx	26-03-2024 14:53	Microsoft Excel W...	9 KB
run.properties	26-03-2024 11:39	PROPERTIES File	1 KB
SWAYAM-PRABHA-V7.0-20.03.2024-Othe...	20-03-2024 18:20	Compressed Archi...	3,326 KB

Figure 11: Multiple HID Template in a Curation Bundle

In the above diagram, it can be seen that multiple ID(HID) based templates are created in the `Curation Bundle` in order to apply the curation logic applicable to the data. But, the thing to notice here is the templates are for basic actions, `HID_useMap` and `HID_add` as shown here. In the below image, it can be seen how these multiple templates are consolidated into a single Comma

Separated Values **`CSV(.csv)`** file and applied as a consolidated logic for the ID(HID) based curation round.



HID_Curate.csv

```
"abcd/123","{""dc.title"":{""action"":[""useMap"",""copyData""], ""useMap"": {""values"":[{""sourceValue"":""Sample Title 1"",
""targetField"":""dc.title"", ""targetValue"":""Sample Title 11""}]}}}"
"abcd/123","{""dc.contributor.author"":{""action"":[""useMap"",""copyData""], ""useMap"": {""values"":[{""sourceValue"":""author11"",
""targetField"":""dc.contributor.author"", ""targetValue"":""author12""}]}}}"
"abcd/123","{""dc.contributor.author"":{""action"":[""useMap"",""copyData""], ""useMap"": {""values"":[{""sourceValue"":""author111"",
""targetField"":""dc.contributor.author"", ""targetValue"":""author112""}]}}}"
"abcd/123","{""dc.contributor.author"":{""action"":[""add""], ""add"":{""targetValue"":[""author113""], ""mode"":""add""}}}"
"abcd/123","{""ndl.educationalLevel"":{""action"":[""add""], ""add"":{""targetValue"":[""ug""], ""mode"":""add""}}}"
"abcd/123","{""ndl.educationalLevel"":{""action"":[""add""], ""add"":{""targetValue"":[""pg""], ""mode"":""add""}}}"
"abcd/123","{""lrmi.learningResourceType"":{""action"":[""add""], ""add"":{""targetValue"":[""article""], ""mode"":""add""}}}"
"abcd/123","{""lrmi.learningResourceType"":{""action"":[""add""], ""add"":{""targetValue"":[""manuscript""], ""mode"":""add""}}}"
"abcd/124","{""ndl.educationalLevel"":{""action"":[""add""], ""add"":{""targetValue"":[""ug"",""pg""], ""mode"":""add""}}}"
"abcd/124","{""ndl.educationalLevel"":{""action"":[""add""], ""add"":{""targetValue"":[""kg""], ""mode"":""add""}}}"
"abcd/124","{""lrmi.learningResourceType"":{""action"":[""add""], ""add"":{""targetValue"":[""article""], ""mode"":""add""}}}"
"abcd/124","{""lrmi.learningResourceType"":{""action"":[""add""], ""add"":{""targetValue"":[""manuscript""], ""mode"":""add""}}}"
```

Figure 12: An Example of ID Level Curation Script(Handle_ID)

As previously mentioned in Section 3.1, the third type of Data Curation process applicable for Large Aggregated Digital Data Repository is `Data Stitching Curation`. The curation script template is a spreadsheet file with a file extension **`MS-EXCEL(.xlsx)`**. Details of this Stitching Curation Process have been explained later in Section 6. The logic file, also known as a `Stitching Template`, contains the mapping between various metadata fields of the data items and the hierarchical information of the data collection. It is very important to note that in `Data Stitching Curation` items, metadata values are not altered, i.e., no metadata transformations take place in a `Data Stitching Curation`. This process is specific to create the inter-relationships of the items and co-relate them togather. Each hierarchical level is identified by a newly created item termed as the `Parent Item`, or in some exceptional cases, an item from the existing collection is identified as the parent. The parent item is logical and hence is physically represented by an Item ID which in the context of Digital Library is the Handle_ID. The `Curation Template` file has the following columns `Level`, `TargetField`, `SourceField`, `SourceValue` and `Comments`. Though a detailed explanation of these fields is given later, we will summarize the explanation of these columns and the structure of the template used to establish the co-relation between objects herein known as `Content Stitching`.

1. **Level (mandatory):**
   The column `Level` denotes the Level of the hierarchy that is applicable to the data. Since the template is mapping information, every data item is supposed to contain the complete branch information to which it belongs. An item may or may not belong to a full branch branch containing source to end level, but it must have the complete information of the branch to which it belongs.

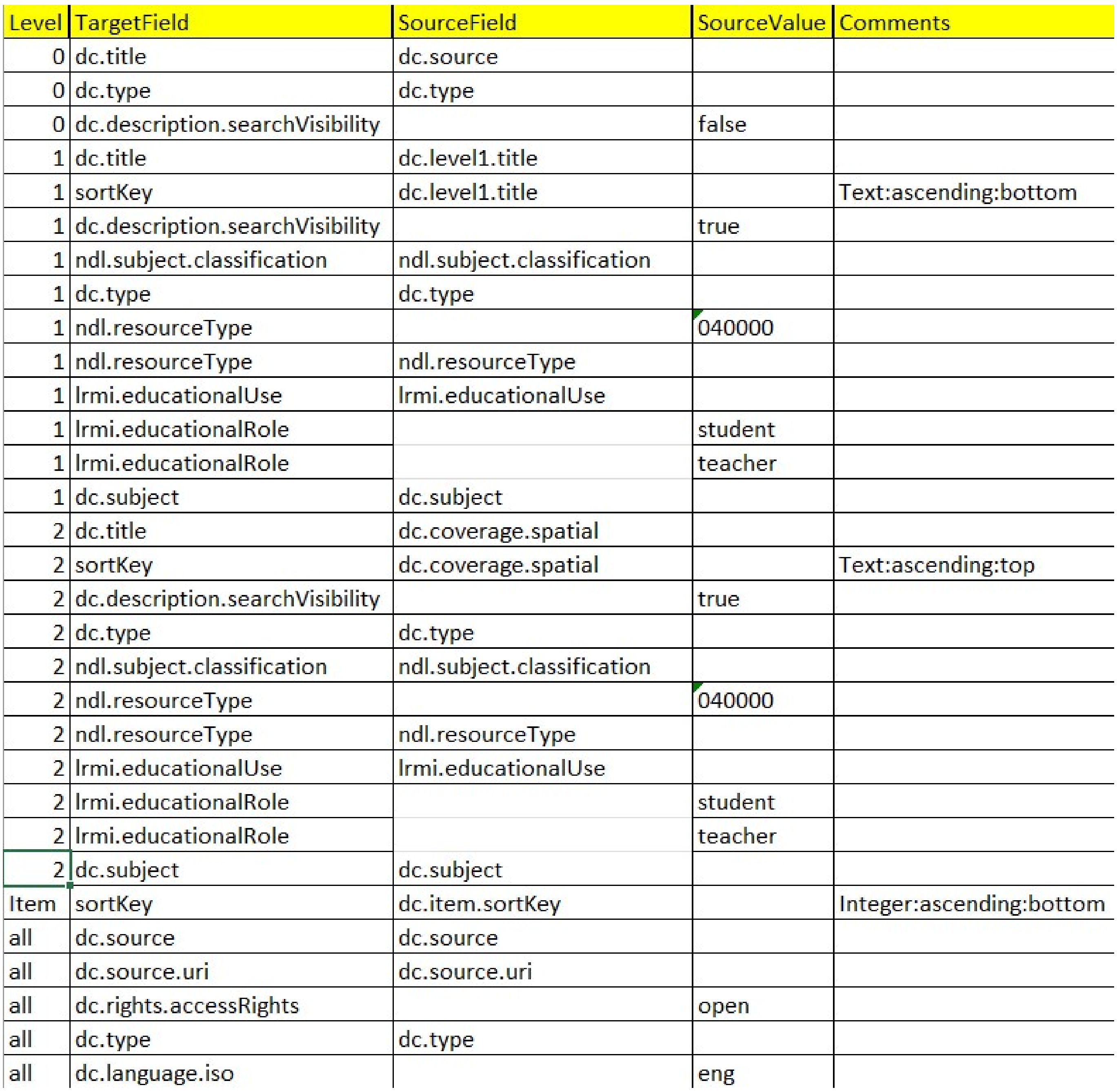
2. **TargetField (mandatory):**
   The column `targetField` mentions the metadata fields in the parent items, which are to be newly created in order to characterise the hierarchical structure. In these fields, information from the data items in which the level mapping fields are stored is captured. A hierarchy level can contain many information along with the level mapping information. The level mapping information is usually identified by the title, however for cases where there are physical items involved the hierarchy information is identified by the item ids. Again, in the context of the Digital Library Data Curation domain, the item id is mostly represented by Handle_ID. There are other specific details about the target metadata fields which are present in the parent data items which are discussed in the later sections.

3. **SourceField (optional):**
   Between the column `sourceField` and `sourceValue` either one is mandatory. They mention the mapping information for the `targetField` column in the Stitching Template. The column

`TargetField` as mentioned above holds the level characteristics information as mapped from the originating data items. However it is also to be mentioned that static values can also be assigned as the level field information. In case of the `SourceField` column the mapping information is denoted where values from metadata fields are designated as the hierarchy fields values. These mappings can be of single entry of multiple entries, i.e., a field in the `targetField` column can be mapped to multiple source fields of the data item. In case the `targetField` column is single valued the first allocated value to the target field would remain.

4. **SourceValue (optional):**
   As mentioned in the brief description of the column `sourceField`, either one between `sourceField` or `sourceValue` is mandatory. A hierarchy characteristics field can hold information from other fields as well as static values that are assigned to it externally. However such `targetFields` may hold both mappings from a `sourceField` as well as a `sourceValue` as per the analysis and justification by a domain curation expert. It is also important to note that in case of multivalued assignment of `sourceValue` to a `targetField` multiple mapping entries would be required.

5. **Comments (optional):** The column `Comments` is a significant column in the template, which provides information regarding the sorting order of a hierarchical level. Each hierarchical level is represented by the `targetField` named `dc.title` in the `Stitching Template`. The `sourceField` mapping value for that respective level would determine the name of the level in the collection hierarchy. A `sourceValue` can also be mapped to create a level, which is often used to create the root level of the hierarchy. There can be multiple entries for a particular level in a collection hierarchy. As we have earlier seen the main purpose of the `Data Content Stitching` is a consolidated display While displaying the entries of a particular level in a hierarchical manner. In this way, it sometimes becomes required to display the values for a level in a sorted manner. This sorting information is specified by a triplet structure denoted in the column `comments`. The structure is mentioned as **<datatType>:<sorting_order>:<position_of_orphans>**. More on the level sorting specifications and their application methods are explained in Section 6.4.5.



Level	TargetField	SourceField	SourceValue	Comments
0	dc.title	dc.source		
0	dc.type	dc.type		
0	dc.description.searchVisibility		false	
1	dc.title	dc.level1.title		
1	sortKey	dc.level1.title		Text:ascending:bottom
1	dc.description.searchVisibility		true	
1	ndl.subject.classification	ndl.subject.classification		
1	dc.type	dc.type		
1	ndl.resourceType		040000	
1	ndl.resourceType	ndl.resourceType		
1	lrmi.educationalUse	lrmi.educationalUse		
1	lrmi.educationalRole		student	
1	lrmi.educationalRole		teacher	
1	dc.subject	dc.subject		
2	dc.title	dc.coverage.spatial		
2	sortKey	dc.coverage.spatial		Text:ascending:top
2	dc.description.searchVisibility		true	
2	dc.type	dc.type		
2	ndl.subject.classification	ndl.subject.classification		
2	ndl.resourceType		040000	
2	ndl.resourceType	ndl.resourceType		
2	lrmi.educationalUse	lrmi.educationalUse		
2	lrmi.educationalRole		student	
2	lrmi.educationalRole		teacher	
2	dc.subject	dc.subject		
Item	sortKey	dc.item.sortKey		Integer:ascending:bottom
all	dc.source	dc.source		
all	dc.source.uri	dc.source.uri		
all	dc.rights.accessRights		open	
all	dc.type	dc.type		
all	dc.language.iso		eng	

Figure 13: Data Stitching Curation Template

3.4 Curation Template

The Curation Template is an audit trail file where all the actions and translation requirements applicable to the data from start to end are documented and their purpose and destinations are also noted. The need for this mandated documentation is because as user's tend to be more comfortable with mentioning their requirements in a textual verbose form, requirements are translated into MetaCur format as mentioned in the Section2 while finally executing the data curation logic onto the data. This would create a readability issue for the summary representation of the logic. Also since curation bundles separate logic into small atomic parts, the overall transformation of data from its original to final form becomes unavailable in a consolidated manner. In order to have a complete picture of the data transformation logic applied to the data from its source to target the `Curation Template` file is used. Since the file is for documentation and audit purposes and not required for input to the ADCT system it is not required to be maintained in the user's local file system. The `Curation Template` can be maintained in any online spreadsheet provider such as Google Sheets or MS 365 online version.

1. **`Status`**.
   The column `Status` depicts the execution status of the requirement as analysed from the correctness of the resultant data. It consists of four states,

   - **`Success`**.
     The state `Success` denotes that the curation logic has been successfully applied to the complete collection of the data and that the resultant state is okay. This means that no such occurrences have been present where the impact of the curation logic has translated into an erroneous result.
   - **`Error`**.
     As mentioned in the point above in case of an erroneous impact of the curation logic the state of the field is considered as an error and the state is marked as an `Error`.
   - **`Hold`**.
     The state `Hold` denotes that the particular curation logic has been kept on hold due to some reason which is necessary though not mandated to be written in the comments column.
   - **`RollBack`**
     From prior experiences, there are many times it has been observed that logic may be required to be rolled back due to its unsuitability to the next levels of curation requirements. In all those cases the curation logic applied for the current transformation level is cancelled and the data is changed to an earlier version before the current one. Since data cannot go back to a prior version as the version number is uni-directional a change in such cases is denoted by the state `Rollback` where all those requirement entries in the template are marked as `RollBack` in the column `Status` and the data is forwarded to the next minor version.

2. **`Version/Round`**
   The Column named version is used to denote the version of the data from its source to destination steps. The data versions primarily have two parts, i) major version, and ii) minor version. With these parts, the nomenclature of the target data looks like, $<$ `data_name` $>$ *_v* $<$ `major_version` $>$. $<$ `minor_version` $>$. A `major_version` of a data is put when there is a significant change of state that happens to it and the new state requires a review of the values and structure. The `minor_version` on the other hand deals with the change of state of the data where intermediate curation logics are applied based on a review of the user involved in the curation. Overall when there is a need to publish the data a major version is released, e.g., after mapping the data from an acquired state, doing the initial round of curation after analysing a sample set, and before and after performing the Content-Stitching process on the data, and before moving the data for quality checks and finally before moving into the production/live instance where a data is finally published for global users to explore and access it.

3. **`Command/Action`**
   The Column `Command/Action` denotes the action that is associated with the particular curation logic concerning the data. It helps in identifying proper tracing of the transformation executed on the data so that if required data versions can be properly reverted back to older versions and necessary corrections can be made to particular configuration files associated with the curation action as mentioned in the template.

4. **`Source Field`**
   The column `sourceField` denotes the field name for which the curation logic is applied. The sourceField must be the same as was written in the logic script. The main motive of this template as defined in the above Curation Template section 3.4 is to establish a connection between the logic script and the descriptive definition of the curation logics.

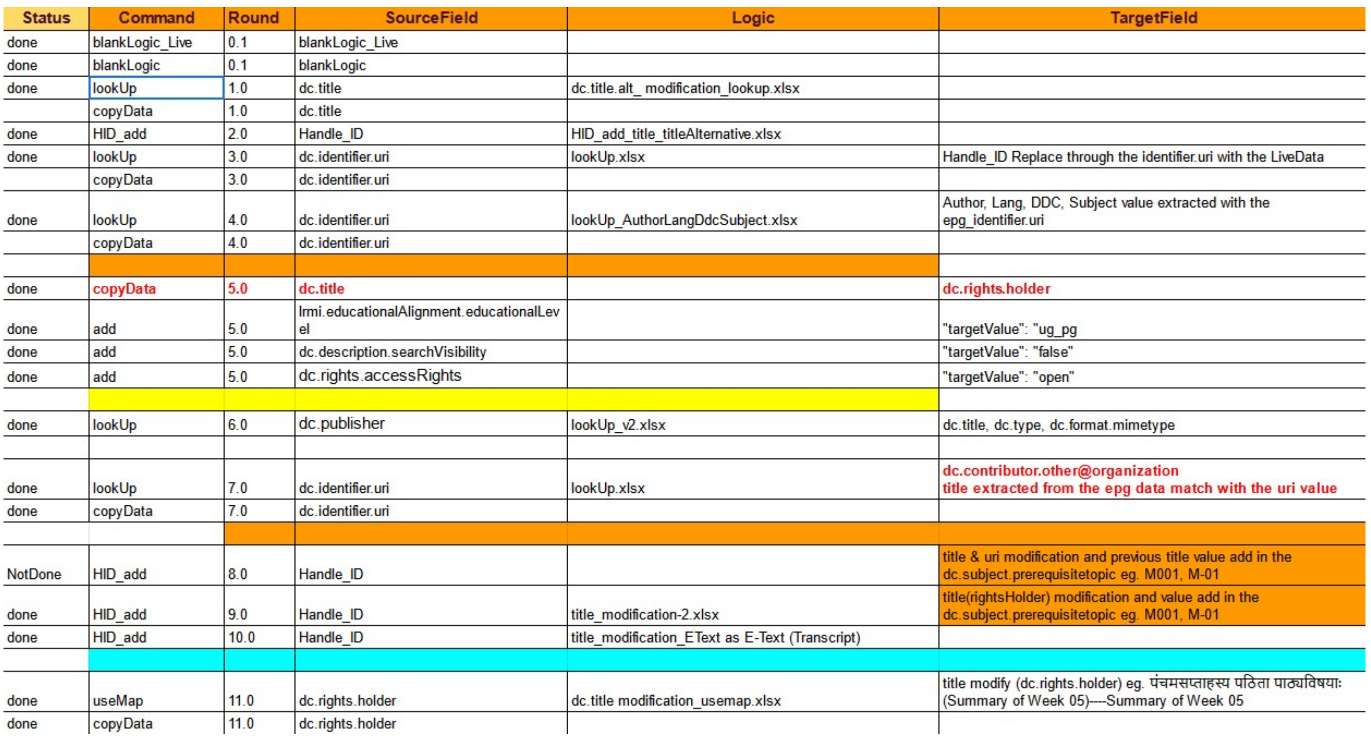
5. **`Logic`**
   The column `Logic` is used to write down the descriptive definition of the purpose of the curation activity with respect to the sourceField in context. It is the main purpose for which the whole `Curation Template` concept is conceived so that it aids the transition between a user from his conceptual domain of the curation logic to writing the `Logic File` as referred in the 3.3. The user is expected to write a description of the curation logic

6. **`Target Field`**
   The column `targetField` denotes the name of the fields post transformation of the data to which the curation logic has been applied with the action mentioned in the `Curaton Template`. It helps us the backtrace any data a user wants to debug or to find the lineage of of any metadata field across multiple versions of curation. The column `targetField` also helps to validate the syntactical correctness of the data by complying with the metadata standard and schema properties of the target system and necessary steps can be taken to convert from one target field to another based on the compliance requirement once the target field of the resultant data and its values are known.

7. **`Comments`**
   The column `Comments` is used for multiple purposes in the Curation Template. Amongst that one important usage is to note the information of the configuration files for the mentioned actions and nesting of commands if applicable in the curation process. THe column can also hold information of other user's comments which are useful to identify any specific activity necessary to identify the transformed data with respect to the defined logic.

It is heavily dependable on the sourceField and targetField. It is written in a spreadsheet, which may be written locally in an MS-EXCEL Worksheet or LibreOffice Calc File or it can also be written inside cloud-based software like Google Sheets, MS Office 365 web edition etc. Ultimately, the purpose is to keep the curation lineage information which can help later to understand the data curation cycle that has undergone to a specific Field-FieldValue pair. Each Action Instruction has its unique and static configuration file specifications that must be followed. Each Action specific configuration file has been discussed later in detail in their respective scopes inside Section 4.2. Below is an example for a Curation Template which will depict the column explained above.

In the below figure we can see the entries in the columns mentioned above. The rows depict the logic which has been applied to the data one after another. Here, we can see that the user has first applied a blankLogic script to the data. A blank logic as introduced in the Section 3.3 is a default curation steps which does basic housekeeping work in a dataset. The respective Version/Round information can be seen to be represented as `0.1` as it is the starting round, and no major curation has been performed. Hence, only the minor version has been incremented. In subsequent steps further logics have been applied and logged. Thus, the template gives an overview of the curation workflow performed on the data and the lineage of a field value can be tracked easily by looking into the workflow.



Status	Command	Round	SourceField	Logic	TargetField
done	blankLogic_Live	0.1	blankLogic_Live		
done	blankLogic	0.1	blankLogic		
done	lookUp	1.0	dc.title	dc.title.alt_ modification_lookup.xlsx	
	copyData	1.0	dc.title		
done	HID_add	2.0	Handle_ID	HID_add_title_titleAlternative.xlsx	
done	lookUp	3.0	dc.identifier.uri	lookUp.xlsx	Handle_ID Replace through the identifier.uri with the LiveData
	copyData	3.0	dc.identifier.uri		
done	lookUp	4.0	dc.identifier.uri	lookUp_AuthorLangDdcSubject.xlsx	Author, Lang, DDC, Subject value extracted with the epg_identifier.uri
	copyData	4.0	dc.identifier.uri		
done	copyData	5.0	dc.title		dc.rights.holder
done	add	5.0	lrmi.educationalAlignment.educationalLevel		"targetValue": "ug_pg
done	add	5.0	dc.description.searchVisibility		"targetValue": "false"
done	add	5.0	dc.rights.accessRights		"targetValue": "open"
done	lookUp	6.0	dc.publisher	lookUp_v2.xlsx	dc.title, dc.type, dc.format.mimetype
done	lookUp	7.0	dc.identifier.uri	lookUp.xlsx	dc.contributor.other@organization title extracted from the epg data match with the uri value
done	copyData	7.0	dc.identifier.uri		
NotDone	HID_add	8.0	Handle_ID		title & uri modification and previous title value add in the dc.subject.prerequisitetopic eg. M001, M-01
done	HID_add	9.0	Handle_ID	title_modification-2.xlsx	title(rightsHolder) modification and value add in the dc.subject.prerequisitetopic eg. M001, M-01
done	HID_add	10.0	Handle_ID	title_modification_EText as E-Text (Transcript)	
done	useMap	11.0	dc.rights.holder	dc.title modification_usemap.xlsx	title modify (dc.rights.holder) eg. पंचमसप्ताहस्य पठिता पाठ्यविषयाः (Summary of Week 05)----Summary of Week 05
done	copyData	11.0	dc.rights.holder		

Figure 14: Example of Curation Template

3.5 Data Files

ADCT system runs of data, as source data curation is the the primary purpose for which ADCT software was meant to be built. Data for ADCT can be of two types,

i) **sourceData**
The data which is fed into ADCT as a source system for curation is termed as the `sourceData` in run configuration file. The source data can be of many types, which are supported by the Digital Library systems. It can be a collection of XML files wither archived or non-archived. The archived XML files are known as `Archival Information Package (AIP)` and non archived XML files are known as `Submission Information Package (AIP)`. Both of them holds an extension of gzipped tar file (`*.tar.gz`) and are supported by the ADCT system. Source data can also be any XLSX or CSV or other type of files which are supported by the Digital Library Framework.

ii) **targetData**
targetData is the output file after execution of curation logic onto the sourceData. It is presently generated as a `Submission Information Package (SIP)` file either archived i.e, with file extension as `*.tar.gz` or collection of standard files and folders.

3.6 Action Configuration Files

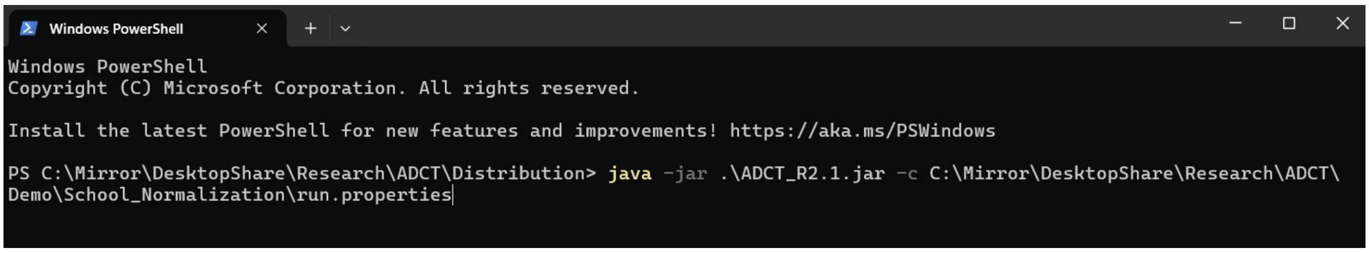
3.7 Other custom files mentioned in the run configuration to override the default configuration mechanism.

4 Collection Level Curation Methods

At collection level curation, the logic is applied to each record across the complete collection of data. The curation bundle for the collection level, as mentioned in Section 3.1, generally comprise the following components:

- Command Prompt Syntax,
- Run Properties Parameter File,
- JSON script logic File and,
- Associated command configuration files.

Each component will be discussed in detail in the subsequent sections. The Collection level Curation mode is run with the flag **-c** in the Command Prompt followed by the Run Properties (`*.run.properties`) file. An example of a command prompt syntax for the Collection Level run is shown below in Figure 15. The ADCT major version number at the time of this documentation was at v_2.0 which would subsequently change with further upgrades.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Mirror\DesktopShare\Research\ADCT\Distribution> java -jar .\ADCT_R2.1.jar -c C:\Mirror\DesktopShare\Research\ADCT\
Demo\School_Normalization\run.properties
```

Figure 15: The Command prompt Syntax for Collection Level Run

4.1 Run Configuration Parameters

The run configuration parameter file is a parameter file, which is the second component of the ADCT input system. The filename should hold an extension `.run.properties`. It enables the ADCT system to execute the logic file onto the source data and produce the desired output. The default assumption is that all associated files are available in the location of the run parameter file location. If files are provided in some other location, they need to be mentioned in the run configuration file as the value for `configLocation` parameter. There are three run modes of the ADCT system,

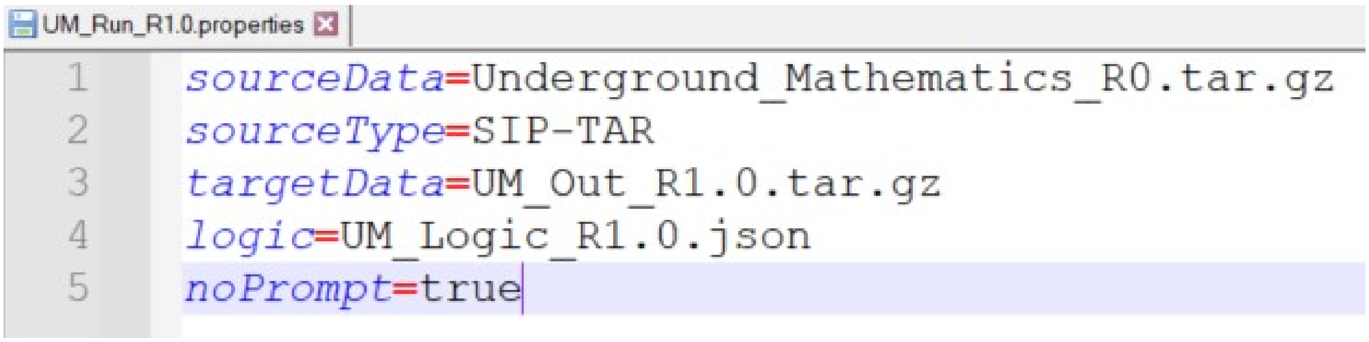
1. Collection Curation level,
2. Item Id / Handle_ID Curation level,
3. Data Stitching Curation level.

The Collection level mode is run with the flag **-c**, the Item_ID/Handle_ID level mode is run with the flag **-h** and the Data Stitching Curation is run with the flag **-s**. The configuration file has the following parameters, which are not required to be mentioned in any specific order; however, with changing modes, values of some specific parameters would change as is mentioned below. The parameters are of two types: Mandatory and Optional. A mandatory parameter is necessary to run the ADCT system. Without a valid value in the parameter entry, the ADCT system would throw an error.

Optional parameters are again of two types,

1. With default configured,
2. Additional Supplemental.

An optional parameter with a default configuration would be required in the system. However, they need not be explicitly mentioned in the Run Configuration Parameter file if the default value is not overridden. On the other hand, values of Additional Supplemental parameters are only provided to the system if the feature is required. Availability or Unavailability of a parameter would consequently turn on or turn off the respective feature. A sample `*.run.properties` file for the Collection Level run is shown below in Figure 16. Each possible parameter of a run properties file will be further elaborated in the following subsections.


UM_Run_R1.0.properties

```
sourceData=Underground_Mathematics_R0.tar.gz
sourceType=SIP-TAR
targetData=UM_Out_R1.0.tar.gz
logic=UM_Logic_R1.0.json
noPrompt=true
```

Figure 16: Run Configuration Properties File (*.run.properties)

4.1.1 Source Data Information

Param Name: sourceData
Usage: Mandatory
The parameter `sourceData` specifies the input data on which the required transformations are needed to be run. The sourceData can hold extensions of varied types as zipped tar files .tar.gz of SIP files, collections or SIP Folders, CSV data, RIS, XLSX, and BibTex files collections as well. Currently work is in progress to support MARC data files.

4.1.2 Source Datatype Information

Param Name: sourceType
Usage: Mandatory
The parameter sourceType helps to identify which type of source is being processed. This is explicitly required as collections of multiple source data can be processed in the system. Currently, the supported values are SIP-TAR, SIP-FOLDER, CSV, MS-EXCEL, RIS, BibTex. The extensions for other formats are currently being worked into.

4.1.3 Target Data Information

Param Name: targetData
Usage: Mandatory
The parameter targetData specifies the output file. The current output formats supported are zipped tar files or simple collection of SIPs.

4.1.4 Logic File Information

Param Name: logic
Usage: Mandatory
The parameter logic holds the filename of the logic description template. The parameter is mandatory as the basic functionality of ADCT system is to curate according to a logic description provided by a Digital Library Curator. For a Collection level (-c) run the logic description template is a JSON file. Hence any other filename extension in the logic file value apart from .json that would throw an error from the ADCT system. Similarly for Handle_ID level run the logic description file is required to be a CSV(Comma Separated Values) file. The logicFile parameter value for a Handle_ID level (-h) run would require to be a filename with .csv extension. The ADCT system would throw errors if the above specifications are not met.

4.1.5 Prompt Control Flag

Param Name: noPrompt
Usage: Optional
The run properties parameter `noPrompt` decares whether to prompt for parameter inputs or proceed with the default configurations as preset in the ADCT software default configuration settings.

4.1.6 Physical Folder Information

Param Name: dataReadPath
Usage: Optional
Provenance: Default Configured
The parameter dataReadPath mentions the physical data path in a zipped tar file. Since a tar file is nothing but an archive file that compresses the physical folders in a filesystem, it may contain other non-data files transported within the tar file. An SIP-TAR file may sometimes contain other configuration files generated out of the export module or be put inside to facilitate easier data transport; these would put extra headaches on the curator to separate those unnecessary files before processing for curation. Hence the parameter dataReadPath helps identify the data location inside such mixed file collections. The parameter is optional in cases where the collection contains no other files other than SIP data. The following Table 1 elaborates on various scenarios which may occur in the case of a SIP-TAR data file and the values of the parameter dataReadPath for such cases.

Usecase	**Parameter Value**
When the data is inside a folder named data immediately within the root folder which is same as the tar file name	data (Default Configured)
When the data is directly under the root folder location which is of the same name as the tar file	blank or /
When the data is under a specific folder inside the root folder which is of the same name as the tar file	the relative folder hierarchy without the root folder sign
When the root folder and tar name do not match	/folder name

Table 1: dataReadPath scenarios-param value

4.1.7 Data Audit Information

Param Name: audit_handle
Usage: Optional
Provenance: Additional Supplemental
The parameter `audit handle` specifies the `Handle_ID(s)` for which data audit reports are generated. The parameter enables the Data Audit module in the ADCT system. Details of this feature are discussed in the Data Logging and Reporting section 7.

4.1.8 Data Logging Information

Param Name: logPath
Usage: Optional
Provenance: Default Configured
The parameter `logPath` mentions the path where system logs will be generated. The ADCT system generates two types of logs,

1. System logs where the curation running instance information are notified
2. Data Audit logs, which note the data lineage information for the document(s) as mentioned in the audit_handle parameter.

The Default Configured path is the current directory from where the run configuration parameter file (`*.parameter`) is executed.

4.1.9 Configuration File Path Information

Param Name: configPath **Usage:** Optional
Provenance: Default Configured
The parameter configPath mentions the path towards config files which are used in the ADCT system. Among these config files one mandatory file is the logic file. The other files include various command configuration files such as useMap.xlsx(default configuration file for useMap command), moveField.xlsx (default configuration file for moveField command) etc. The Default Configured path for configPath is the current directory from where the run configuration file is executed.

4.1.10 Handle_ID Format Pattern

Param Name: Handle_ID Format
Usage: Optional
Provenance: Additional Supplemental
The parameter **Handle_ID format** is used while curating and converting data from other formats to SIP or any other data repository ingestible formats. Since the field **Handle_ID** 5 uniquely identifies an artifact in the collective repository, creation strategies for such remains an important task.

4.1.11 Data Transformation Schema Information

Param Name: schema
Usage: Mandatory
The mandatory schema parameter plays an important role in the ADCT system. Since in the assumption of ADCT the metadata of an artifact is represented in a Key-Value pair form, nesting of keys may occur. The schema as defined in the schema parameter would also required to be a JSON

file which holds this information. It simply defines the field as the key and its respective properties as JSON values of that particular key. The whole schema is represented as a single JSON object holding such multiple Key-Value pairs. The field entries in the schema file are not exhaustive, i.e., for fields which are not mentioned in the schema definition the ADCT system would not halt their operation. Instead it would proceed with a default assumption for the field behavior. Details of the schema file and field behavior are mentioned in section 3 under Field Properties sub-section.

4.1.12 Data Transformation Schema Type Information

Param Name: schemaType
Usage: Optional
Provenance: Default Configured
`schemaType` is a parameter specific to the application domain of the schema. It is based on the behaviour of the field, which might change based on the application domain while retaining the same name across all areas. The value of this parameter is specific to the application area where ADCT is being used. Combined, schema and schemaType parameter defines the representation of a field value. Currently, in the application area for NDLI, the schemaType values being used are **general**, **school** and **c&h**. The default value for the `schemaType` parameter currently used is general.

4.1.13 Data Service IP Information

Param Name: serviceIP
Usage: Optional
Provenance: Additional Supplemental
The `serviceIP` parameter defines the API endpoint for normalizing data representation. The API and its rendered services are coupled with the representative schema for the application area in which the ADCT system is used. In its current usage for the application domain of NDLI, the data normalization service is being used via Data Service V3 API. The current version of the service resides at http://10.72.22.155:65/services/v3 and also used in ADCT in the parameter `serviceIP`.

4.1.14 Field Separator

Param Name: CSVFieldSep
Usage: Optional
Provenance: Default Configured
Information for CSV files.

4.1.15 MultiValue Separator

Param Name: multiValueSep
Usage: Optional
Provenance: Default Configured
The run config parameter `multiValueSep` is used to capture multivalued field information for CSV and Excel files. CSV and Excel files are row-based data, and each record can only be assigned a single row. However, one item of the data can have multiple values in its fields.

A sample properties file would look like:

```
sourceData=Library_of_Congress-Ph16-Export-31.12.2021.tar.gz
sourceType=SIP-TAR
targetData=Library_of_Congress-Ph16-V1.0-10.02.2021.tar.gz
logic=Phase_16_round1_logic_01.02.2022.json
dataReadPath=Library_of_Congress-Ph16-Export-31.12.2021/data
serviceIP=http://10.72.22.155:65/services/v3
```

Table 2: Collection-curation.run.properties

4.2 Logic File

The logic file is the core building block of the curation activity. It contains the logic in a JSON format which is to be applied to individual fields to achieve the desired curation of the data. As briefly mentioned in the 2.1 curation logic can be primarily of three types: 1. Collection level curation, where the logic is applied across the complete collection of the data, 2. Handle ID level curation, which means the curation is applied at a Handle ID level where each Handle ID has separate logic for implementation, and 3. Content Stitching, in which internal relationships of the data are explored and links are established within them to display their hierarchical association.

Field Translation Block (FTB):

We have described the FTB in a nutshell above. We shall now describe in detail the three components of FTB and their respective principles and applications. A Field Translation Bock is the core part of the data translation description so understanding it in absolute clarity is important.

4.2.1 Field Properties

The Field Properties parameters combinedly describe the behavior of a field. It involves two components, i) A prior schema file definition, ii) properties key values mentioned in a FTB, however none of these are mandatory and absence of any of these would entail the default behavior specific of ADCT field parsing and translation. Hence in order to understand the complete working principle of Field Properties parameters we first need to describe the schema file and its parameters.

Field Properties Parameters in a FTB

The Field Properties parameters which can only exclusively be defined inside a FTB are dependency and sourcePriority. The parameter dependency defines the processing dependency between two fields, i.e., the translations of one field is processed after the one it is dependent is processed, and sourcePriority which controls the priority of values assigned to a particular field. It can be used to regulate the values assigned to a single valued as well as a multi valued field in the system. The structure of dependency and sourcePriority is described below:

1. Dependency:

The parameter dependency is mentioned as a key in the FTB and its values are mentioned as a JSONArray. The mentioned order of fields neither defines nor order of processing

2. SourcePriority:

- sourcePriority on lookUp.

Requirement 1:

There requirement is the replacement of handle_Id with the old data handle_Id on the new data handle_Id. So I have prepared a configuration file which is also called lookUp.

Requirement 2:

The lookUp would be run on sourcePriority, Because already there handle_Id are existing on the new data and have to replace with the old data, That is why have to be set a priority.

4.2.2 Schema File

The schema file contains the list of fields for which some properties would be associated. The list specifies but does not necessarily prevent any field from being processed that is not mentioned. This also aids in parsing complex data as they would come in a nested key-value pair form mentioned within a data value. An example of the Field Descriptions in the schema file is shown below:

```
{
  "dc.contributor.author": {
    "datatype": "person",
    "multiValued": true,
    "controlled": false,
    "validation": false
  },
  ...,
  "dc.identifier.other@edition": {
    "datatype": "Integer",
    "multiValued": true,
    "controlled": false,
    "validation": false
  },
  ...,
  ...,

  "ndl.sourceMeta.additionaInfo@note": {
    "datatype": "text",
    "multiValued": true,
    "controlled": false,
    "validation": false
  }
}
```

Figure 17: Example Schema File format in ADCT

We shall now explain the 4 parameters and their values which constitutes the description of a field in the schema, **i) datatype, ii) multiValued, iii) controlled, and iv) validation.**

i) **Datatype:**

The key “datatype" categorizes the type of data which is associated with the field. Again, since the language and curation philosophy is inclusive, conformity of the field value with the type is not enforced. However if required they can be enforced by making the validation parameter true. Datatypes defined here for these fields are not only primitive, i.e., String, Integer, Date, custom data types can also be specified as per the specification and definition of the underlying schema. When validation is turned on for a field the associated conversion of the data takes place and is checked against the specifications of the datatype mentioned. This can be done by making the respective change in the schema file or can also be passed as a parameter with the same name as “**validation**” in the Field Properties Parameter section while writing the requirement. An example of such is shown below:

```
{
    "Fields": {
        "dc.publisher.date":{
            "validation": true,
            "action" : [ . . .  ],
            ...
        }
    }
}
```

Figure 18: Usage of DataType in Field Normalization

While the dc.publisher.date is mentioned as a Field in the schema file as date, making the validation properties to true mean it would ensure that the dates passed on or being translated from other fields would render it in the form dd/mm/yyyy form or any other form as agreed upon in the specification of the schema. Details of the datatypes can be found more in the respective schema specifications in use.

ii) **MultiValued:**

The key “multiValued” is self-explanatory. It specified the existence of multiple occurrences of the key in the data item. Once set as false the first occurrence of the field is being assigned during translation. Any subsequent occurrences can be controlled by using a feature called “sourcePriority” in the Field Properties Parameter. We would explain the parameter in the Field Properties Parameter section below.

iii) **Controlled:**

The key “controlled” determines the valid values of a particular field. It is right now used as a placeholder to mark the property of the field and assignment of a valid value is left to the Digital Library Curator.

iv) **Validation:**

The key "validation” controls the behavior of a field. The functionality has already been described in the DataType section.

All the above Field Properties parameters can also be described in a FTB inside a logic description file.

4.2.3 Action Instructions:

The section Action Instructions describes the set of available actions which can be applied to the data. These actions are mentioned as a JSON Array value for the "action" key. Each action can be implemented standalone or they can be put together with other actions sequentially in order for the transformations to take effect accordingly. Actions can be either a halting action or a continuing action. A halting action means the consequent actions in the action container would not be executed after a successful execution of the current action and transformations returned from the current action are implemented to the data. However, for a continuing action the execution falls through the action container and each action which matches criteria for execution would be implemented till a halting action is encountered or the end of the container is reached. This means all the transformations during this process are collected together and are implemented to the data. Actions are associated with an action descriptor, which attributes the configurations of a specific action. If an action descriptor is not explicitly specified, default configurations are applied to that action. The following actions are currently available for MetaCur, which makes it quite exhaustive to achieve the required transformations. However there is always scope for introducing new actions and apply with the same method in the future. The set of current actions are **i) useMap, ii) copyData, iii) moveField, iv) lookUp, v) add, vi) deleteField, vii) merge, viii) curate, ix) attach and x) filter**. We would speak about these actions and their action descriptors in details below.

4.2.4 useMap

The action useMap facilitates the application of one-to-one mapping to the data. The exact key-value pair, as provided in the mapping details, when matched with the respective field-field value pair combination of an item, a transformation of the data takes place. There are two working levels of useMap action

1. `Collection Level`, where useMap action verifies every item in a collection to execute one-to-one mapping as specified in the configuration file
2. `Handle_ID level`, one-to-one mapping, as mentioned in the configuration file is executed to only specified Handle_Id s in the respective configuration file.

The matching and the transformation are mentioned in a configuration file which is either to be explicitly specified or configured default in its action descriptor. An exact equality between the data and the logic is verified based on the text value and the result is returned generating out of the mentioned transformation.

The Collection level useMap action has two associated components, 1) An **Associated Configuration File**, which specifies the underlying transformation details and 2) An **Action Descriptor** that specifies the action parameters. In the following section we would elaborate their specifications and usages in detail.

1) Specifications of useMap Configuration File:

The Configuration File is an .xlsx file consisting of four 4 columns namely **A.) sourceField, B.) sourceValue, C.) targetField and D.) targetValue**. It is extremely important to state that the column orders and their naming convention are static and the transformation would fail to execute if there is any change in the column order of the configuration file template of useMap, commonly known as useMap.xlsx. Also the template assumes only the first tab holds the designated data. Rest of the tabs even if present are ignored. The instruction works with an exact match between the information provided in the column sourceField sourceValue pair and that with the field-field value combination of the data. All the fields for which such translations apply are included in the configuration file. Hence the configuration file becomes a single file where all such occurrences can be listed. We shall explain the columns and their applications in details below:

1.A.) sourceField:

This column corresponds to the fieldname for which useMap instruction applies. This field name has to be mentioned in the Json logic file as a key and the action JSON Array should contain useMap in the FTB for this particular field in order for the transformation to take effect.

1.B.) sourceValue:

This column refers to the specific value of the sourceField for which on an exact matching the requisite translation would take place. The matching is case sensitive and textual. By case sensitive, a value "AB" mentioned in the configuration file would match with only "AB" as in the data and not with any other variation as "ab" or "aB" or "Ab". By textual it is meant that numerical equality or inequality or specific properties of any other datatypes as date, URI etc., can not be applied to the data. Hence a field though it is characterized as an Integer or date or any other datatype in order to use useMap action exact field value is to be provided in the configuration file.

1.C.) targetField:

The column targetField is part of the transformation process as mentioned in the description of the Action. For the action useMap a transformation can be one to one or one to many. This means a key-value pair upon a successful matching would shift the value to the respective field as mentioned in the targetField column. The value to be shifted can be the original value which is the same as the sourceValue applied for equality checking or any other value as specified in the targetValue. Important thing to note here is, if such a pair requires a transformation to multiple target fields separate entries are to be made in the useMap configuration file.

1.D.) targetValue:

TargetValue column mentions the value to be assigned to the targetField after transformation.This can be the original value of the field, some new value or some composite value containing the original value. Since transformations can be one to one or one to many, the 10 targetValue column is enabled to hold multiple values for the respective field mentioned in the targetField column. Unlike for targetField where individual target fields need to be mentioned in separate entries for a one to many mapping scenario, multiple target values destined for a single targetField can be written in one entry separated by a single character delimiter. However this does not prevent the user from writing different target values for the same targetField, in case the delimiter can not be specified.

Below we would show an example of the useMap configuration file(useMap.xlsx) and explain each property mentioned above with an entry from that.

Row-1 specifies the column names in the example file and are self explanatory. The presence and order of the header row in the configuration file is extremely important as the respective column values are considered accordingly.

In the `sourceField` column, it is important to note that the configuration file contains all those fields that apply the Action `useMap`. Here we can see the list of source fields contains `lrmi.learningResourceType`, `dc.description`, `dc.coverage.spatial`, `dc.contributor.author` etc. All these fields, in order to apply the desired translation in the data, need to specify the action useMap in the JSONArray value against the key action in their respective FTBs. Sample FTB

sourceField	sourceValue	targetField	targetValue
lrmi.learningResourceType	researchHighlight	remove	remove
lrmi.learningResourceType	notes	remove	remove
lrmi.learningResourceType	Script	remove	remove
lrmi.learningResourceType	report	remove	remove
lrmi.learningResourceType	journal	remove	remove
lrmi.learningResourceType	summary	remove	remove
dc.description	Manuscripts. 2 p., 8 1/2 x 13, mss. (unbound) |	remove	remove
dc.description	Manuscripts. 10 p., 8 1/2 x 14, mss. (unbound) |	remove	remove
dc.contributor.author	Jackson, James & Washington	dc.contributor.author	Jackson, James;Washington
dc.contributor.author	Moore, [Mrs] Robert	dc.contributor.author	Moore, Robert
dc.contributor.author	Military Rations	ndl.sourceMeta.additionalInfo@note	Contributor Names: `$Currvalue$`
dc.coverage.temporal	Zions Har, Boston, Massachusetts, September 28, 1892	dc.coverage.temporal	September 28, 1892
dc.coverage.temporal	Zion's Herald, Boston, Massachusetts, September	remove	remove
dc.coverage.temporal	Woonsocket, Rhode Island, 1982	dc.coverage.temporal	1982
dc.coverage.temporal	WM. H. Moore, Washington, D.C., December 14, 1887	dc.coverage.temporal	December 14, 1887
dc.coverage.temporal	WM H. Alden, Philadelphia, Pennsylvania, october 3, 1894	dc.coverage.temporal	October 3, 1894

Table 3: useMap

for the above example cases are mentioned below in Figure 19

```
{
  "Fields": {
    "lrmi.learningResourceType": {
      "action": [
        "useMap",
        "copyData"
      ],
      "useMap": {
        "inputFile": "useMap.xlsx"
      }
    },
    "dc.description": {
      "action": [
        "useMap",
        "copyData"
      ],
      "useMap": {
        "inputFile": "useMap.xlsx"
      }
    },
    "dc.coverage.spatial": {
      "action": [
        "useMap",
        "copyData"
      ],
      "useMap": {
        "inputFile": "useMap.xlsx"
      }
    }
  }
}
```

Figure 19: Example of a Field Translation Block applying the Action `useMap`

It is to be noted that the configuration file when named useMap.xlsx and kept at the config location would serve as the default configuration for the useMap action. Hence it need not be separately configured in the action descriptor block and consequently if no other action configurations are specified, no action descriptor would be required for such cases as shown above for field lrmi.learningResourceType and dc.description. However they can be explicitly mentioned in the action descriptor block too as shown for the field dc.coverage.spatial for clarity if the curator feels. Separate configuration files for separate fields are not necessary, unless otherwise required. We shall explain such cases in detail in the action descriptor part where multiple useMap configuration files might be required based on filters so it helps achieve different transformations on the same field - field value pairs on different criteria.

The column sourceValue as described above contains the values to be matched. As already mentioned above the matching is exact and textual, hence a value in an Integer/ date/ other field as defined by the schema is not converted to their respective data types and then matched ith the sourceValue entry in the useMap configuration file. Row # shows an example of sourceValue for the field dc.format.extent@pageCount where the configuration entry has been put as 07 for matching with the value 07 as it has appeared for the field dc.format.extent@pageCount. Here, even if the datatype of the field is described as Integer in the respective schema JSON file (herein, NDLIGeneralSchema.json) the sourceValue in useMap configuration is still specified as 07 and not 7 to ensure proper textual matching. The way to mention this in a .xlsx file varies from the software which is used to create such files. In most of the cases specifying the type of the cell as text serves the purpose otherwise starting the value with an apostrophe (') would mark the value entered in the cell as text in software's like Microsoft Excel. Similarly for fields having datatype as date matches the value in an useMap configuration file in a textual way. In Row# such examples are shown for the field dc.publisher.date. In this cases the field values are matched as they have appeared in the data and not by their contextual meaning. For example in Row# dc.publisher.date holds a value of 2021/01/03 which would exactly match the value appearing in the data in the same way, although the standard representation of the date datatype would be YYYY-MM-DD, i.e., 2021-01-03. Similarly Row# shows some more variety of such occurrences and their configuration entries in the useMap.xlsx file.

The targetField and targetValue columns are part of the transformation criteria. The column targetField would either hold a field name against which the targetValue would be assigned, or it may hold a special value called remove. The value "remove" in the target field is a reserved keyword which has the purpose of delete in the resulting transformation. If a key-value pair is matched using the configuration of useMap action then specifying remove in the targetField column would delete that particular key-value pair in the data. For any other value it is perceived as a field name and the corresponding target value is assigned as per the field properties mentioned in either Field Properties Block or in the schema information file. If a field name is not mentioned in any of these places, a field is default considered as following:

```
    "<FieldName>" : {
"datatype":"text",
"multiValued":true,
"controlled":false,
"validation":true
}
```

The final column, targetValue, assigns the provided value or expression to the specified target-Field post transformation. The type of assignment can be of three types,

- delete (the matching pair from data),
- single value assignment,
- multi-value assignment.

If both the targetField and targetValue columns contain the reserved keyword **`remove`**, the matching key-value pair from the data is deleted. Single value assignment can be of two types, static value assignment and dynamic value assignment. Both these assignments transform the matched key-value to the target forms except for dynamic assignment where target values can be substituted using a reserved keyword **`$Currvalue$`** or **`$<field-name>$`**. If the targetValue column does not contain these keywords then the exact value mentioned is assigned to targetField after transformation. This is a static value assignment. For dynamic value assignment, the notation **`$Currvalue$`** or **`$<field-name>$`** would resolve to their specific references, and the resultant value would be assigned accordingly to the target field post transformation. The notation **`$Currvalue$`** refers to the current value to which the action useMap has been applied. In Row# it can be seen that if a matching occurs for the field-value pair “dc.contributor.author” and “Military Rations” a transformation takes place which assigns the value `Contributor Names: $Currvalue$` to the field `ndl.sourceMeta.additionalInfo@note`. Now during the transformation, **`$Currvalue$`** is resolved to the matching value of the sourceField `dc.contributor.author` which is **`Military Rations`** and the final assignment to ndl.sourceMeta.additionalInfo@note becomes **`Contributor Names: Military Rations`**. For **`$<field-name>$`** substitution, the respective field value from the source data is replaced in place. In case the field is not present in the source data, the particular notation is removed, and the rest of the value is assigned to the mentioned target field. A multi-value assignment is just a repetition of any form of single-value assignment separated by a delimiter. As is seen in Row1 targetValue column contains values separated by a delimiter **`;`** (semicolon) for targetField. This would make the transformation to split the mentioned entry in the targetValue <> column by “;” and assign each value to the mentioned field name in the targetField column. One important thing to note here is that the delimiter can be any character based on the decision of the DL curator and need not be explicitly the “;” character. It also needs to be mentioned in the action descriptor if the delimiter has been used for a multi-value transformation in the targetValue column. Otherwise, the whole value is treated as a single value, and necessary substitution would take place as per the process of the single value assignment method defined above. The presence of a multi-value transformation does not prevent any single-value transformation from taking effect. However, if a transformation is intended to be single-valued and the multi-valued delimiter is present within, it would split accordingly to the delimiter, and multiple values would be assigned to the mentioned target field if supported. For a single-valued field, the first part of the split entry would be assigned, and the rest of the parts would be ignored.

2) The Action descriptor for useMap contains the following:

The curation bundle for the collection level useMap action comprise of a "*.run.properties file" that accepts a Json script as logic file. A sample Json script to exemplify useMap action is shown below where the Field Translation Block constitute Fieldname, Action Block and Action Descriptor as discussed in the earlier section of properties of FTB. It is important to remember that the ***Fall through property for useMap action is False and copyData command is mandatory with useMap action to avoid data loss.***

Action Descriptor for useMap action comprises of the following parameters: **A) a filter action(optional)**, to specify the filter criteria for useMap to apply, **B) an action container to hold nested actions (optional) C) parameter inputFile**, to identify the configuration file and **D)parameter delimiter (optional)**, to specify the multi value separator character. A sample useMap action descriptor looks like the following:

```
    "useMap" : { "filter": [
... // filter specifications
],
"inputFile": "<useMap-configuration-file>",
"delimiter":"<the delimiter character>",
"action":[
{
 "Fields": {
    "<Field_Name1>": {
        "action": ["useMap",
                   "copyData"],
        "useMap": {
            "filter": [//filter specifications],
            "inputFile": "useMap.xlsx",
            "delimiter": "<delim>",
            {"Fields": ... } //Nested Action
            }
        }
    }
}
```

Figure 20: `useMap` Action Descriptor Example

... // nested actions
],
}

In the below section We will explain each parameter and its specifications.

A) Filter:

A filter action is an instruction specified by the key “filter”. When it is mentioned in the action descriptor, the useMap configuration is applied to only those data which matches the filter criteria. A filter in useMap consists of its own specification parameters along with the other parameters of useMap. The construct for a filter when applied to the useMap command looks the following:

"filter": [
"filter1",
"filter2",
...
],
"filter1": {
// filter specifications
"inputFile": "<useMap-configuration-file-for-filter1.xlsx>" // Mandatory
"delimiter":"<delimiter-character>" // Optional, only if multivalued transformation
},
"filter1": {
// filter specifications
"inputFile": "<useMap-configuration-file-for-filter2.xlsx>" // Mandatory
"delimiter":"<delimiter-character>" // Optional, only if multivalued transformation
},
...
]

We shall explain the filter action separately in its section.

B) Nested Action Container:

A Nested Action Container holds any subsequent actions after a successful application of useMap to the input data. These actions are applied to the transformed data and not on the original field-field value combination since these are nested within one another. Nesting of command can be upto any level, but the DL curator has to be cautious so as not to introduce repetition and circular transformation arising out of the nested action as the whole utility is designed towards inclusiveness of the system and does not validate the curator's decision of data management. The list of nested action is specified by the "action" key and a JSON Array construct as the value holding the action instructions in a sequential order.

C) **InputFile:**

The inputFile parameter specifies the name of the useMap configuration file. The key is "inputFile" and the value is the name along with the file extension .xlsx mentioned as a String JSON value. The default name of the configuration file is denoted as useMap.xlsx when the inputFile parameter is not specified but during the curation execution it would be prompted for verification if indeed the file is to be used. The following is the screenshot from the curation execution for inputFile configuration interaction with the user.

The text in green is the user input provided during the verification step. One point to be noted is that all configuration files have to reside within the config location of run.properties configuration file. Now referring to the discussion that when it is necessary to use the inputFile parameter, multiple fields can have different delimiters for their multi value transformation configurations depending on the data and keeping in mind for the simplicity of useMap configuration addition of another column delimiter is not efficient. Hence in that particular case separate useMap configuration files can be created for the field having a different delimiter than the others. Another use of distinct input file specification is within filter action. The filter action consists of its own specification along with the parameters of useMap, i.e., based on a filter criteria certain transformations can behave differently for the same data. In those cases the useMap configuration file inside the filter action would be different than that of the configuration file for a non filtered transformation. It is to be also noted that inputFile is a mandatory parameter inside a filter action in useMap and if used then the parent configuration file becomes optional, in sense, thus facilitating the variation of application of transformation in more ways. We shall see all these examples and their applications in detail in the use cases section.

D) **Delimiter:**

The parameter delimiter is specified by using the key "delimiter" and the delimiter character(s) as a JSON String value in the action descriptor for the purpose of multi-valued transformation. The delimiter parameter can be a single character or a set of characters as applicable for the transformation defined in the target-Value column of the useMap configuration file. It has already been mentioned above that if multi valued transformations are mentioned in the configuration file but the delimiter is not specified in the action descriptor the transformations would be treated as single value transformation and the substitution rules would be applied accordingly, if applicable.

It is to be noted that none of these parameters are mandatory, which means absence of an action descriptor for useMap action would not prevent the action to be applied on the data. However, in that case a default configuration to the action description is applied for the mapping and transformations to take effect. In case of a default configuration there would be no filter action, no nested action and no delimiter applicable for this action, only inputFile parameter would be default assigned to the value useMap.xlsx as a default identifier for the useMap action configuration file.

4.2.5 copyData

This action copies the data from one field to another based on the configuration defined in its action descriptor or to be configured default when no action descriptor is present. The action has to be specified inside an action container of a FTB in order for the copyData to take effect for that field against which the action is specified. As mentioned earlier the action copyData can be specified as a standalone action for the field or it can be put together with other actions inside the action container to be executed in an ordered way. The action copyData terminates upon a successful execution, otherwise the control falls through the rest of the actions if defined. It is a halting action. It is also important to note here that there are subtle features that make a simple copyData action interesting and non trivial. Remembering from earlier, it was mentioned that if no action has been specified for a particular field it is defaulted to copyData with targetField parameter in the action description configured to self. However, if an action container has been defined for a field and any action has been mentioned inside the container then it would be required to mention copyData command explicitly if the unmatched data is required to be copied to the desired target. Otherwise when fall through occurs in an action container absence of copyData would result in removal of the specific data.

The Action Descriptor for copyData consists of the following parameters **i) Target Field(targetField),ii) Target Value(targetValue), iii) Delimiter (delimiter), iv) Nested Action Description(action), v) Filter**. The description of Filter Action and Nested Action Description has already been covered in earlier sections(<mention section ref.>) we would only focus on the copyData specific action descriptor parameters here.

I) **`targetField`**:

The parameter targetField defines the field name to which the data would be copied over. In case of a default action description, where the descriptor is not explicitly mentioned, the value of targetField is the same as the source field name. However when the action description is mentioned for copyData defining the targetField name is mandatory. This scenario arises when the action descriptor for copyData contains filter action. If a filter is defined inside the copyData descriptor the absence of targetField parameter for copyData outside the filter action would signify the removal of the specific field when one or more filter matches are not found. The parameter targetField can hold a single field name or it may also hold multiple field names mentioned as a JSONArray in case of replicating the source field value to multiple target fields.

- **Target Value(targetValue):**

Parameter targetValue has a similar implementation as targetField inside the action descriptor of copyData, implemented as a JSON Key. The value to the parameter is also similarly can be single valued as well as multi values which is represented using the JSONArray construct. One important token which can be used inside the target-Value parameter is *value*. The token *value* is a meta token containing a specific application for ADCT. Wherever it is used the token is replaced by the source value which is undergoing the translation. E.g., Let us say that a copyData translation is applied to a Dublin-core (dc.) field dc.creator.researcher and inside the action descriptor block the value for the parameter targetField is mentioned as dc.contributor.author and the respective value for parameter targetValue is mentioned as Prof. *value*. The outcome of this translation would result in appending Prof. token before each value of the dc.creator.researcher field and moving them to dc.contributor.author. Below is the implementation of targetValue key in the action descriptor of copyData as mentioned in the example above

```
    "dc.creator.researcher": {
"action": [
"copyData"
],
"copyData": {
"targetField" : "dc.contributor.author"
"targetValue": "Prof. value"
}
}
```

It is important to note that none of the keys in the action descriptor are mandatory, however an empty action descriptor would throw an error. The parameter keys are mutually exclusive.

4.2.6 moveField

This action transforms one field-field value pair to one or multiple based on the matching criteria. The main difference between this action and the action useMap is that useMap is a very specific action and has been optimized for faster performance as it checks for an exact equality between the data and logic. It serves only one operation and is introduced based on empirical analysis. So understanding the difference between them is important for when to use which action.The working principle of moveField action involves the stated transformations. Upon successful matching criteria or pattern identification on the basis of certain match value/expression: (i) the source field may retain the source value post transformation as specified in the configuration file, (ii) the source value may be shifted to the target field where the target value may be same as the source value without any transformation, (iii) the source value may be shifted to the target field, where the target value may be a new value as specified in the configuration file, (iv) the source value may be shifted to the target field, where the target value may be a composite value while retaining the original value with new value specified in the configuration file, (v) source value may be completely removed from the data.

Similar to useMap action, the curation bundle for moveField action at the collection level comprises of the following: **1) Command Line Syntax, 2) Run Configuration Parameter File, 3) JSON Script Logic File, 4) Associated Configuration File**

1) Command Line Syntax:

As already discussed in earlier section, collection level mode is run with -c flag for any action instruction. It is similarly applicable for the moveField action as well. It will facilitate the execution of the moveField action to every content in the collection.

2) Run Configuration Parameter File for Collection Level_ moveField action:

In the curation bundle Run Configuration Parameter file will accept a **JSON script (*.json)** as a logic description file with an associated configuration file with specifications of the desired operation.

3) JSON Script logic file for Collection Level_moveField action:

Importantly, while writing a JSON script file for the moveField action it must be noted that ***Fall through property is positively False and copyData command is mandatory***. Generic structure of Collection Level_moveField_Json Script is shown below. Each property of the JSON Script for moveField action will be discussed in detail further below.

```
{
"Fields": {
"<Field_ Name1>": {
"action": [
"moveField",
"copyData" ],
"moveField": {
"filter": [ //filter specifications],
"inputFile": "moveField.xlsx" "delimiter":"Value2"
"action":[//nested action]
} }
}
}
```

Each field for which moveField action has to be applied has its respective Field Translation Block (FTB). Each FTB will have the following components that will be discussed in detail below: i) Key Value, ii) Action Block, iii) Action Descriptor

i) **Key Value:**

`Field_Name1` key in the indicative JSON script above corresponds to the source field for which moveField action has to be applied. The `Field_Name` key in the Field-FTB Key-Value pair holds the JSON Object as the value. This JSON Object Value encloses the curation instructions in a in JSON Array construct that are to be performed on the `Field_Name1`. The action **`moveField`** mentioned in the action array instructs the program to search the `Field_Name1` for the matching criteria specified in the associated configuration file as defined in the action descriptor.

ii) **Action Block:**

A set of `Actions` inside the "action" parameter corresponds to the JSON Object Value as the Key-Value pair for "<Field_Name1>". It is the key for the action container in the FTB. “moveField” function that has to be applied on the "<Field_Name1>" will be enclosed inside a JSON Array Construct named Action as JSON Object Value in Field Translation Block.

iii) **Action Descriptor: (Optional)**

“moveField” action is described by its Action Descriptor. The Action Descriptor specification for the “moveField” function looks like the following.

```
{
  "_comment": "other specifications of a FTB above",
  "moveField": {
    "filter": [
      "filter specifications"
    ],
    "inputFile": "moveField.xlsx",
    "delimiter": "Value2",
    "Fields": {
      "_comment": "nested action"
    }
  }
}
```

Figure 21: moveField Action Descriptor

Action descriptor further comprises of some mandatory /optional parameters that are discussed below.

Filter: (Optional)

It is an action specified by the key “filter”. It is an optional parameter in the action descriptor. If it is mentioned in the Action Descriptor, “moveField” will apply to only those data that match the filter criteria. If the “filter” action is not mentioned in the Action Descriptor, then the “moveField” action will apply to every item in the collection. A “filter” action in an action descriptor has its particular specification parameters along with the other parameters of “moveField” action.

Parameter Input File (inputFile):

This is the Key to specify the associated configuration file.It is important to note that **i)** if moveField action is required to be executed on more than one metadata field, then it is not needed to create separated moveField.xlsx for each field. The curation requirements for each field that have are categorized under moveField action can be incorporated in a single moveField.xlsx configuration file.**ii)** if the configuration file follows the naming convention moveField.xlsx and kept at the config path, then the program automatically picks up the file by default configuration and there is no need for the action descriptor with the parameter inputFile.

Parameter “delimiter”: (Optional)

It is specified by the Key “delimiter”. It defines the multivalued separator character(s) or delimiter character(s) as a JSON string value in the action descriptor to facilitate multivalued transformation. The delimiter parameter can be a single character or a set of characters as applicable for the transformation. If the multivalued transformations have been mentioned in the configuration file without delimiter specifications in the action descriptor, the transformation would be treated as a single-valued transformation.

As in the case of useMap action, none of these parameters are mandatory in action descriptor, which means absence of an action descriptor for moveField action would not prevent the action to be applied on the data, unless otherwise required. However, in that case a default configuration to the action description is applied for the mapping and transformations to take effect. In case of a default configuration there would be no filter action, no nested action and no delimiter applicable for this action, only inputFile parameter would be default assigned to the value moveField.xlsx as a default identifier for the moveField action configuration file.

Nested actions: (Optional)

It is an optional parameter. It holds any subsequent actions after a successful application of the “add” action. These actions are applied to the transformed data and not on the original field-field value combination since these are nested within one another. Nesting of command can be up to “n” number of levels, but the Digital Library curator is advised to be cautious enough to keep control of repetition and circular transformation arising out of the nested action. The whole utility has been designed towards the inclusiveness of the system and does not validate the curator’s decision regarding data management. The list of nested actions is specified by the “action” key, and a JSON Array construct as the value holding the action instructions in sequential order.

4) Specification of the moveField Configuration File:

moveField action has an associated configuration file with specifications of the desired operation. The file is an .xlsx file, generally following the naming convention as “moveField.xlsx”. In the action descriptor for lookUp function, parameter “inputFile” serves as the key to hold “moveField.xlsx” config file name as key-value pair as shown below.

```
    "moveField": {
"inputFile": "moveField.xlsx"
}
```

(i) It is noted that the configuration file when named moveField.xlsx and kept at the config location would serve as the default configuration for the lookUp action. Hence, it need not be separately configured in the action descriptor block and consequently if no other action configurations are specified; no action descriptor would be required for such cases (as shown below for the field “dc.description") . However, they can be explicitly mentioned in the action descriptor block for clarity if the digital curator feels so. **(ii)** Separate configuration for each field is not necessary, unless otherwise required. All the fields requiring such translation will be configured in a single file. In the below example, a single moveField.xlsx file has been used for "dc.description", “dc.title” and “dc.coverage.spatial”.

```
{
  "Fields": {
    "dc.description": {
      "action": [
        "moveField",
        "copyData"
      ],
      "moveField": {
        "inputFile": "moveField.xlsx"
      }
    },
    "dc.title": {
      "action": [
        "moveField",
        "copyData"
      ],
      "moveField": {
        "inputFile": "moveField.xlsx"
      }
    },
    "dc.coverage.spatial": {
      "action": [
        "moveField",
        "copyData"
      ],
      "moveField": {
        "inputFile": "moveField.xlsx"
      }
    }
  }
}
```

Figure 22: An example of the Action `moveField`

Now in moveField, the action is similarly configured like useMap, i.e., it has a configuration file to specify the details of the desired operation; however the columns that formulate the logic are different from that of useMap. The configuration file for moveField consists of the following 8 columns namely **(i) sourceField, (ii) matchGroup, (iii) matchType, (iv) matchValue, (v) targetField, (vi) transformType, (vii) targetExpr, (viii) targetReplace**. The column naming convention and the order is static. In the following part we would explain the meaning and working principle of each and every configuration column.

//Generic template of a moveField.xlsx file is shown below:

sourceField	match_ group	src_ exprType	src_ expression	targetField	tgt_ exprType	tgt_ expres

i) **sourceField:**

This column specifies the field name to which the transformation logic is to be applied. Since a configuration file is specific to the instruction rather than being specific to the field all fields which undergo the transformation are contained in the configuration file. In the Json logic file, this field name has to be mentioned as a key and its action JSON Array should contain moveField in the FTB for this particular field in order for the transformation to take effect. All the metadata fields where the moveField action is applicable should be mentioned in a single configuration file under the column sourceField.

ii) **match_ Group:**

The instruction moveField is enabled with all possible transformation types that can happen to a field. Hence it supports sequential as well as parallel application of logic. The column matchGroup enables the instruction to achieve that task. When a set of logics are mentioned for a sourceField moveField applies all logics to the source data in parallel, i.e., if multiple transformations exist for a value of a particular field all those are captured. However this property can be controlled using the matchGroup column. A set of logic grouped by an identifier are applied in sequence and are not available for parallel execution.

iii) **src_ exprType:**

This column refers to a type of matching criteria of the match value in the sourceField. Specifically, it is used in the lookUp of the configuration File. It defines the matching criteria depending on which the match value/expression is captured in the sourceField. This parameter specifies how the program matches the lookup_ value (src_ expression) with values in lookup_ array (sourceField column). It distinguishes between uppercase and lowercase letters when matching text values. Criteria can look very different from each other, depending on the data type of the field to which they apply and specific requirements. The commonly used match criteria are **i) containsToken, ii) contains, iii) not_contains, iv) startsWith, v) endsWith, vi) matches, vii) count**

containsToken: This match class looks up tokens by distinct values. For example, if the lookUp token value is John then it will match with the source value John and not Johnson. Similarly, if the lookUp token value is Editor then it will match with the source value Editor and not Editorial.
Left Tokenizer:
|,|
s|
.|

(Right Tokenizer:
s|,|
.|$|
)

contains: This match class looks up for match token value both in distinct and composite source values. For example, if the match token value is John then it will match with all the source values John, Johnson and PJhonson. Similarly, if the match token value is Editor then it will match with both the source values Editor and not Editorial.

IV) **src_ expression:**

This column specifies the specific match value or expression, also referred to as <token value> for which the matching criteria will be applied to the sourceField to execute requisite transformation in the targetField. It is an expression or value that a program can compare to values in a metadata field in a query. If for a record, the source value fulfills the match criteria, desired transformation takes place in the query results.The match value is a mandatory parameter for moveField function and can be numerical, alphabetical, expressions etc. The matching is case sensitive and textual regardless of the datatype. By case sensitive, a value “AB” mentioned in the configuration file would match with only “AB” in the data and not with any other variation as “ab”, “aB” or “Äb”. By textual it is meant that numerical equality or inequality or specific properties of any other datatypes as date, URI etc., can not be applied to the data. Hence, a field, though characterized as Integer, date or other datatype, exact token from the source value is to be provided in the configuration file for exact matching. The match value allows the identification of certain pattern in a field values that can be further bulk curated.

V) **targetField:**

This column is a part of the transformation process as mentioned in the description of Action. Upon successful matching of the sourcefield - src_expression based on src_exprType, requisite transformation will take place in the targetField. The targetField can be same as the sourceField or a different metadata field.

(vi) tgt_exprType:

This column describes the action to be performed on the match value or the token. Depending on the basis of src_exprType and upon successful matching of the sourcefield-src_expression, desired action will be performed on the match value as specified in this particular column. The resultant target value post transformation will be then assigned to the the targetField. If the matching criteria gives true value, then there is a list commonly used actions performed on the match value **(a) replace, (b) replaceToken (c) regexReplace, (d) remove, (e) move, (f) split**

(a) replace: This keyword allows the match value in the source value of the source field to be replaced by a blank (if tgt_stringValue column is blank) or static value (if tgt_stringValue column holds some static value). Post transformation the target value is then moved to the target field.

(b) replaceToken: This keyword functions in a similar manner as "replace" with the difference that "replace" works on match value irrespective of its distinct presence in the source value but "replaceToken" acts on the discrete token value in the source value. For example, if in dc.subject there are two values say, journal and journalism. If "replace" <match value> as "journal" is applied, it will not only remove the distinct value ’journal’ but also from the ’journalism’ leaving behind and incomplete 'ism'. Comparatively, if "replaceToken" <match value> as "journal" is applied, it will only remove the distinct value 'journal' but not from the 'journalism'. Here also, the token value in the source value of the source field to be replaced by a blank (if tgt_stringValue column is blank) or static value (if tgt_stringValue column holds some static value). Post transformation the target value is then moved to the target field.

(c) regexReplace: Similar to above mentioned replace action, this keyword allows the match value in the source value of the source field to be replaced by a blank (if <tgt_stringValue> column is blank) or static value (if <tgt_stringValue> column holds some static value). However, unlike the above mentioned replace action, regexReplace is used when the match value is a regex expression.

(d) remove: If the source values fulfill the matching criteria and the reserved keyword "remove" is mentioned both in the <targetField> and <tgt_exprType> column, it serves the purpose to delete the matched source values and are not translated to any target field.

(e) move: If the matching criteria is true, this reserved keyword facilitates the shifting of source value from the source field to some other target field (which is not same as source field). The target value can be (i) same as original source value, (ii) composite value, (iii) static value assignment.

(f) split: If the matching criteria is true and the source field is multivalued separated by some delimiter, this reserved keyword facilitates the splitting of the source value by delimiter mentioned in <tgt_expr> column into single valued target field (may or may not be same as the source field).

vii) **tgt_expression:**

This column clearly defines as to how the source value will be expressed in the targetField. Depending on the successful pattern identification, possible transformations could be possible: (i) the source value may undergo some curation and retained in the sourceField itself, (ii) the source value may be shifted to the target field as original value,(iii) the source value may be shifted to the target field as a composite value (original value along with new value). If the target value is subsituted using a reserved keyword **`$Currvalue$`**, that means it is a dynamic value assignment. However, it should not be confused with useMap that involves field translation on the basis of specific sourceValue. Comparatively, in moveField action, tgt_expression depends on the specifications mentioned for the type of token value, matching criteria and the target action to be performed on the token itself to express the source value as desired targetValue in the desired target field.

viii) **stringValue:**

This column also defines as to how the source value will be expressed in the targetField using static value assignment.

For Example:

4.2.7 lookUp

This action searches a metadata field for specified value or expression based on the matching criteria. Depending on the result of matching criteria, targetField is transformed. This action searches one field for specified value or expression based on the matching criteria to bring about some transformation in another field.There are two working Levels of lookUp function: **A)** Collection Level_lookUp function **B)** Handle ID Level_lookUp function

sourceField	match_group	src_exprType	src_expression	targetField	tgt_exprType	tgt_expression	tgt_stringValue
dc.subject.other@ccl		matches	\d{4}-\d{2}-\d{2}	dc.publisher.date	move		
ndl.subject.mesh	1	contains	–	dc.subject	split	–	
dc.coverage.temporal		contains	?	remove	remove		
dc.title		contains	[Untitled]	dc.title	replace	[Untitled]	
dc.title		contains	(untitled)	dc.title	replace	(untitled)	
dc.title		contains	, undated	dc.title	replace	, undated	
dc.title		contains	Untitled and undated;	dc.title	replace	Untitled and undated;	
dc.title		contains	; Untitled; Undated	dc.title	replace	; Untitled; Undated	
dc.title		contains	; Undated; Untitled and unidentified	dc.title	replace	; Undated; Untitled and unidentified	
dc.title		contains	Untitled,	dc.title	replace	Untitled,	
dc.title		contains	(undated)	dc.title	replace	(undated)	
dc.title		contains	(undated)	dc.title	replace	(undated)	
dc.title		contains	undated;	dc.title	replace	undated;	
dc.subject	1	matches	.*	dc.subject	regxreplace	,\s*$	
dc.subject	1	count:=1	,	dc.subject	regxreplace	(.*),(.*)	$2 $1
ndl.subject.hesh	1	contains	–	dc.subject	split	–	

Table 4: moveField

A. Collection Level_lookUp function

The curation bundle for the Collection Level_lookUp function comprises of the following : **1) Command Line syntax, 2) Run Configuration Parameter File, 3) JSON Script logic file 4) Associated Configuration File.**

1) Command Line syntax for Collection Level_lookUp function:
The Collection Level mode is run with the flag –c in the command prompt, i.e. it defines the execution of the lookup function to every content in the collection.

2) Run Configuration Parameter File for Collection Level_lookUp function:

In the curation bundle Run Configuration Parameter file will accept a JSON script as a logic description file with an associated configuration file with specifications of the desired operation.
3) JSON Script logic file for Collection Level_ lookUp function:

While writing a JSON script file for the lookUp function it is important to remember that ***Fall through property is positively True and Copydata command is mandatory***

Generic structure of Collection Level_lookUp_Json_Script

```
{
"Fields": {
"<Field_Name1>": {
"action": [
"lookUp",
"copyData" ],
"lookUp": {
"filter": [ //filter specifications],
"inputFile": "lookUp.xlsx" "delimiter":"Value2"
"action":[//nested action]
} }
}
}
```

Each field for which lookup function has to be applied has its respective Field Translation Block (FTB). Each FTB will have the following components that will be discussed in detail below:i) Key Value, ii) Action Block, iii) Action Descriptor

i) Key Value:

"<Field_Name1>" corresponds to the sourceField for which lookup has to be applied. It is mentioned as key in the FTB and holds the JSON Object Value as the Key-Value pair. This JSON Object Value encloses the curation instructions to be performed on the "<Field_Name1>" in JSON Array Construct. “lookUp” function mentioned in the action array instructs the program to search the "<Field_Name1>" for the matching criteria specified in the associated configuration file as defined in the action descriptor.

ii) Action Block:

"action" corresponds to the JSON Object Value as the Key-Value pair for <Field_Name1>. It is the key for the action container in the FTB. “lookUp” function that has to be applied on the <Field_Name1>"will be enclosed inside a JSON Array Construct named Action as JSON Object Value in Field Translation Block.

iii) **Action Descriptor: (Optional)**

“lookUp” action is described by its Action Descriptor. The Action Descriptor specification for the “lookUp” function looks like the following.

```
   "lookUp": {
"filter": [ //filter specifications],
"inputFile": "lookUp.xlsx"
"delimiter":"Value2"
"action":[//nested action]
}
```

Action descriptor further comprises of some mandatory /optional parameters that are discussed below.

Filter:

It is an action specified by the key “filter”. It is an optional parameter in the action descriptor. If it is mentioned in the Action Descriptor, “lookUp” will apply to only those data that match the filter criteria. If the “filter” action is not mentioned in the Action Descriptor, then the “lookUp” action will be applicable to every item in the collection. A “filter” action in action descriptor has its own specification parameters and the other “lookUp” action parameters.

Parameter Input File (inputFile):

This is the Key to specify the associated configuration file. (The details of the configuration file will be discussed in the subsequent section. 2.5. Specification of the lookUp Configuration File)

Parameter “delimiter”:

It is specified by the Key “delimiter”. It defines the multivalued separator character(s) or delimiter character(s) as a JSON string value in the action descriptor to facilitate multivalued transformation. The delimiter parameter can be a single character or a set of characters as applicable for the transformation. If the multivalued transformations have been mentioned in the configuration file without delimiter specifications in the action descriptor, the transformation would be treated as a single valued transformation.

<u>Nested actions:</u>

It is an optional parameter.It holds any subsequent actions after a successful application of the “add” action. These actions are applied to the transformed data and not on the original field-field value combination since these are nested within one another. Nesting of command can be up to “n” number of levels but Digital Library curator is advised to be cautious enough to keep a control on repetition and circular transformation arising out of the nested action. The whole utility has been designed towards inclusiveness of the system and does not validate the curator’s decision of data management. The list of nested actions is specified by the “action” key and a JSON Array construct as the value holding the action instructions in a sequential order.

4) Specification of the lookUp Configuration File:

LookUp function has an associated configuration file with specifications of the desired operation. The file generally follows the naming convention as “lookUp.xlsx”. In the action descriptor for lookUp function, parameter “inputFile” serves as the key to hold “lookUp.xlsx” config file name as key-value pair as shown below.

```
   "lookUp": {
"inputFile": "lookUp.xlsx"
}
```

It is noted that the configuration file when named lookUp.xlsx and kept at the config location would serve as the default configuration for the lookUp action. Hence, it need not be separately configured in the action descriptor block and consequently if no other action configurations are specified; no action descriptor would be required for such cases (as shown below for the field “dc.description") . However, they can be explicitly mentioned in the action descriptor block for clarity if the digital curator feels so. Separate configuration for each field is not necessary, unless otherwise required. In the below example, a single lookUp.xlxs file has been used for both “dc.title” and “dc. coverage. spatial”

```
   {
"Fields": {
"dc.description": {
"action": [
"lookUp",
"copyData" ],
"lookUp": {
"inputFile": "lookUp.xlsx" } },
"dc.title": {
"action": [
"lookUp",
"copyData" ],
"lookUp": {
"inputFile": "lookUp.xlsx" } },
"dc.coverage.spatial": {
"action": [
"lookUp",
"copyData" ],
"lookUp": {
"inputFile": "lookUp.xlsx" } },
}
}
```

LookUp configuration file consists of six columns namely a) sourceField, b) matchTyp, c) sourceValue, d) targetField, e) targetValue, f) targetValueType.

Generic template of a lookup.xlsx file is shown below:

sourceField	matchTyp	sourceValue	targetField	targetValue	targetValueType

The column order and the naming convention for each column is static. In the following section, the meaning and working principle of each and every configuration column will be explained.

a) **`sourceField`:**

This column specifies the fieldname to which the lookup logic is to be applied. This column lists the fieldname that has to be searched for specific value or expression based on the matching criteria or type of match value. In the JSON logic script, this sourceField is mentioned as a key value of the FTB and the action JSON Array contains a lookup for this particular field for the transformation to take place. This column is a part of the transformation process as mentioned in the description of Action.

b) **`matchTyp`:**

It refers to a type of match value or matching criteria of the value in the sourceField. It specifies the matching criteria depending on which the value/expression is captured in the sourceField. This parameter specifies how the program matches lookup_ value (sourceValue column) with values in lookup_ array (sourceField column). It distinguishes between uppercase and lowercase letters when matching text values. Criteria can look very different from each other, depending on the data type of the field to which they apply and your specific requirements. The commonly used match criteria are discussed in the section below.

i) Starts with:

This criteria searches sourceField mentioned in the first column of the lookUp configuration file (lookUp.xlxs) for the value that starts with or begins with the specified match value defined in the third column.

sourceField	matchTyp	sourceValue	targetField	targetValue	targetValueTy
dc.title	startsWith	Editorial	lrmi.learningResourceType	article	Value

For example, in the above table, the template would allow the tool to verify dc.title of of each content in the dataset. If the tool comes across any content whose value in dc.title field starts with "Editorial", it will fulfill the successful matching criteria and the corresponding targetField lrmi.learningResourceType will be assigned the value "article". It is important to remember, that the it is only important for the sourceValue to begin with the given token "Editorial" and subsequently any alphabet, number, string or expression can follow.

- **Not starts with:**

It is the negation of the matchTyp (starts With). This criteria searches sourceField mentioned in the first column of the lookUp confguration file (lookUp.xlxs) for the value that is does not start with specified match value defined in the third column.

ii) Ends with:

This criteria searches sourceField mentioned in the first column of the lookUp configuration file (lookUp.xlxs) for the value that ends with the specified match value defined in the third column.

sourceField	matchTyp	sourceValue	targetField	targetValue	targetValueTy
dc.title	endsWith	thesis	lrmi.learningResourceType	thesis	Value

For example, in the above table, the template would allow the tool to verify dc.title of of each content in the dataset. If the tool comes across any content whose value in dc.title field ends with "thesis", it will fulfill the successful matching criteria and the corresponding targetField lrmi.learningResourceType will be assigned the value "thesis". It is important to remember, that it is only important for the sourceValue to end with the given token "thesis" and any alphabet, number, string or expression can preceed the token.

- **Not ends with:**

It is the negation of the matchTyp (ends With). This criteria searches sourceField mentioned in the first column of the lookUp confguration file (lookUp.xlxs) for the value that is does not end with specified match value defined in the third column.

iii) **Contains:**

This criteria searches sourceField mentioned in the first column of the lookUp configuration file (lookUp.xlsx) for the value that contains the specified match value defined in the third column. The specified match value can be at any relative position within the existing source value.

sourceField	matchTyp	sourceValue	targetField	targetValue	targetValueType
dc.description	contains	https	dc.format.mimetype	video	Value

For example, in the above table, the template would allow the tool to verify dc.description of of each content in the dataset. If the tool comes across any content whose value in dc.description contains "https", it will fulfill the successful matching criteria and the corresponding targetField dc.format.mimetype will be assigned the value "video". It is important to remember, that it is only required for the sourceValue to contain or hold the given token "video" within the value and any alphabet, number, string or expression can preceed and succeed the token.

iv) **equals:**

This criteria searches sourceField mentioned in the first column of the lookUp configuration file (lookUp.xlxs) for the value that is an exact match or replica of the specified match value defined in the third column.

sourceField	matchTyp	sourceValue	targetField	targetValue	targetValueType
dc.identifier.issn	equals	98346712	dc.language.iso	deu	Value

For example, in the above table, the template would allow the tool to verify dc.identifier.issn of of each content in the dataset. If the tool comes across any content whose value in dc.identifier.issn exactly equals "98346712", it will fulfill the successful matching criteria and the corresponding targetField dc.language.iso will be assigned the value "deu". It is to remember, that sourceValue must exactly hold only the token value "98346712" and no other preceeding or succeeding alphabet, number, string, space or expression. A single extra space or character can result deviation from matching criteria in wrong curation.

not_equals_to:

It is the negation of the matchTyp (equals).This criteria searches sourceField mentioned in the first column of the lookUp confguration file (lookUp.xlxs) for the value that is not an exact match or replica of the specified match value defined in the third column.

v) **matches:**

This criteria searches sourceField mentioned in the first column of the lookUp configuration file (lookUp.xlxs) for the value that is an exact match of the specified expression defined in the third column. It performs character pattern matching to match our criteria and makes the query dynamic. It shows the implementation of pattern matching using regular expressions.

c) **sourceValue:** This column specifies the specific match value or expression for which the matching criteria will be applied to the sourceField to execute requisite transformation in the targetField. It is an expression or value that a program can compare to values in a metadata field in a query. If the value for that field in a given record fulfills the match criteria, desired transformation takes place in the query results.The sourceValue is a mandatory parameter for lookUp function and can be numerical, alphabetical, expressions etc. The matching is case sensitive and textual. By case sensitive, a value “AB” mentioned in the configuration file would match with only “AB” in the data and not with any other variation as “ab”, “aB” or “Äb”. By textual it is meant that numerical equality or inequality or specific properties of any other datatypes as date, URI etc., can not be applied to the data. Hence, a field, though characterized as Integer, date or other datatype, exact sourceValue is to be provided in the configuration file for exact matching.

d) **`targetField`**:

This column is a part of the transformation process as mentioned in the description of Action. Upon successful matching of the sourcefield-sourceValue on the basis of matchTyp, requisite transformation will take place in the targetField. Importantly, the lookup logic is applied on the sourceField but the resultant transformation takes place in the targetField as specified in the targetValue. sourceField remains unchanged.

e) **`targetValue`**:

This column mentions the value to be assigned to the targetField after successful matching of the sourcefield and sourceValue based on matchTyp. If the program can successfully capture the sourceValue as per the defined matching criteria, the resultant transformation takes place in the targetField. This targetValue can be a textual value or an expression.

f) **`targetValueType`**:

It defines the nature of the assignment of value to the targetField. The target value type column is to distinguish the difference between textual value and an expression written in the target value field. As we already know from sec(<target value section>) the target value can be a text or it can also be an expression. When the target value is an expression the target value type is written as expr, for the rest of the value the value type can be left blank or else it can be mentioned as value.

i) **`value`**:

For a record, if the targetValueType is populated as `value`, it means that upon successful matching of the `sourceField-sourceValue` combination based on `matchTyp`, `targetField` will be assigned with the exact textual value mentioned in the `targetValue` column.

ii) **`expr`**:

For a record, if the targetValueType is populated as “expr”, it means that upon successful matching of the sourcefield and sourceValue on the basis of matchTyp, targetField will be assigned with a regex expression.

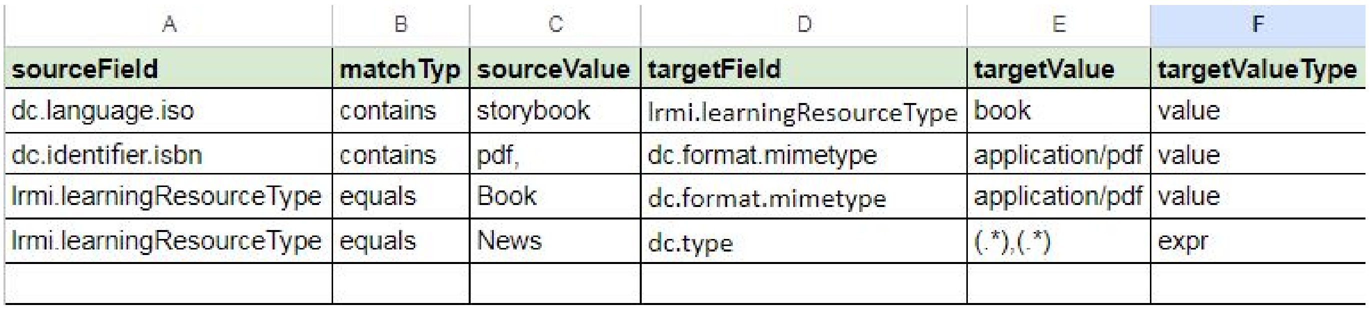
sourceField	matchTyp	sourceValue	targetField	targetValue	targetValueType
dc.identifier.uri	equals	https://epgp.inflibnet.ac.in/epgpdata/uploads/epgp_content/S000438BE/P000725/M018751/ET/1515487623BSE_P6_M35_e-text.pdf	Handle_ID	inflibnet_epgp/business_economics_2980_E_Text	value
dc.identifier.uri	equals	https://epgp.inflibnet.ac.in/epgpdata/uploads/epgp_content/S000438BE/P000725/M018751/LM/1515487633BSE_P6_M35_know_more.pdf	Handle_ID	inflibnet_epgp/business_economics_2980_Learn_More	value
dc.identifier.uri	equals	https://www.youtube.com/watch?v=aOjUTqNazKs	Handle_ID	inflibnet_epgp/business_economics_2980_Self_learning	value
dc.identifier.uri	equals	https://www.youtube.com/watch?v=94NAdF5IYR8	Handle_ID	inflibnet_epgp/economics_13391_Self_learning	value
dc.identifier.uri	equals	https://epgp.inflibnet.ac.in/epgpdata/uploads/epgp_content/S001827/P001855/M030300/ET/15260391699.35_ET.pdf	Handle_ID	inflibnet_epgp/hotel_amp_tourism_management_1855_30300_74427_E_Text	value
dc.identifier.uri	equals	https://epgp.inflibnet.ac.in/epgpdata/uploads/epgp_content/S001827/P001855/M030300/LM/15260392379.35_LM.pdf	Handle_ID	inflibnet_epgp/hotel_amp_tourism_management_1855_30300_74428_Learn_More	value
dc.identifier.uri	equals	https://epgp.inflibnet.ac.in/epgpdata/uploads/epgp_content/S001610/P001799/M025880/ET/1513934516Mod18Q1TrainingandDevelopment.pdf	Handle_ID	inflibnet_epgp/human_resource_management_5669_E_Text	value
dc.identifier.uri	equals	https://epgp.inflibnet.ac.in/epgpdata/uploads/epgp_content/S001610/P001799/M025880/LM/1513934529Mod18Q3.pdf	Handle_ID	inflibnet_epgp/human_resource_management_5669_Learn_More	value
dc.identifier.uri	equals	https://www.youtube.com/watch?v=M5OP5C7m_Lg	Handle_ID	inflibnet_epgp/human_resource_management_5669_Self_learning	value
dc.identifier.uri	equals	https://epgp.inflibnet.ac.in/epgpdata/uploads/epgp_content/S000737SP/P001448/M023360/ET/1506078970P13_M14E-text.pdf	Handle_ID	inflibnet_epgp/spanish_8485_E_Text	value
dc.identifier.uri	equals	https://www.youtube.com/watch?v=jC88h35c7Qw	Handle_ID	inflibnet_epgp/spanish_8485_Self_learning	value
dc.identifier.uri	equals	https://epgp.inflibnet.ac.in/epgpdata/uploads/epgp_content/S000737SP/P001448/M023360/LM/1506078980P13_M14KnowMore.pdf	Handle_ID	inflibnet_epgp/spanish_8485_Learn_More	value

Table 5: `lookUp` Action Configuration File

An example to demonstrate the working of a lookUp.xlsx file

An example of the `lookUp` configuration file (lookUp.xlsx) is shown below. Each property mentioned above would be explained further with an entry from the table below.

`For Example:`



A	B	C	D	E	F
sourceField	**matchTyp**	**sourceValue**	**targetField**	**targetValue**	**targetValueType**
dc.language.iso	contains	storybook	lrmi.learningResourceType	book	value
dc.identifier.isbn	contains	pdf,	dc.format.mimetype	application/pdf	value
lrmi.learningResourceType	equals	Book	dc.format.mimetype	application/pdf	value
lrmi.learningResourceType	equals	News	dc.type	(.*),(.*)	expr

Figure 23: lookUp Configuration File Specification Example

B. **Handle_ID Level lookUp function**

The curation bundle for the Handle_ID Level lookUp function comprises of the following: **1) Command Line syntax, 2) Run Configuration Parameter File, 3) Associated Configuration File**. The Handle_Id Level mode is run with the flag –h in the command prompt, i.e. it defines the execution of the lookup function to a specific HandleID file in the collection and no other content. In the curation bundle Run Configuration Parameter file will accept a CSV (Comma Separated Values) as a logic description file. Handle_ID Level lookUp Function has only one mandatory and important component:

1) Associated HandleID CSV configuration file.Unlike Collection Level, HandleID LookUp configuration file consists of seven columns namely Hid, sourceField, matchTyp, sourceValue, targetField, targetValue, targetValueType.

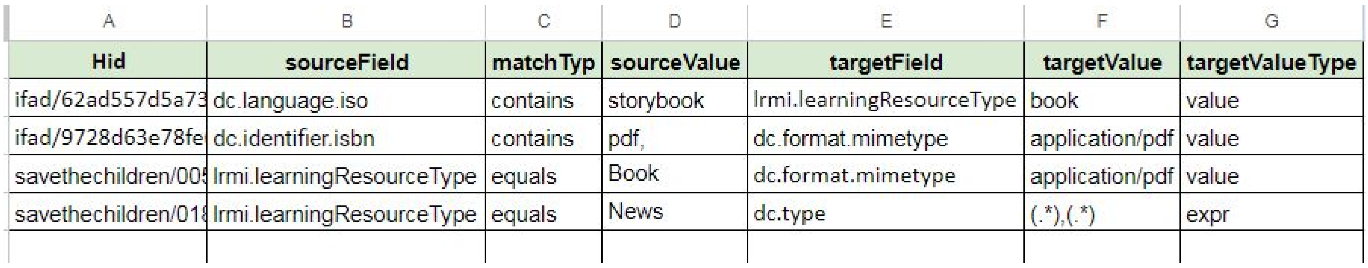
//Generic template of a HandleID lookup.xlsx file is shown below:

Hid	sourceField	matchTyp	sourceValue	targetField	targetValue	targetValueType

The column order and the naming convention for each column is static. In the following section, the meaning and working principle of each and every configuration column is exactly the same as in Collection level except the first Hid column.

Hid:

This column specifies the HandleId identifier for the a particular record or content where the lookup logic is required to be applied.



A	B	C	D	E	F	G
Hid	**sourceField**	**matchTyp**	**sourceValue**	**targetField**	**targetValue**	**targetValueType**
ifad/62ad557d5a73	dc.language.iso	contains	storybook	lrmi.learningResourceType	book	value
ifad/9728d63e78fe	dc.identifier.isbn	contains	pdf,	dc.format.mimetype	application/pdf	value
savethechildren/005	lrmi.learningResourceType	equals	Book	dc.format.mimetype	application/pdf	value
savethechildren/018	lrmi.learningResourceType	equals	News	dc.type	(.*),(.*)	expr

Figure 24: lookUp_ HandleId_ Configuration_ File Specification_ Example

4.2.8 add

ADD command instructs the tool to add new information to the existing data, i.e. it allows addition of a new field with a specified value or addition of new specified value to the already existing field in the data.There are two working Levels of Add command: **i)** Collection Level_Add function **ii)** Handle ID Level_Add function

A. Collection Level_Add function

The curation bundle for the Collection Level_Add function comprises of the following : **1) Command Line syntax, 2) Run Configuration Parameter File, 3) JSON Script logic file**

1) Command Line syntax for Collection Level_Add function:

This action is executed when a new metadata field with specified value(s)is to be added or a new specified value to an existing field is to be added to each item in the collection. The Collection Level mode is run with the flag –c in the command prompt.

2) Run Configuration Parameter File for Collection Level_Add function:

Similar to lookUp function above, in the curation bundle Run Configuration Parameter file will accept a JSON script as a logic description file with an associated configuration file with specifications of the desired operation.

3) JSON Script logic file for Collection Level_Add function:

Fall through property is true and by default inbuilt with add function. Hence, it is not needed to mention "Copydata" command explicitly in the action block for add fuction.

Generic structure of Collection Level_ADD_Json Script

```
   {
"Fields": {
"<Field_Name1>": {
"action": [
"add"
],
"add": {
"filter": [ //filter specifications],
"targetValue":  "Value1"
"delimiter":"Value2"
"action":[//nested action]
}
}
}
}
```

a) **Elaboration on the//Generic structure of Collection Level_ADD_Json Script//**

For each new field that is to be added to the existing data, there will be a respective Field Translation Block (FTB). FTB will specify the “add” action to be implemented on the field. Each field for which lookup function has to be applied has its respective Field Translation Block (FTB). Each FTB will have the following components that will be discuused in detail below:i) Key Value, ii) Action Block, iii) Action Descriptor

i) Key Value:

"<Field_Name1>" corresponds to the targetField that is to be added. It is mentioned as a key in the FTB. It will hold the JSON Object Value as the Key-Value pair. This JSON Object Value encloses the curation instructions to be performed on the "<Field_Name1>" in JSON Array Construct. For each new field that is to be added to the existing data, there will be its respective Field Translation Block (FTB) with targetField to be added as the key value in the FTB.

ii) Action Block:

"action" corresponds to the JSON Object Value as the Key-Value pair for "<Field_ Name1>". It is the key for the action container in the FTB. “add” action that has to be applied on the Key value "<Field_ Name1>" will be enclosed inside a JSON Array Construct named Action as JSON Object Value in Field Translation Block.

iii) Action Descriptor: (Mandatory)

“add” action is described by its Action Descriptor. Action Descriptor is mandatory for the “add” function. If the FTB holds an “add” action container without its Action Descriptor, it will throw an error. The Action Descriptor specification for the “add” action looks like the following.

Action descriptor further comprises of some mandatory /optional parameters that are discussed below.

```
   "add": {
"filter": [
//filter specifications
]
, "targetValue": "Value1"
"delimiter":"Value2"
"action":[
//nested action
]
}
}
}
}
```

Filter:

It is an action specified by the key “filter”. It is an optional parameter in the action descriptor. If it is mentioned in the Action Descriptor, “add” action will apply to only those data that match the filter criteria. If the “filter” action is not mentioned in the Action Descriptor, then the “add” action will be applicable to every item in the collection. A “filter” action in action descriptor has its own specification parameters and the other “add” action parameters.

Parameter Target Value (targetValue):

This is the Key to specify the value that is to be added as the key-value pair to the mentioned targetField "<Field_Name1>" in the FTB. The value of the parameter can be single-valued as well as multivalued which is represented using JSONArray construct.

Parameter “delimiter”:

It is specified by the Key “delimiter”. It defines the multivalued separator character(s) or delimiter character(s) as a JSON string value in the action descriptor to facilitate multivalued transformation. The delimiter parameter can be a single character or a set of characters as applicable for the transformation. If the multivalued transformations have been mentioned in the configuration file without delimiter specifications in the action descriptor, the transformation would be treated as a single-valued transformation.

Nested actions:

It is an optional parameter. It holds any subsequent actions after a successful application of the “add” action. These actions are applied to the transformed data and not to the original field-field value combination since these are nested within one another. Nesting of command can be up to “n” number of levels but Digital Library curator is advised to be cautious enough to keep control on repetition and circular transformation arising out of the nested action. The whole utility has been designed towards the inclusiveness of the system and does not validate the curator’s decision of data management. The list of nested actions is specified by the “action” key and a JSON Array construct as the value holding the action instructions in sequential order.

b) A Sample example to demonstrate the execution of “add” function at Collection Level

Example 1//

S.No.	New Target Field	New Target Value
1	dc.description.searchVisibility	TRUE
2	dc.rights.accessRights	open
3	dc.format.mimetype	application/pdf

Consider Example 1// where the curation requirement is to add the above mentioned three fields with specific targetValues to each item of the collection. Hence, here an entirely new targetField (s) with a specified targetValue (s) are required to be added to the existing data using “add” mode at Collection Level. For a Collection Level requirement, *.run.properties will accept a .json logic file to execute this curation.

A Sample JSON script specification for the above requirement would be as below:

```
   {
"Fields": {
"dc.description.searchVisibility": {
"action": [
"add"
],
"add": {
"targetValue": "true" }
}
, "dc.rights.accessRights": {
"action": [
"add"
]
, "add": {
"targetValue": "open" }
}
, "dc.format.mimetype": {
"action": [
"add"
]
, "add": {
"targetValue": "application/pdf" }
}
, }
}
```

To add the above mentioned fields in the data, there will be three Field Translation Blocks (FTB), one for each new field. Each FTB will have the following components.

Key Value: The targetField(s) that is to be added to each item in the collection will be mentioned as key in their respective FTB. In the table # above, for Row1# 2# and 3#, “dc.description.searchVisibility”, “dc.rights.accessRights”, “dc.format.mimetype” are the target metadata fields required to be added to each item of the collection and are mentioned as the Key in their respective FTBs in the JSON file.

Action Block: It contains an action container with the key as Action with JSON Array Construct. “add” function is enclosed inside the JSON Array Construct and is mentioned as the JSON Object Value in Field Translation Block. Each FTB will hold its respective action container to execute the “add” function to its corresponding targetField in the FTB.

Action Descriptor: Each FTB holds its respective action container to execute the “add” function to its corresponding targetField in the FTB. Each “add” action container is described by its Action Descriptor which specifies the "targetValue" to be added to the targetField.

For example, in the FTB for the targetField "dc.description.searchVisibility", Action descriptor mentions "true" value to the key "targetValue". This will facilitate addition of “true” value to the targetField "dc.description.searchVisibility” in each content of the collection.

Similarly, in the FTB for the targetField "dc.rights.accessRights”, Action descriptor will facilitate addition of “open” value to the targetField "dc.rights.accessRights” in each content of the collection and in the FTB for the targetField " dc.format.mimetype”, Action descriptor will facilitate addition of “application/pdf” value to the targetField in each content of the collection.

As a result, the above JSON script will allow the curation bundle to add the above mentioned three targetFields with their specific targetValues to each item of the collection.

B. Handle_Id Level_Add:

Add function is executed at the Handle_ID level when a new metadata field is to be added or a new specified value to an existing metadata field is to be added to a particular file or Handle_ID in the collection.The curation bundle for the Handle_ID Level_add function comprises of the following : **1) Command Line syntax, 2) Run Configuration Parameter File, 3) Associated Configuration File.** The HandleId Level mode is run with the flag –h in the command prompt, i.e. it defines the execution of the lookup function to a specific Handle_ID file in the collection and no other content. In the curation bundle Run Configuration Parameter file will accept a CSV (Comma Separated Values) as a logic description file. Handle_ID Level_add Function has only one mandatory and important component:*Associated HandleID CSV configuration file*

// Generic and exemplory Specification of Handle_ID_Level_ ADD Csv File//

Hid	targetField	mul_sep	targetValue	mode
savethechildren/0186	lrmi.learningResourceType	;	Journal;article	coalesce
savethechildren/008	dc.language.iso	;	deu;eng	coalesce
savethechildren/0189	dc.publisher.place		United States of America	coalesce
loc/2018270203	dc.description.searchVisibility		True	add
loc/2018270203	lrmi.learningResourceType		newspaper	add
loc/2018270203	dc.rights.accessRights		open	add

Properties of HandleID_Add Csv file

It must be remembered that the HandleID_Add CSV file contains five columns and the Column Order is static i.e. Hid, targetField, mul_sep, targetValue and Mode. About the above table, each column of the Add configuration file will be discussed in detail in the below section.

i) Handle_ID (Hid)

This column mentions the Handle_ID for a particular file or content to which the transformation will take place.
For example in Row1# the curation will take place in the metadata field of content with Handle_ID “savethechildren/0186” only and on no other content in the collection. Similarly, Row2# the curation will take place in the metadata field of content with Handle_ID “savethechildren/008” only and on no other content in the collection.

ii) Target Field (targetField)

This field defines the metadata field name of the particular handle_Id content to which the transformation will take place.
For example, in Row1# the curation will take place in the target metadata field "lrmi.learningResourceType” of content with Handle_ID “savethechildren/0186” only and on no other content in the collection. Similarly, Row2# the curation will take place in the target metadata field" dc.language.iso" of content with Handle_ID “savethechildren/008” only and on no other content in the collection.

Also, if we intend to make more than one transformation in more than one target metadata field for the same Handle_ID, we are required to make a individual entries in each row for each transformation. Example, in Row3# Row4# and Row5# we need to add values to the metadata fields dc.description.searchVisibility, lrmi.learningResourceType and dc.rights.accessRights respectively for the same Handle_ID “loc/2018270203”. Hence, curation requirement for each metadata field will have separate individual entries in different rows of the logic file.

iii) Target Value (targetValue)

This column mentions the value to be assigned to the targetField after transformation to a particular Handle_Id file. This new value can be single valued or multi valued separated by a delimiter.

iv) Multivalue Separator (mul_sep)

This column defines the multivalued separator character(s) or delimiter character(s) to facilitate multivalued transformation. If the multivalued transformations have been mentioned in the configuration file without mul_sep specification, the transformation would be treated as a single valued transformation. For example, in Row1# the requirement is to add two distinct values (Journal; article) to the targetField “lrmi.learningResourceType” of the Handle_Id “savethechildren/0186”. Since column mul_sep clearly mentions (;) to be a multivalue separator, the program will add two individual “lrmi.learningResourceType” fields with values “Journal” and “article” respectively for the Handle_ID “savethechildren/0186”. If the column mul_sep did not mention (;) as a multivalue separator for this entry, then the resultant transformation would be a Handle_ID “savethechildren/0186” file with a single “lrmi.learningResourceType” field a single with value as “Journal; article”. If there is no separator in the targetValue, mul_sep column is left blank but not deleted.

v) Mode(mode)

HandleID_add function performs the addition of new values in three modes.

a) Add

b) Coalesce

c) onBlank

Add:

This mode is default in action i.e. If the data doesnt contain the mentioned targetField, it adds the entirely new required targetField with a specified targetValue to the existing data. Also, if the data already has the mentioned targetField with an existing value, the default add mode refreshes the targetField with the existing value and the newly specified targetValue as add on.

In the above example, Row4# Row5# and Row6# mentions three HandleID files respectively. For each Handle_ID, the correspondingly specified targetFields are newly required to be added to the file with the listed targetValues. This states that the targetFields dc.description.searchVisibility, lrmi.learningResourceType and dc.rights.accessRights did not exist in their corresponding Handle_ID files.

Coalesce:

This mode of addition is in action at Handle_ID level when the targetField mentioned in the logic file already exists in the data with some value(s). Coalesce mode reintroduces the mentioned targetField with targetValue in the specified Handle_ID file which takes the place of the existing data in the Handle_ID file. This transformation is independent of values that are already present for the field in that mentioned Handle_ID.

In the above example, in Row1# the program introduces a two individual targetFields "dc.language.iso" with targetValues "deu" and "eng" in the Handle_ID file savethechildren/008. This addition has occurred in coalesce mode which means the Handle_ID savethechildren/008 already has "dc.language.iso" field with some populated values. However, Coalesce mode will add two individual targetFields "dc.language.iso" with targetValues "deu" and "eng" in the Handle_ID file savethechildren/008 and there will not be any existence of the previous data i.e. "dc.language.iso" field with earlier populated values in the Handle_ID savethechildren/008.

The point to remember is that Coalesce mode should be confused with one to one mapping of values in a field in Handle_ID_useMap. In Handle_ID_useMap,a particular sourceValue is required to be replaced by a targetValue. However, Handle_ID_add with coalesce mode does not depend on the source value. It only introduces a targetField with targetValue that discards the earlier presence of that field in the mentioned Handle_Id.

onBlank:

This mode of addition should not be confused with the default add mode. Though in the default case where a new field is being added both of them might work the same way but when a field is already existing for certain items in the data and a new value needs to be added on condition of absence this particular option is helpful for an inbuilt concise implementation instead of applying default add options with filters.

In a nutshell, the action `add` behaves in the following ways, in the default `add` mode it executes transformations three ways i) when `sourceData` does not have the specified metadata field, it adds on the field for individual items with specified value, ii) if `sourceData` has the specified metadata field with some value for either each item or few , default add will go on to add the newly specified value without disturbing the existing value and existing values would be transformed into the `targetData` based on the further actions defined in the action list for the specified field. Whereas, the `coalesce` mode will reflect some changes only if the data has items holding values. For those items all existing values would be lost and the newly specified value will replace them in the `targetData` even if actions have been put in to transform existing data. For items which did not have any value it would show the new value as they are freshly added. Further, onBlank mode works only where items in the `sourceData` does not have any value. The action would add a new value for all those specific items and the field value for the rest would be transformed as per the actions defined in the action list for the particular field in the curation logic script. One important observation to be noted here is that if a transformation is not mentioned for the existing data, `onBlank` would work only for those items which does not hold any value in the `sourceData` which means in the `targetData` fields which contained some values would now become blank and new value has only been added for those where it was absent and existing values have not been transformed.

4.2.9 deleteField

The action `deleteField` allows the curator to delete or remove a particular metadata field and its value(s) from the data when applied inside the action block. Similar to previous actions, `deleteField` also works at two levels as per the requirement: i) Collection level and ii) Handle ID level.

At the collection level, the action will remove the specified metadata field along with its existing value from every record in the data where exists. This action has a unique feature in that it can not be clubbed with other commands in the action block as the purpose of this action is to delete a field. Since it is logically imperative that once a field is deleted it is not required to perform any further actions onto that field. Hence, the action `deleteField` when applied inside the action block in a `Field Translation Block (FTB)` removes the field from the data item and goes on to perform the actions in the next `FTB`.

```
{
  "Fields": {
    "ndl.sourceMeta.additionalInfo@note": {
      "action": [
        "deleteField"
      ]
    },
    "dc.coverage spatial": {
      "action": [
        "deleteField"
      ]
    },
    "dc.description.sponsorship": {
      "action": [
        "deleteField"
      ]
    }
  }
}
```

Figure 25: An example of a simple `deleteField` action

For Handle ID level implementation the template to run `deleteField` is referred to in Subsection 5.5. It follows the principles of HID curation strategies to apply the logic of field deletion when implemented in a Handle_ID specific situations.

As already known from earlier chapters the Field Translational Block (FTB)>, there is individual FTB for each metadata field to be transformed. For example, in the above instance, the metadata fields to which deleteFunction needs to be applied are "ndl.sourceMeta.additionalInfo@note", "dc.coverage spatial", "dc.description.sponsorship" and each field has its respective FTB. Each FTB has the following properties: **(i) Key Value, (ii) Action Block, (iii) Action Descriptor**

(i) Key Value:
Keyvalue refers to the field name to which the deleteFunction needs to be applied. For example, in the above instance, "ndl.sourceMeta.additionalInfo@note", "dc.coverage spatial", "dc.description.sponsorship" serve as the <key value> for the metadata fields to which deleteFunction needs to be applied.

(ii) Action Block:
Action Block refers to the action container which specifies the action instruction to be applied on the mentioned <Field Name>. It has "action" as the <Key Value> for the action block and the action instruction is enclosed ina Json arry object. For Example, in the above example, Field Translational Block for each query field has the "deleteField" action instruction enclosed in a Json array object for the "action" key.

(ii) Action Descriptor:
It is an optional requirement for the deleteField function unless other wise required as can be seen in the above example. It becomes important to be mentioned if any filter criteria or nested action is to be applied. In such cases the Action Descriptor for "deleteField" looks as shown below.

```
{
  "Fields": {
    "dc.coverage.spatial": {
      "action": [
        "deleteField"
      ],
      "deleteField": {
        "filter": [
          "delete_from_item"
        ],
        "delete_from_item": {
          "fields": [
            "field1"
          ],
          "field1": {
            "fieldName": "dc.relation.haspart",
            "criteria": "exists"
          },
          "expr": "$field1"
        }
      }
    }
  }
}
```

Figure 26: Example of a logic file containing deleteField action

In the above example, the **deleteField** action will delete the metadata field "dc.coverage spatial" only for those items in the data for which the filter criteria is true. So, as discussed earlier, the action descriptor for deleteField function is otherwise optional. Depending on the requirement of the curator it may include the following parameters that have already been discussed in detail in earlier section <Action Descriptor>: **(i) filter, (ii) Nested action, (iii) Delimiter parameter**. When deleteField action is applied on a collection level, there is no associated configuration file.

Handle_ID level "deleteField" method shares similar concept as Collection level curation process with an additional process of generating the logic script by a script generator software. At Handle_ID level, the command prompt, as stated earlier, runs with a -h flag to indicate that the deleteFunction will remove the specified metadata field along with its existing value from **Only specified Handle_ID item** as mentioned in the configuration file. Run Properties Parameter File, as common for any Handle_Id run, accepts a Comma Separated Values (csv) logic file generated by a script generator software.The Comma Separated Values (csv) logic file template shown below is an instance from real data to exemplify working of deleteFunction at Handle_ID level.

Hid	sourceField
savethechildren/0186	lrmi.learningResourceType
savethechildren/008	dc.language.iso
savethechildren/0189	dc.publisher.place
loc/2018270203	dc.description.searchVisibility

The deleteField configuration file has simple specification as shown in the above example. It is a <*.csv> file comprising two columns with static order and namimg convention **(1) Hid, (ii)**

sourceField. In reference to the above example, the program will accurately delete the metadata fields specified under "sourceField" column from their respective Handle_Id files as mentioned under "Hid" column.

4.2.10 attach

The action `attach` attaches assets to the SIP tar file data. This function attaches the specified asset to the target file or displays a link to the file when included in a landing page. An asset is a file that is bundled and deployed with each content and is accessible at runtime. Assets may include full-text files, thumbnails, PDFs, still images, audios, and videos that have been created electronically or have been converted to machine-readable format for preservation or to provide electronic access that is location independent. Common types of assets include static data (for example, JSON files), configuration files, icons, and images (JPEG, WebP, GIF, animated WebP/GIF, PNG, BMP, and WBMP).

ADCT can access the assets through the property `inputFile` of `attach` action inside the curation bundle. The curation Bundle for the attach action comprises of the following

(i) Source data on which assets are need to be attached,

(ii) Run Properties Parameter File defining the ADCT specifications,

(iii) JSON logic File for defining the action for asset attachment and

(iv) Input folder containing the assets.

The asset bundle allows to load a string/text asset or an image/binary asset, given a logical key (attach) is assigned to identify the parent(item) and the child(asset) object. The attach action descriptor maps to the path to the asset specified at build time.

(i) Command Prompt Syntax: The Collection Level mode is run with the flag –c in the command prompt.

(ii) Run Properties Parameter File: As discussed earlier Run Properties Parameter file is a vital component in the curation bundle. Its structure has been explained in detail in the Section 4.1. Run properties file for <attach> action accepts a JSON script file as logic file. A sample Run Properties file for <attach> action is demonstrated below.

```
sourceData=Digital-Library-Ph1-Export-31.12.2021.tar.gz
sourceType=SIP-TAR
targetData=Digital-Library-Ph1-V1.0-10.02.2021.tar.gz
logic=Phase_1_round1_logic_01.02.2022.json
```

Table 6: Collection-curation.run.properties

(iii) JSON script file as logic file for <attach> action: Similar to other ADCT action instructions, JSON script follows similar structural properties. A generic JSON script for <attach> action is shown below.

As can be seen from the above JSON script, the Field Translation Block has the following parameters: (i)Asset Keyvalue,(ii) <attach> Action block, (iii) <attach> Action Descriptor. Each parameter will be elaborated further below.

```
{
  "Fields": {
    "asset.thumbnail": {
      "action": [
        "attach"
      ],
      "attach": {
        "filter": [
          "<filter_name_1>",
          "<filter_name_2>"
        ],
        "inputFile": "<path_to_the_asset_folder>",
        "assetKey": "<asset_identifier>"
      }
    }
  }
}
```

Figure 27: Example logic Script for Asset Attachment

(i) Asset Keyvalue <asset.*>: It is the JSON key that holds the <attach> action block as JSON object. It specifies the type of asset that needs to be attached to the data. The asset type could be (a) thumbnail, (b) fulltext, (c) content file (pdf, word etc.). The <asset.* specifies files that should be included as an asset.

(ii) <attach> Action block: It acts as the JSON object for the asset type mentioned as the key value. The action reserved word <attach> is assigned as object to the key value "action". It instructs the program to attach the specified asset type to the data.

(iii) <attach> Action Descriptor: It describes the mandatory or optional parameters that are applicable to the <attach> action. It includes Filter, inputFile, assetKey, Nested action. The purpose of filter and nested action is optional and functions in a similar manner as discussed earlier. Hence, the two <attach> action specific parameters will be discussed in detail below.

- **inputFile:**

Each asset is identified by an explicit path where the asset file is located.This is a mandatory parameter and is the keyvalue that holds path value where the assets have been placed or located with in the <attach> curation bundle.The order in which the assets are declared in the path doesn’t matter.

- **assetKey:**

This is an optional parameter. It specifies the metadata field name whose value will be used to map the asset to the data. It is important to remember, that if the <assetKey> is not provided then the default mapping of asset to data will be Handle_Id based and hence the each asset should be named accordingly. On the other hand, if <assetKey> is provided then the Handle_Id based default mapping of asset to data will be turned off and each asset will be mapped accoring to the fieldvalue of the the field mentioned in <assetKey>. Therefore, again the asset should be named accordingly.

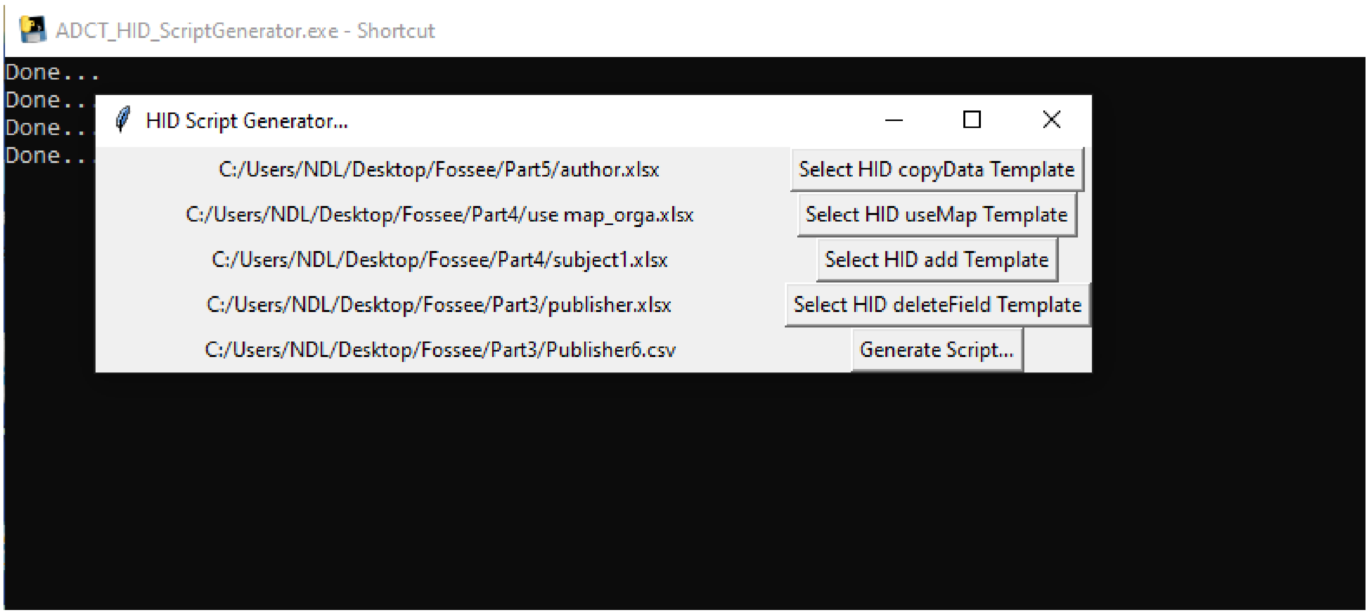
4.2.11 merge

4.2.12 curate

5 Handle_ID Level Curation Methods

Handle_ID level curation method is no different than Collection level curation process with an additional process of generating the logic script by a script generator software.Handle_ID level run comprise of the following components: **(i) the command prompt syntax**, as stated earlier, runs with a -h flag to indicate that the any action instruction will perform the specified action **Only on specified Handle_ID item** as mentioned in the configuration file, **(ii) Run Properties Parameter File** as common for any Handle_Id run, accepts a Comma Separated Values (csv) logic file generated by a script generator software, **(iii) Associated Configuration File**

HID_ScriptGenerator.exe Generally the various HID_configuration files (as discussed for useMap, lookUp, add, deleteField) are prepared as .xls file. These files are later converted to csv files by HID_ScriptGenerator.exe tool. These final csv files serve as the logic files for the Run Properties Parameter File for every Handle_Id run for the action instruction. HID_ScriptGenerator.exe tool looks as shown below.


ADCT_HID_ScriptGenerator.exe - Shortcut
HID Script Generator...
Select HID copyData Template
Select HID useMap Template
Select HID add Template
Select HID deleteField Template
Generate Script...

```
Done...
Done..
Done..
Done..
```

C:/Users/NDL/Desktop/Fossee/Part5/author.xlsx
C:/Users/NDL/Desktop/Fossee/Part4/use map_orga.xlsx
C:/Users/NDL/Desktop/Fossee/Part4/subject1.xlsx
C:/Users/NDL/Desktop/Fossee/Part3/publisher.xlsx
C:/Users/NDL/Desktop/Fossee/Part3/Publisher6.csv

Figure 28: HID Script 1

5.1 HID_useMap

5.2 HID_copyData

5.3 HID_lookUp

5.4 HID_add

5.5 HID_deleteField

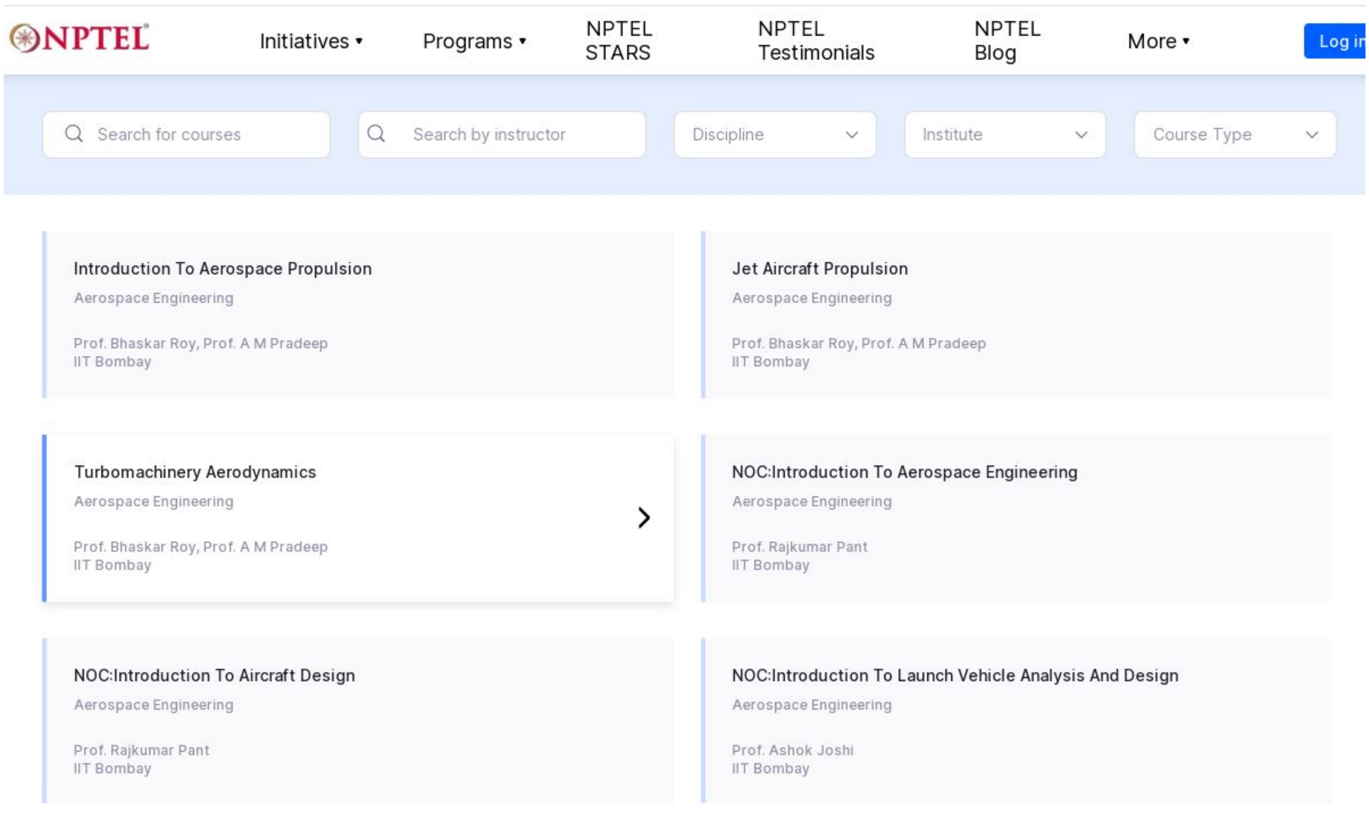
6 Stitching Module

6.1 Introduction to Data Hierarchy

6.1.1 What is hierarchical data?

Data hierarchy refers to the organizational structure of data elements, where data are arranged in a hierarchical order or a tree-like structure, with each level of the hierarchy representing a particular level of abstraction and detail that conforms to a parent-child relationship. In hierarchical data, each of these “children” nodes has only one “parent”, but each parent can have multiple children. In a data hierarchy, the top node level usually represents the most general or abstract information in the hierarchy. The first node, at the top of the hierarchy, is called the root node represented by a level identifier 0. As we move down the hierarchy, we encounter data that become increasingly specific or detailed. The general notion of representing these hierarchial structures is by level identifiers where the root level is represented by the number 0 and the subsequent hierarchies incrementatly. The last and final level of a hierarchy is called a leaf level which is denoted by the level ID *Item*.

Case 1: Figure 29 shows a very basic hierarchical data structure from the source NPTEL (National Programme on Technology Enhanced Learning). NPTEL offers a wide range of online courses in various disciplines that are taught by faculty members from the IITs and IISc, and are available for free to anyone interested in learning. The figure captures a few of such courses that can be considered as the top "parent" nodes.



Introduction To Aerospace Propulsion
Aerospace Engineering
Prof. Bhaskar Roy, Prof. A M Pradeep
IIT Bombay

Jet Aircraft Propulsion
Aerospace Engineering
Prof. Bhaskar Roy, Prof. A M Pradeep
IIT Bombay

Turbomachinery Aerodynamics
Aerospace Engineering
Prof. Bhaskar Roy, Prof. A M Pradeep
IIT Bombay

NOC:Introduction To Aerospace Engineering
Aerospace Engineering
Prof. Rajkumar Pant
IIT Bombay

NOC:Introduction To Aircraft Design
Aerospace Engineering
Prof. Rajkumar Pant
IIT Bombay

NOC:Introduction To Launch Vehicle Analysis And Design
Aerospace Engineering
Prof. Ashok Joshi
IIT Bombay

Figure 29: NPTEL Hierarchical Data

Below Figure 30 represents the course details of the parent course "NOC:Introduction to Aerospace Engineering". The course detail is arranged in a tree-like structure. The course name "NOC:Introduction to Aerospace Engineering" represents the top level, which is further subdivided into "weeks" as the second level and finally "video lectures" as the leaf level child item.

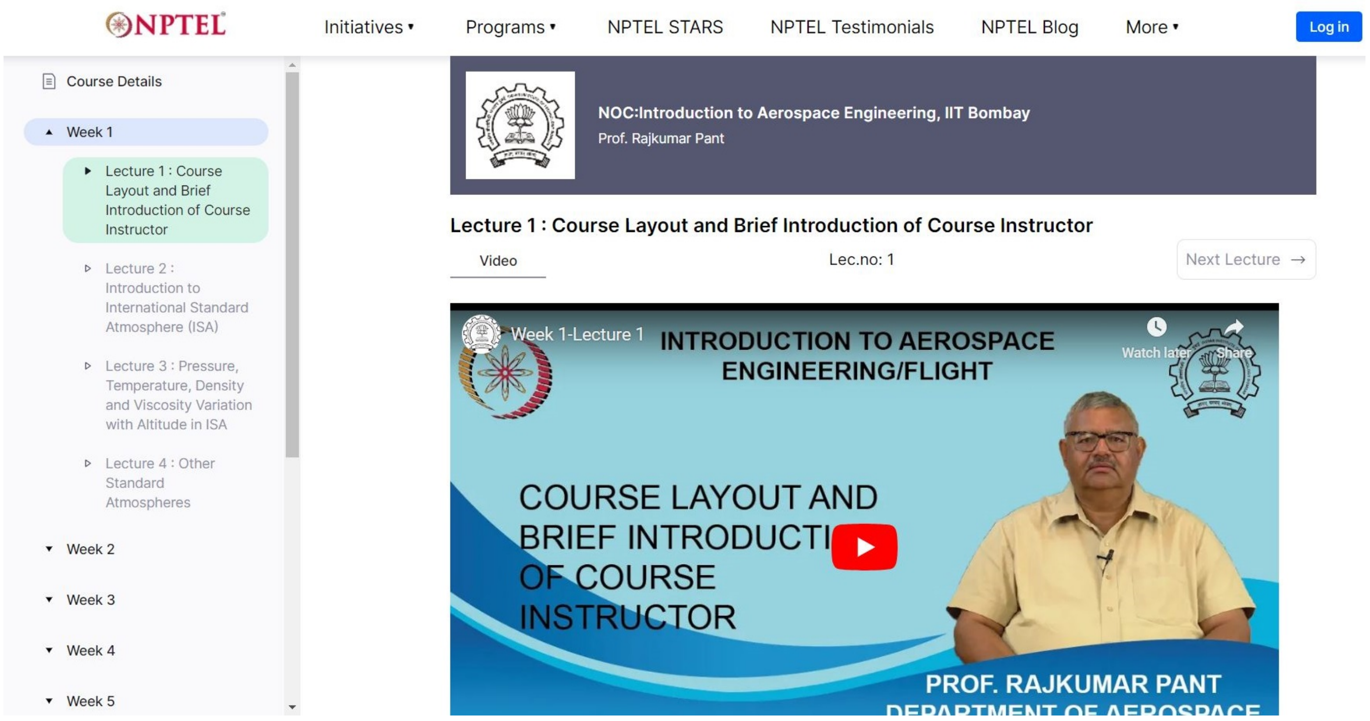
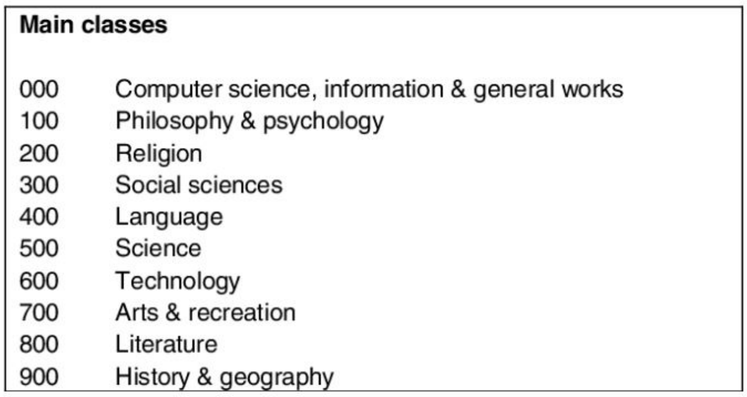

NPTEL
Initiatives ▾
Programs ▾
NPTEL STARS
NPTEL Testimonials
NPTEL Blog
More ▾
Log in
Course Details
Week 1
Lecture 1 : Course Layout and Brief Introduction of Course Instructor
Lecture 2 : Introduction to International Standard Atmosphere (ISA)
Lecture 3 : Pressure, Temperature, Density and Viscosity Variation with Altitude in ISA
Lecture 4 : Other Standard Atmospheres
Week 2
Week 3
Week 4
Week 5
NOC:Introduction to Aerospace Engineering, IIT Bombay
Prof. Rajkumar Pant
Lecture 1 : Course Layout and Brief Introduction of Course Instructor
Video
Lec.no: 1
Next Lecture →
Week 1-Lecture 1
INTRODUCTION TO AEROSPACE ENGINEERING/FLIGHT
Watch later
Share
COURSE LAYOUT AND BRIEF INTRODUCTI OF COURSE INSTRUCTOR
PROF. RAJKUMAR PANT

Figure 30: NPTEL Hierarchical Data

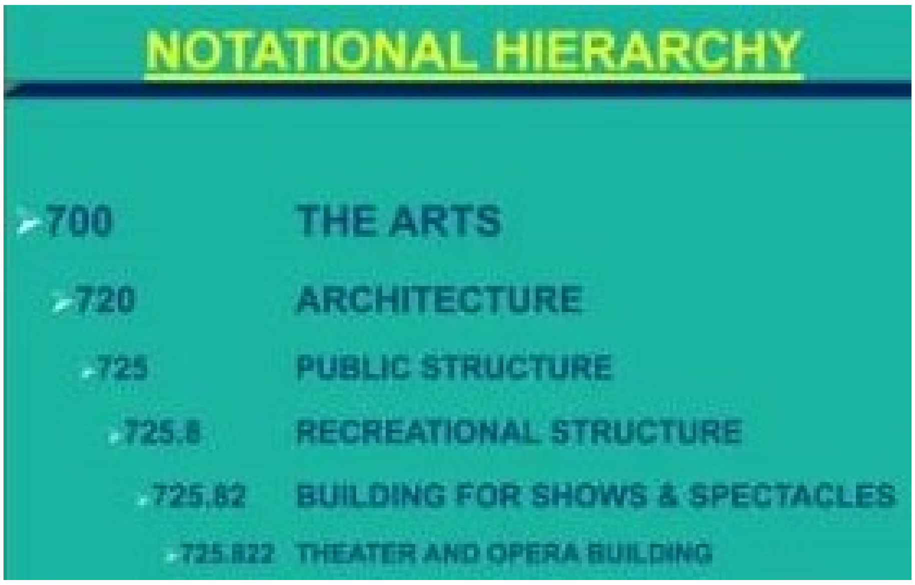
Case 2: A Library classification hierarchy is a simple and typical example of a hierarchical data structure. The classification hierarchy, generally known as Dewey Decimal Classification (DDC), is used to organize and classify books and other materials in a library. The classification is represented in a 3-digit number, in which the top level is represented by a change in the least significant digit of the number, and then subsequent classes are represented by a change in the next significant digit and so on. DDC presents a 3 level hierarchical structure of the data. It starts with the highest level, which is the major subject area, and then moves down through subclasses, sub-subclasses, and so on. The top ten main classes of DDC are shown in Figure 31.


Main classes

000	Computer science, information & general works
100	Philosophy & psychology
200	Religion
300	Social sciences
400	Language
500	Science
600	Technology
700	Arts & recreation
800	Literature
900	History & geography

Figure 31: DDC Main Classes

Here it can be seen that the top-level hierarchy is represented only by a change in the least significant digit of the number. Hence, it can be inferred that the top-level DDC nodes are only restricted to 10 classes. That is why in the previous section, it is already mentioned that DDC represents a simple form of Data Hierarchy, as there are restrictions imposed by the inherent nature of the structure. Below is an example of the DDC hierarchy for the subject area of 700 in Figure 32.



NOTATIONAL HIERARCHY

700	THE ARTS
720	ARCHITECTURE
725	PUBLIC STRUCTURE
725.8	RECREATIONAL STRUCTURE
725.82	BUILDING FOR SHOWS & SPECTACLES
725.822	THEATER AND OPERA BUILDING

Figure 32: DDC Class 700

6.1.2 Why is data structure important?

Data hierarchies represent the relationship between different data elements in a structured way, often in a tree-like structure. Understanding data hierarchies involves identifying the relationships between data elements and the hierarchy they belong to, which is important for organizing and visualizing data in a meaningful way. The ability to input, process, retrieve, and maintain information is essential for any business. This hierarchical structure is often used to organize data in a way that makes it easier to understand, manage, and analyze.

6.1.3 When should an organization use hierarchical data?

Hierarchical data is best used when:

- The data can be stored in a “tree” form with a clear parent and child structure
- There is a need to capture the structure of the hierarchy
- There are high and complex data volume requirements: Data hierarchies can help to organize and manage complex data sets by breaking them down into smaller, more manageable pieces. This can make it easier to find and analyze specific data elements.
- Creating taxonomies: Taxonomies are often used in fields like biology, medicine, and library science to classify and organize data based on shared characteristics or attributes. Data hierarchies can provide a structure for these taxonomies, making it easier to navigate and search for specific data.
- Analyzing data relationships: Data hierarchies can be useful for analyzing relationships between different data elements. By visualizing these relationships in a hierarchy, you can identify patterns and connections that might not be immediately obvious.

6.1.4 Components of Data Hierarchies

Data hierarchies consist of multiple components that are used to organize and classify data in a hierarchical structure. Some of the key components of data hierarchies include:

1. Nodes: Nodes are the building blocks of data hierarchies, representing individual pieces of data. Nodes can contain various types of data, such as text, numbers, or images.
2. Relationships: Relationships are the connections between nodes in a data hierarchy. They define how nodes are related to one another and how they fit together within the larger structure.
3. Parent Nodes: Parent nodes are nodes that contain other nodes within a hierarchy. These nodes are often referred to as "containers" or "branches" and are used to group related data together.
4. Child Nodes: Child nodes are nodes that are contained within parent nodes. These nodes are often referred to as "leaves" and represent individual pieces of data that are part of a larger group.
5. Levels: Levels represent the different tiers within a data hierarchy. Each level is defined by the relationship between parent and child nodes, with the highest level containing the most general data and the lowest level containing the most specific data.
6. Metadata: Metadata is data that provides additional information about the data within a hierarchy. This can include things like dates, descriptions, or tags that help to classify and organize the data.
7. Taxonomies: Taxonomies are systems of classification that are used to organize data within a hierarchy. They are often used to group similar data together based on shared characteristics or attributes.

6.1.5 Advantages of hierarchical data structures

1. Data is easily retrieved: Because the links between data nodes are so well defined, finding and retrieving data is easy. Because the parent and children are stored closely together, navigation and data retrieval is fast too.
2. Referential integrity: The integrity of the data is always maintained because all changes made in the parent table are automatically changed in the child table.
3. Simple structure: The upside-down, parent-child relationship structure is immediately and easily understood. It is conceptually simple, has a clear chain of command within the database, and as a result it offers high performance. Because of the simple structure, it also promotes data sharing.
4. Organization: Data hierarchies provide a structured approach to organizing data, which makes it easier to manage and maintain. By organizing data in a hierarchical structure, it becomes easier to locate and retrieve specific data elements, which can save time and increase efficiency.
5. Analysis: Hierarchical data structures provide a natural way to analyze data at different levels of granularity. By drilling down into the hierarchy, analysts can gain a deeper understanding of the data and identify patterns or trends that may not be apparent at the higher levels.
6. Decision-making: Hierarchical data structures can be used to support decision-making by providing a clear and concise representation of the data. By presenting data in a hierarchical structure, decision-makers can quickly identify key performance indicators, compare different levels of data, and make more informed decisions.

7. Scalability: Hierarchical data structures can be scaled to accommodate large and complex datasets. As data grows, additional levels can be added to the hierarchy to accommodate new data elements, making it easier to manage and analyze data as it grows.

6.2 Hierarchical Data in Digital Library

6.2.1 How can data be related in Digital Library?

In Digital Libraries, often DC needs to handle flat raw data. It could be a set of journal articles, video lecture series, book chapters, educational course material etc. Sometimes, theses contents in a source data may share relationship based on common data lineage (ex. articles published in a particular journal volume), sequential pattern (ex. series of video lectures or chapters in a book) or organizational subject).

6.2.2 What is Content Stitching?

In a Digital Library Data Archival System, the information often comes in a collection of items. Such collection of data items may contain information about an institution or organization. For Large Aggregated Data Archival Systems, data may be spread across multiple subject areas or domains. Sometimes, these data sources or domain specific data collections contain co-related items. Such items when searched in such systems, need to be displayed together so that the relationship across items are maintained. The organization of such related contents within data sources or domain specific collection is called Content Stitching. It is the organization of the related contents in a parent-child relationship in a tree-like structure for a logical and efficient visualization.

6.2.3 Why Content Stitching is needed in a Digital Library?

In a Digital Library, when a DC is assigned a raw dataset, DC can view the curation logic from two aspects as per the requirement: Metadata Level Curation and Hierarchy Level Curation(Content Stitching).
Metadata Level Curation: In this level of curation, DC can simply curate the required metadata fields by following the norms and curation guidelines for the chosen metadata standard. This kind of curation involves the following criteria.

- Annotation of mandatory metadata fields . ex. each content must have title, URL, language, keywords, resource type etc.,
- Ensuring the annotation of values to their expected defined fields. ex. abstract cannot be assigned to a dc.description field and vice-versa. Abstract should be assigned to dc.description.abstract and other descriptions can be assigned to dc.description field.
- Maintaining the correct format of the values in the metadata fields. ex.generally author name follows the format "surname, Firstname", publisher date is written in "yyyy-mm-dd" format, language follows a three-letter code (ex. english as eng),
- Checking for null metadata fields,
- Ensuring if the URL is valid for the contents,
- Checking for redundant values and duplicate contents

DC can perform a flat metadata standard check as mentioned above on the raw dataset, curate and upload the curated data on the website for the users to access. In this case, user can retrieve these contents simply searching by keywords, title or by filter using language, type etc. if necessary metadata are curated correctly. But, the user will not be able to identify the lineage and co-relation within contents (if any common relational data lineage exists) at a glance unless a hierarchical view of the contents is displayed to the user.

Hierarchy Level (Content Stitching): The need to display the lineage of content and its relationship across contents to the user brought about the concept of hierarchical arrangement of contents which in a digital curation world is termed as Content Stitching. Content Stitching is a process of displaying data in a hierarchy based on common data point relations between assets and data objects. More specifically, stitching establishes and showcases relations between the contents with the same data source and holds them together in a technical lineage or hierarchical organization.

6.2.4 Understanding Content Stitching in a Digital Library

Let us understand the concept by a simple example as mentioned below. Consider Table 7 as a raw dataset made available to a DC. The dataset is a collection of fifty journal articles with their respective titles, URLs, contributors and further columns that list out journal name, volume, issue, ISSN where they have been published.

Journal_ Name	Volume	Issue	Title	Contributor	ISSN	URL
Journal1	Vol 1	Issue 1	Article 1: Study of ADCT	Author1	15905896	https://link.JournalArticles.com/article/10.100-0
Journal1	Vol 1	Issue 1	Article 2: Review study on Digital Libraries	Author2	15905896	https://link.JournalArticles.com/article/10.100-1
Journal2	Vol 1	Issue 1	Article 1: Understanding Stitching Module	Author11	15905905	https://link.JournalArticles.com/article/10.100-10
Journal2	Vol 1	Issue 1	Article 2: Understanding Data Lineage	Author12	15905905	https://link.JournalArticles.com/article/10.100-11
Journal2	Vol 1	Issue 2	Article 1: Need for Content Stitching	Author13	15905905	https://link.JournalArticles.com/article/10.100-12
Journal2	Vol 1	Issue 2	Article 2: Trends of Digital Curation	Author14	15905905	https://link.JournalArticles.com/article/10.100-13
Journal2	Vol 1	Issue 2	Article 3: New in Digital Curation	Author15	15905905	https://link.JournalArticles.com/article/10.100-14
Journal2	Vol 2	Issue 1	Article 1: Highlights of Digital Curation	Author16	15905905	https://link.JournalArticles.com/article/10.100-15
Journal2	Vol 2	Issue 1	Article 2: Digital Curation Automation	Author17	15905905	https://link.JournalArticles.com/article/10.100-16
Journal2	Vol 2	Issue 2	Article 1: Content Stitching Module	Author18	15905905	https://link.JournalArticles.com/article/10.100-17
Journal2	Vol 2	Issue 2	Article 2: Composite Content Stitching	Author19	15905905	https://link.JournalArticles.com/article/10.100-18
Journal2	Vol 2	Issue 2	Article 3: Uniform Content Stitching	Author20	15905905	https://link.JournalArticles.com/article/10.100-19
Journal1	Vol 1	Issue 2	Article 1: Comparison of Curation Tools	Author3	15905896	https://link.JournalArticles.com/article/10.100-2
Journal3	Vol 1	Issue 1	Article 1: Parameters of Content Stitching	Author21	15905916	https://link.JournalArticles.com/article/10.100-20
Journal3	Vol 1	Issue 1	Article 2: Content Stitching Benefits	Author22	15905916	https://link.JournalArticles.com/article/10.100-21
Journal3	Vol 1	Issue 2	Article 1: Stitching Template Specifications	Author23	15905916	https://link.JournalArticles.com/article/10.100-22
Journal3	Vol 1	Issue 2	Article 2: Case study on Content Stitching	Author24	15905916	https://link.JournalArticles.com/article/10.100-23
Journal3	Vol 1	Issue 2	Article 3: What is data Hierarchy?	Author25	15905916	https://link.JournalArticles.com/article/10.100-24
Journal3	Vol 2	Issue 1	Article 1: Understanding data Hierarchy	Author26	15905916	https://link.JournalArticles.com/article/10.100-25
Journal3	Vol 2	Issue 1	Article 2: Understanding data Hierarchy and lineage	Author27	15905916	https://link.JournalArticles.com/article/10.100-26
Journal3	Vol 2	Issue 2	Article 1: Concept of real and virtual nodes	Author28	15905916	https://link.JournalArticles.com/article/10.100-27
Journal3	Vol 2	Issue 2	Article 2: Learning Content Stitching	Author29	15905916	https://link.JournalArticles.com/article/10.100-28
Journal3	Vol 2	Issue 2	Article 3: Introduction to Content Stitching	Author30	15905916	https://link.JournalArticles.com/article/10.100-29
Journal1	Vol 1	Issue 2	Article 2: Review study on Curation Tools	Author4	15905896	https://link.JournalArticles.com/article/10.100-3
Journal1	Vol 1	Issue 2	Article 3: Study on Content Stitching	Author5	15905896	https://link.JournalArticles.com/article/10.100-4
Journal1	Vol 2	Issue 2	Article 1: Review Study on Content Stitching	Author6	15905896	https://link.JournalArticles.com/article/10.100-5
Journal1	Vol 2	Issue 2	Article 2: Concept of Content Stitching	Author7	15905896	https://link.JournalArticles.com/article/10.100-6
Journal1	Vol 2	Issue 2	Article 1: Language of ADCT	Author8	15905896	https://link.JournalArticles.com/article/10.100-7
Journal1	Vol 2	Issue 2	Article 2: Data Curation Tools	Author9	15905896	https://link.JournalArticles.com/article/10.100-8
Journal1	Vol 2	Issue 2	Article 3: Stitching Module Concept	Author10	15905896	https://link.JournalArticles.com/article/10.100-9

Table 7: Raw Dataset for Journal Articles from the source system JournalArticles.com

In the first instance, DC may perform the necessary metadata level curation and upload the data for the user. In this case, Content Stitching is not performed and data is flat and unstitched. This data when accessed by the user will be displayed on the interface as shown below in Table 8

Title
Article 1: Study of ADCT
Article 2: Review study on Digital Libraries
Article 1: Understanding Stitching Module
Article 2: Understanding Data Lineage
Article 1: Need for Content Stitching
Article 2: Trends of Digital Curation
Article 3: New in Digital Curation
Article 1: Highlights of Digital Curation
Article 2: Digital Curation Automation
Article 1: Content Stitching Module
Article 2: Composite Content Stitching
Article 3: Uniform Content Stitching
Article 1: Comparison of Curation Tools
Article 1: Parameters of Content Stitching
Article 2: Content Stitching Benefits
Article 1: Stitching Template Specifications
Article 2: Case study on Content Stitching
Article 3: What is data Hierarchy?
Article 1: Understanding data Hierarchy
Article 2: Understanding data Hierarchy and lineage
Article 1: Concept of real and virtual nodes
Article 2: Learning Content Stitching
Article 3: Introduction to Content Stitching
Article 2: Review study on Curation Tools
Article 3: Study on Content Stitching
Article 1: Review Study on Content Stitching
Article 2: Concept of Content Stitching
Article 1: Language of ADCT
Article 2: Data Curation Tools
Article 3: Stitching Module Concept

Table 8: Output of Unstitched data

In the second instance, DC after performing necessary metadata curation observes that the contents share lineage and relation. The sample raw dataset has articles that belong to three parent journals – *"Journal1", "Journal2" and "Journal3"*. Each parent journal is further has its respective subdivisions at volume and issue level. Here DC might wisely perform the Content Stitching and upload stitched data for the user to access. This Stitched data will be displayed on the interface as shown below in Table 9

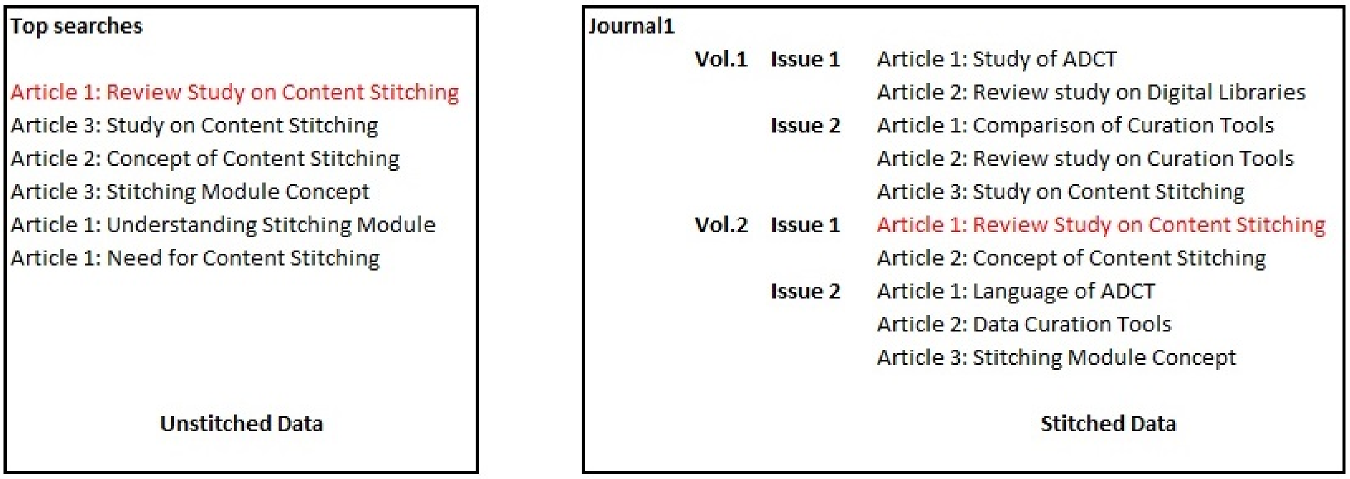
Level 1	Level 2	Level 3	Level 4
Journal 1			
	Vol 1	Issue 1	Article 1: Study of ADCT
			Article 2: Review study on Digital Libraries
		Issue 2	Article 1: Comparison of Curation Tools
			Article 2: Review study on Curation Tools
			Article 3: Study on Content Stitching
	Vol. 2	Issue 1	Article 1: Review Study on Content Stitching
			Article 2: Concept of Content Stitching
		Issue 2	Article 1: Language of ADCT
			Article 2: Data Curation Tools
			Article 3: Stitching Module Concept
Journal 2			
	Vol. 1	Issue 1	Article 1: Understanding Stitching Module
			Article 2: Understanding Data Lineage
		Issue 2	Article 1: Need for Content Stitching
			Article 2: Trends of Digital Curation
			Article 3: New in Digital Curation
	Vol. 2	Issue 1	Article 1: Highlights of Digital Curation
			Article 2: Digital Curation Automation
		Issue 2	Article 1: Content Stitching Module
			Article 2: Composite Content Stitching
			Article 3: Uniform Content Stitching
Journal 3			
	Vol. 1	Issue 1	Article 1: Parameters of Content Stitching
			Article 2: Content Stitching Benefits
		Issue 2	Article 1: Stitching Template Specifications
			Article 2: Case study on Content Stitching
			Article 3: What is data Hierarchy?
	Vol. 2	Issue 1	Article 1: Understanding data Hierarchy
			Article 2: Understanding data Hierarchy and lineage
		Issue 2	Article 1: Concept of real and virtual nodes
			Article 2: Learning Content Stitching
			Article 3: Introduction to Content Stitching

Table 9: Output for Stitched data

Comparison

Now, let us compare the two instances. Assume that a user needs to retrieve a journal article titled *"Article 1: Review Study on Content Stitching"* from the *"Journal1 : Vol2 : Issue1"*.

In the first instance, the user will give a search query and may get the desired article in the top 5-10 searches, as shown below in Figure 15(left side output). In this case user may only access the searched article, but will remain be ignored of its preceding or succeeding related articles in that issue and its lineage in the data. But, in the second instance, if the articles belonging to a Journal are arranged in a hierarchy (say, volume and issue), user can not only have a an easy and efficient data retrieval but also have broad view about its lineage in the hierarchy and will be guided to easily navigate through its preceding or succeeding articles, volumes and issues. The user can have a complete visualization of the source data structure by simply searching for a desired content. The comparison for the two instances has been demonstrated below in Figure 33.



Top searches

Article 1: Review Study on Content Stitching
Article 3: Study on Content Stitching
Article 2: Concept of Content Stitching
Article 3: Stitching Module Concept
Article 1: Understanding Stitching Module
Article 1: Need for Content Stitching

Unstitched Data

Journal1

Vol.1	Issue 1	Article 1: Study of ADCT
		Article 2: Review study on Digital Libraries
	Issue 2	Article 1: Comparison of Curation Tools
		Article 2: Review study on Curation Tools
		Article 3: Study on Content Stitching
Vol.2	Issue 1	Article 1: Review Study on Content Stitching
		Article 2: Concept of Content Stitching
	Issue 2	Article 1: Language of ADCT
		Article 2: Data Curation Tools
		Article 3: Stitching Module Concept

Stitched Data

Figure 33: Comparison of Stitched and Unstitched Data

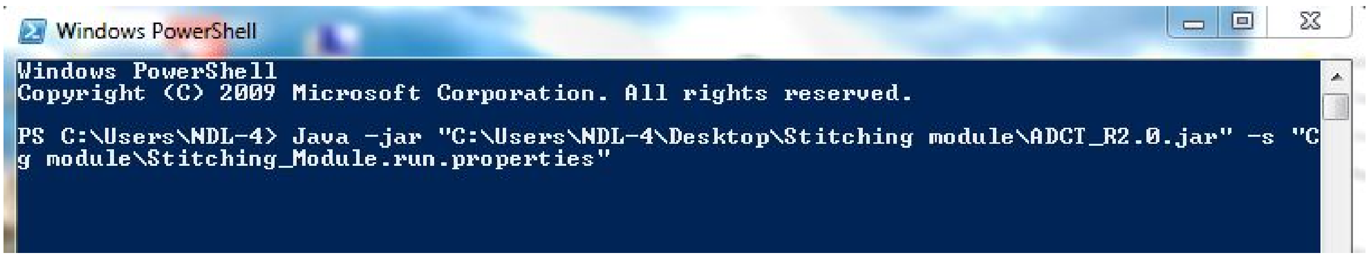
6.3 Key Components of Stitching Module in ADCT:

The idea of Stitching Level Curation has been briefly touched earlier in Section 3, however the prime components and prerequisites to run a Stitching module will be briefly discussed in detail in this section. The Curation Bundle for Stitching module includes the following components, and each component is briefly explained in the later subsections.

1. Command Line syntax for Stitching
2. Run Configuration Parameter File for Stitching
3. Stitching Template configuration
4. Source Data

Command Line syntax for Content Stitching:

The stitching module is run with the execution flag **–s** in the command prompt, as shown in Figure 34. The execution flag **–s** has been identified after the name name of the module, that is content Stitching.



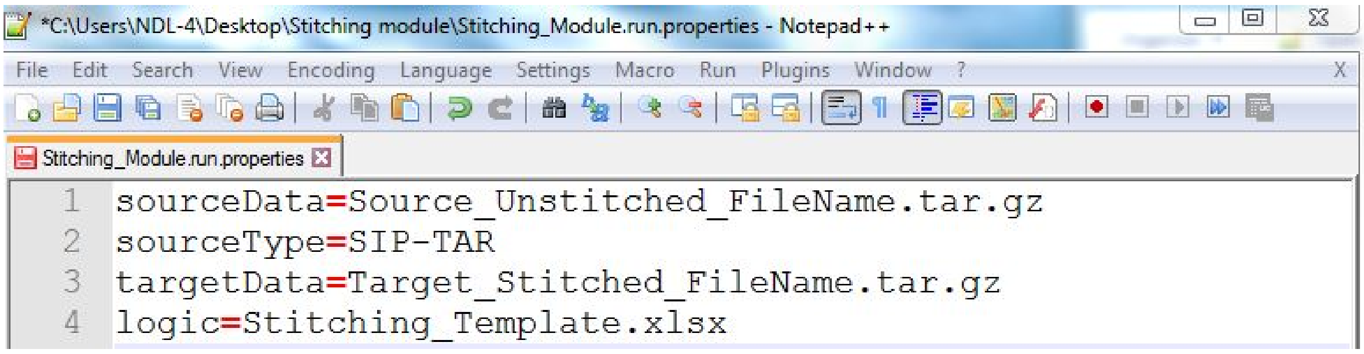
```
Windows PowerShell
Copyright (C) 2009 Microsoft Corporation. All rights reserved.

PS C:\Users\NDL-4> Java -jar "C:\Users\NDL-4\Desktop\Stitching module\ADCT_R2.0.jar" -s "C
g module\Stitching_Module.run.properties"
```

Figure 34: Command Prompt Syntax for Stitching

Run Configuration Parameter File for Stitching:

The curation bundle for stitching requires a Run Configuration Parameter file **(*.run.properties file)** that will accept a **Microsoft Excel (.xlsx)** file as a logic file parameter with specifications of the desired operation Figure 35. This .xlsx file is called the Stitching Curation Template and has its specific configuration. Hence the logic file parameter in the *.run.properties file becomes an Excel spreadsheet file pointing to the stitching template in the Curation Bundle.

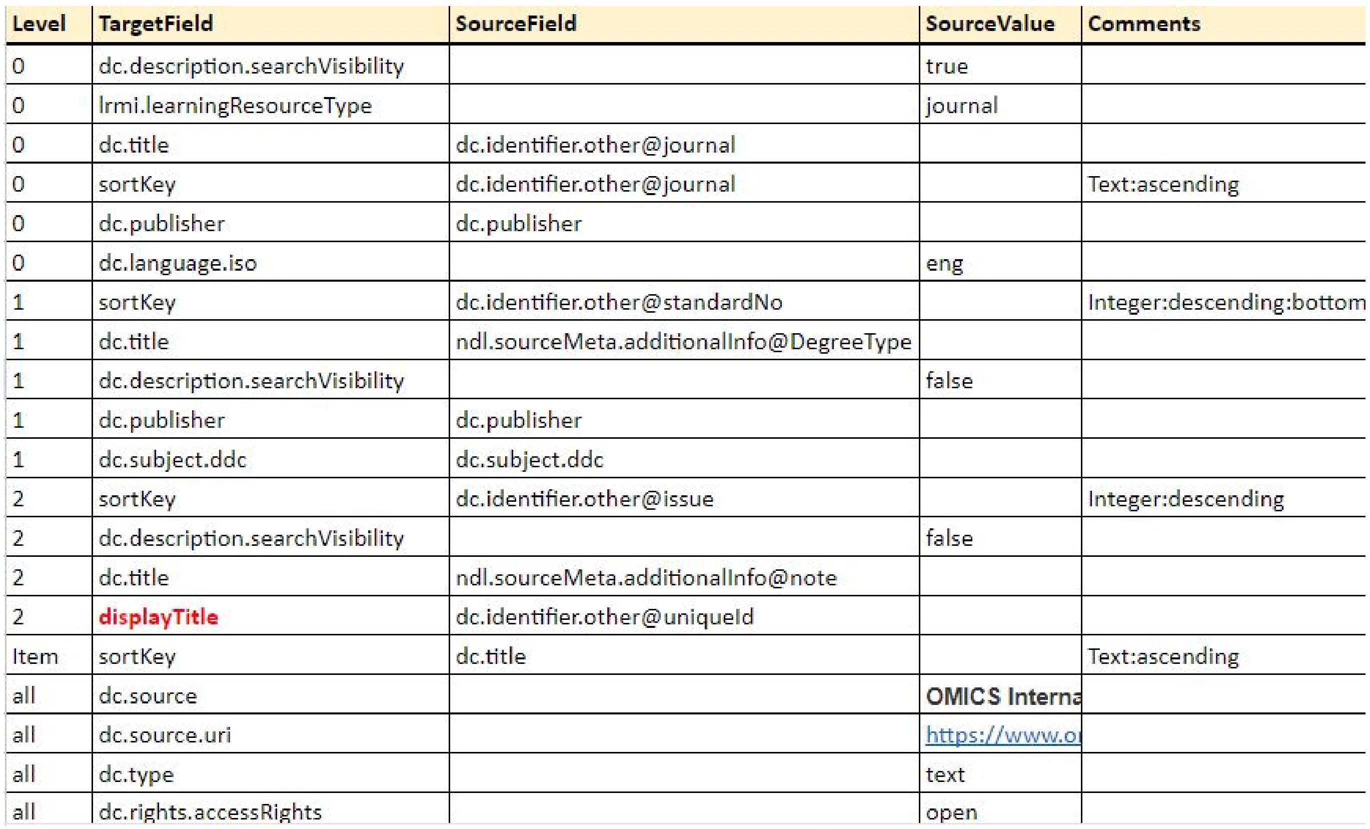

*C:\Users\NDL-4\Desktop\Stitching module\Stitching_Module.run.properties - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
Stitching_Module.run.properties

```
sourceData=Source_Unstitched_FileName.tar.gz
sourceType=SIP-TAR
targetData=Target_Stitched_FileName.tar.gz
logic=Stitching_Template.xlsx
```

Figure 35: Run Properties for Stitching

6.4 Stitching Curation Template configuration:

As discussed earlier in Section 3.3 above The Stitching Curation template is a spreadsheet file with a file extension **MS-EXCEL(.xlsx)**. It includes the mapping between the hierarchical data of the data collection and the different metadata fields of the data items. The fact that metadata values in Data Stitching Curation items remain unchanged—that is, no metadata modifications occur—is a crucial point to remember. This procedure is unique in that it establishes the connections between the objects and how they relate to one another. A newly formed item known as the Parent Item, or, in certain rare circumstances, an item from the current collection, is used to identify each hierarchical level. The parent item is logical and hence is physically represented by an Item ID which in the context of Digital Library is the Handle_ID.
The Stitching template consists of five columns that have static naming convention and column order. They are ***(i) Level, (ii) TargetField, (iii) SourceField, (iv) SourceValue, (v) Comments***. An exemplary stitching template has been shown below in Figure 36 and Table 10 and each parameter of the Stitching template will be discussed in detail further below.



Level	TargetField	SourceField	SourceValue	Comments
0	dc.description.searchVisibility		true	
0	lrmi.learningResourceType		journal	
0	dc.title	dc.identifier.other@journal		
0	sortKey	dc.identifier.other@journal		Text:ascending
0	dc.publisher	dc.publisher		
0	dc.language.iso		eng	
1	sortKey	dc.identifier.other@standardNo		Integer:descending:bottom
1	dc.title	ndl.sourceMeta.additionalInfo@DegreeType		
1	dc.description.searchVisibility		false	
1	dc.publisher	dc.publisher		
1	dc.subject.ddc	dc.subject.ddc		
2	sortKey	dc.identifier.other@issue		Integer:descending
2	dc.description.searchVisibility		false	
2	dc.title	ndl.sourceMeta.additionalInfo@note		
2	displayTitle	dc.identifier.other@uniqueId		
Item	sortKey	dc.title		Text:ascending
all	dc.source		OMICS Interna	
all	dc.source.uri		https://www.o	
all	dc.type		text	
all	dc.rights.accessRights		open	

Figure 36: Stitching Template Example from real data

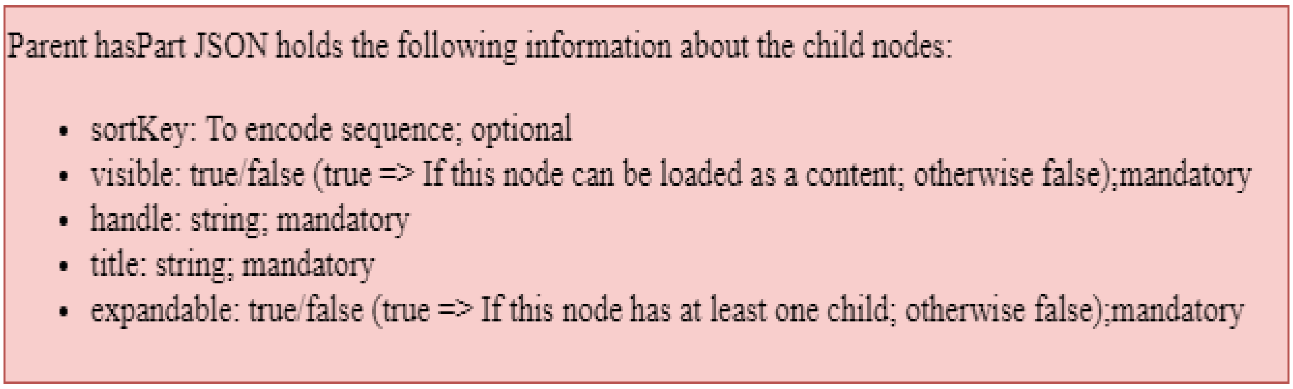
Level	TargetField	SourceField	SourceValue	Comments
0	dc.title	Title_ Meta1.addInfo		
1	dc.title	Title_ Meta2.addInfo		
1	sortKey	Sort_ Meta1.addInfo		Integer:descending:bottom
2	dc.title	Title_ Meta3.addInfo		
2	sortKey	Sort_ Meta2.addInfo		date:ascending:bottom
3	dc.title	Title_ Meta4.addInfo		
3	sortKey	Sort_ Meta3.addInfo		date:ascending:bottom
4	dc.title	Title_ Meta5.addInfo		
4	sortKey	Sort_ Meta4.addInfo		Real:descending:bottom
Item	displayTitle	Title_ Meta6.addInfo		
Item	sortKey	Sort_ Meta5.addInfo		Integer:ascending:bottom
all	dc.source		Source_ Data_ Name	
all	dc.source.uri		Source_ Data_ URL	
all	dc.language.iso		eng	

Table 10: Stitching Template Specification

6.4.1 Level

Usage: Mandatory
The column Level in the Stitching Template maps the level hierarchy information within the collection. The key features of this component are listed below.

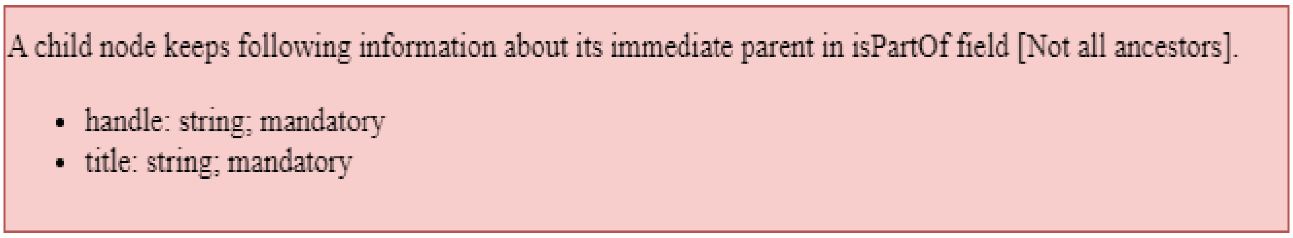
- When contents are stitched together into a single lineage, a specific hierarchical pattern is followed. The first column in the Stitching Template refers to the different levels or nodes in a hierarchy. The name of the column is "Level" which has the following attributes.
- Each hierarchical level is denoted by an Integer number (0, 1, 2, 3, .., item). "0" denotes the topmost level which is also known as Top Parent Level or Root Level. "item" denotes the main content and is also known as Leaf level. "1, 2, 3..." denote intermediate levels such as (1st, 2nd, 3rd...levels). If the there is a value "all" in this column that applies to all the parent nodes except the Leaf level.
- The number of levels or hierarchical depth is theoretically infinite but is data and system dependent.
- Except of Leaf level "item", all other nodes can be real or virtual i.e. virtual nodes will not have the value for loadable content URL whereas real nodes will have the value loadable content URL. The Leaf level mandatorily be a real node and should have value with a loadable content.
- Two consecutive nodes hold a Parent-Child relationship. The DC fields *dc.relation.haspart* and *dc.relation.ispartof* is used to encode hierarchical data organization in resource meta-data. Each preceding level is the Parent item for its immediate succeeding level. A parent node keeps information about its immediate child nodes but not all the successors. Parent *dc.relation.ispartof* of a root element does not exist. The Parent *dc.relation.ispartof* of any other node is the nearest available ancestor node.


Parent hasPart JSON holds the following information about the child nodes:

- sortKey: To encode sequence; optional
- visible: true/false (true => If this node can be loaded as a content; otherwise false);mandatory
- handle: string; mandatory
- title: string; mandatory
- expandable: true/false (true => If this node has at least one child; otherwise false);mandatory

Figure 37: Components of Parent hasPart

- Also, each succeeding level is the Child item for its immediate predecessor or ancestor node. Child of (hasPartOf) a leaf element does not exist. Child (hasPartOf) of any other node is the nearest visible descendant or successor item.
- Every node (whether virtual or not) is a physical item and has to be assigned a handle-id. Item that corresponds to a virtual node will have its searchVisibility metadata value to be false. Stitching hierarchy though goes down from Root level to Leaf level but stitching information and parameters are executed from Leaf to Root level.


A child node keeps following information about its immediate parent in isPartOf field [Not all ancestors].

- handle: string; mandatory
- title: string; mandatory

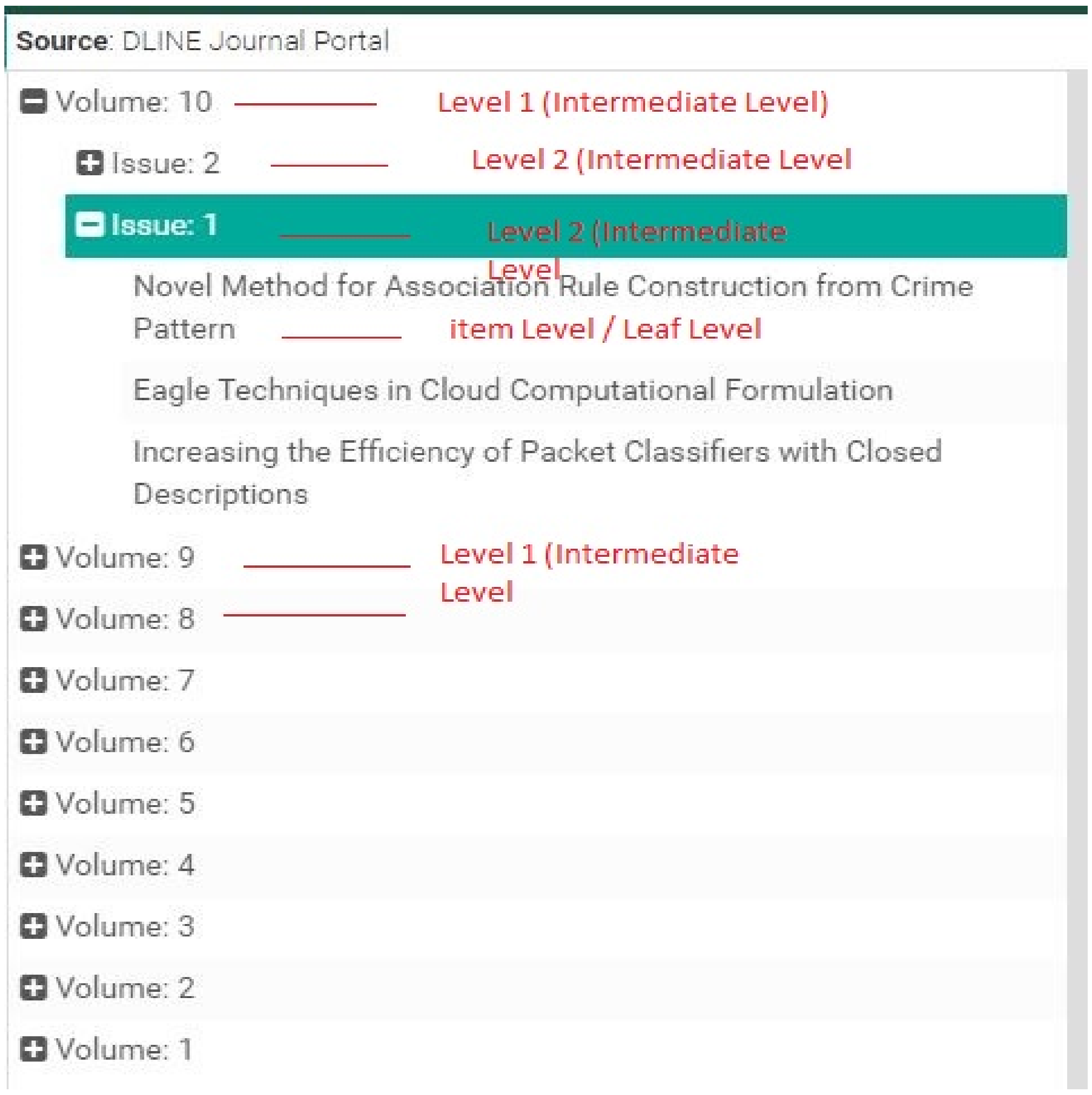
Figure 38: Components of Child isPartOf

Stitching Level	Description of Level	Virtual or Real
0	Top Parent/Root Level	Virtual or Real
1	1st Intermediate Level	Virtual or Real
2	2nd Intermediate Level	Virtual or Real
3	3rd Intermediate Level	Virtual or Real
..	Intermediate Level	Virtual or Real
..	Intermediate Level	Virtual or Real
item	Leaf Level	Real
all	all the nodes from Root level to the Leaf level	Virtual or Real

This can be further understood by some examples as follows.

1. **Journal article content stitching**: If the digital curator (DC) has a set of journal articles and intends to organise it digitally in a way that is feasible for the user, DC will need to organise and stitch together the articles in a specific pattern. The most common hierarchical pattern followed is shown below.In Figure 39 shows four levels of stitching that belong to a common data source (journal). The journal name is the Root Level and the actual content article represents the Leaf level. Volume and issue levels represent the first and second intermediate levels. Hence, the leaf level represents the journal articles of vol. 10, issue 1 of a certain journal.

2. **Book chapters content stitching**: Similarly, incase of books, if its contents are available as individual items (front cover, Table of contents, Preface, Units, chapters, appendix, answers etc.), the stitching pattern may be follows. Figure 40 represents five level of hierarchical organization. The highlighted content represents a chapter from the english book "An Alien Hand" of Class 7.



Source: DLINE Journal Portal

- Volume: 10
  - Issue: 2
  - Issue: 1
    - Novel Method for Association Rule Construction from Crime Pattern
    - Eagle Techniques in Cloud Computational Formulation
    - Increasing the Efficiency of Packet Classifiers with Closed Descriptions
- Volume: 9
- Volume: 8
- Volume: 7
- Volume: 6
- Volume: 5
- Volume: 4
- Volume: 3
- Volume: 2
- Volume: 1

Level 1 (Intermediate Level)
Level 2 (Intermediate Level
Level 2 (Intermediate Level
item Level / Leaf Level
Level 1 (Intermediate Level

Figure 39: Journal stitching Pattern

3. **Newspaper content stitching**: If the data consists of daily newspapers of different months and years of different newspapers, where each page of a newspaper represents an item, stitching pattern can be as follows. Figure 41 shows a five level common data lineage. Page 1, 2....8 represent the pages of a certain parent newspaper dated 1 Jan 1920, Edition 1.

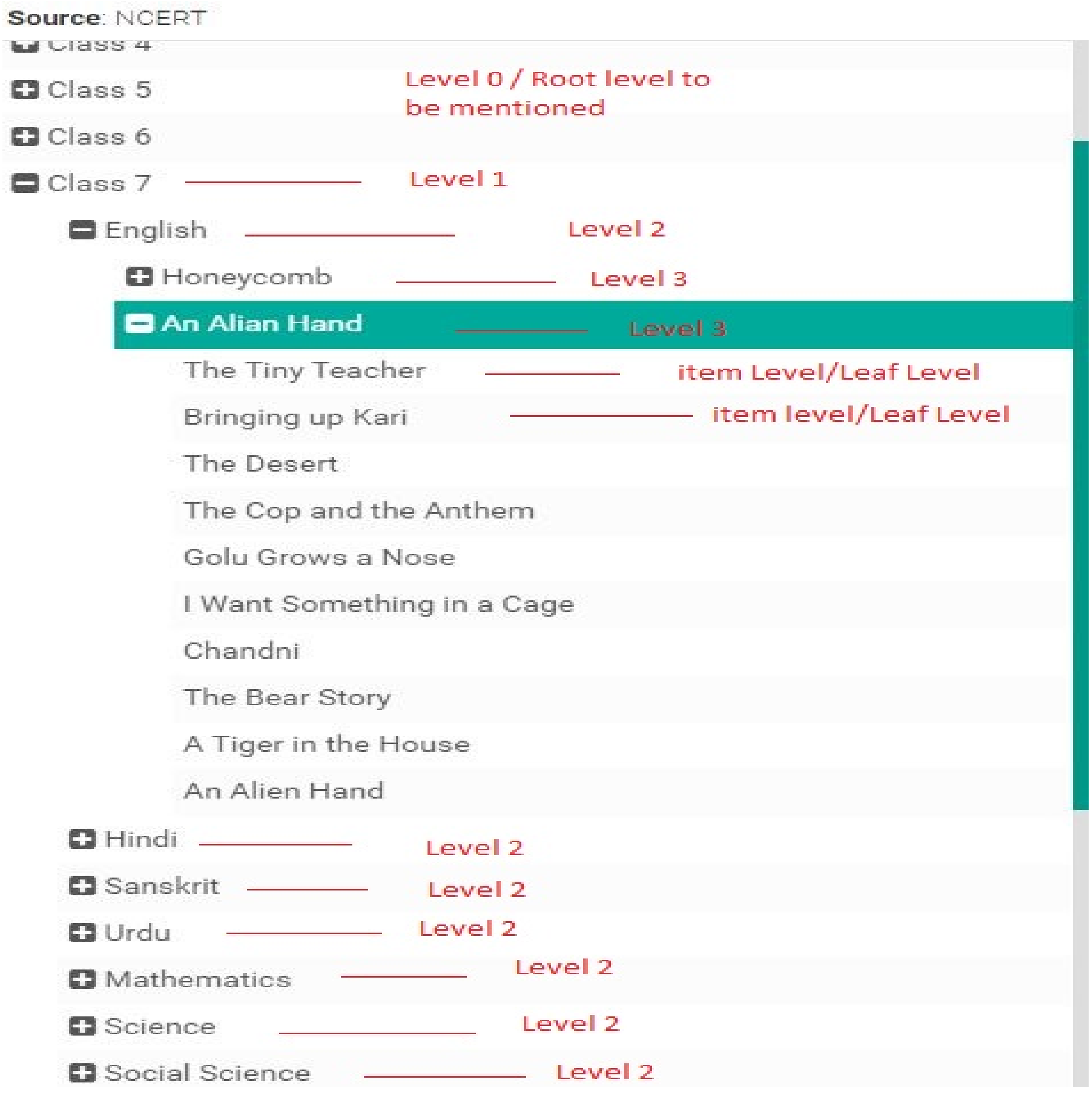
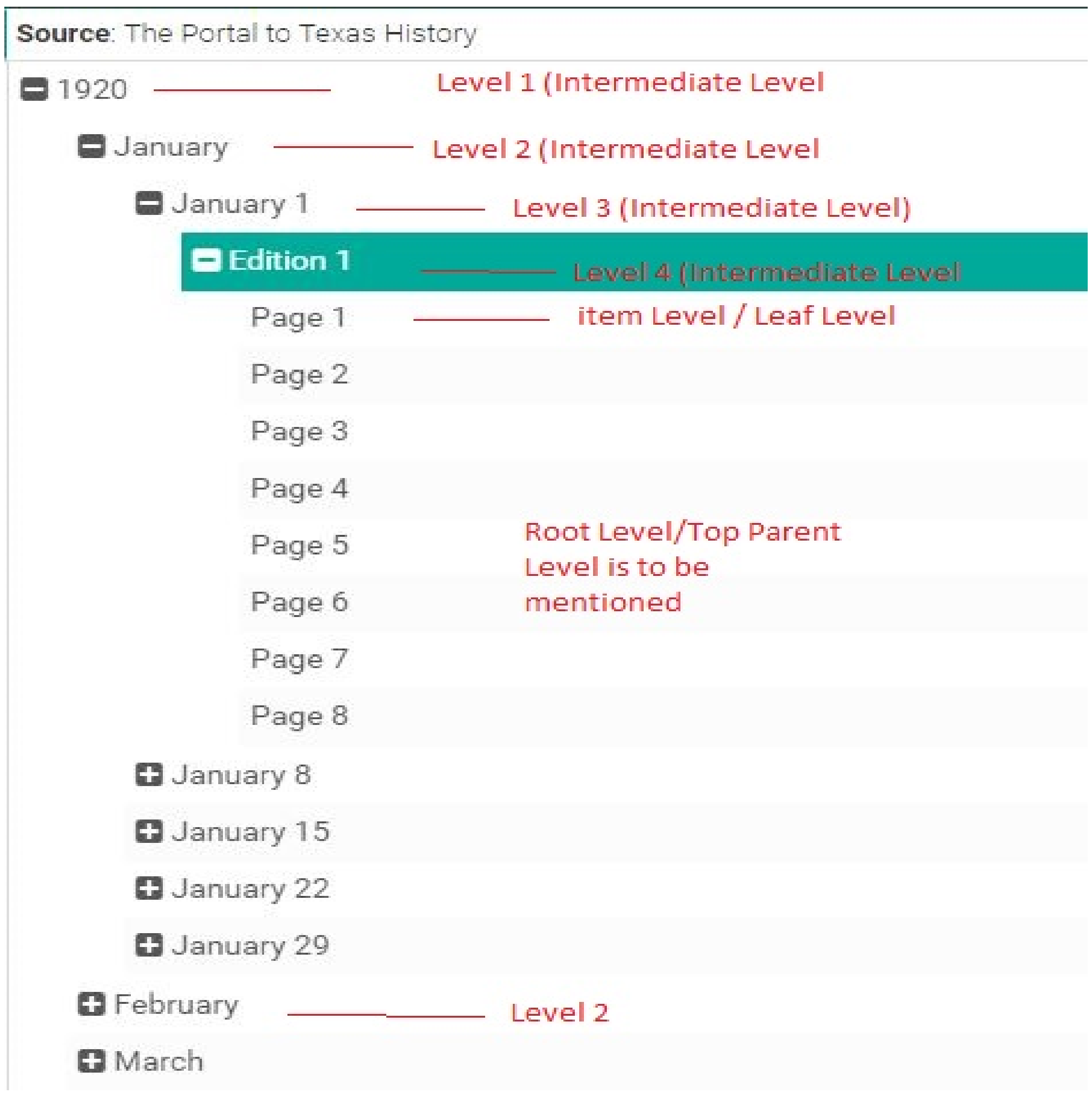


Figure 40: Book stitching Pattern

The whole idea to state the above examples is to demonstrate that there is no predetermined stitching design for each resource type and each resource type can be stitched together differently. The stitching design and hierarchical depth depend on the type of resource, availability of sufficient metadata that may be important parameters for stitching, source discretion, user's perspective which is how the user demands to look at a particular content and also DC choice as to how the content is chosen to be displayed at the interface.

6.4.2 TargetField

Usage: Mandatory
The column TargetField holds the information about he metadata fields that are to be created at the Parent levels to define the hierarchical structure. It allows mapping of newly created metadata fields at the Parent level to the metadata fields (dynamic mapping) or values (static mapping) from the originating data items. In most circumstances, the title indicates the level mapping information; however, in situations when physical items are involved, the item ids indicate the hierarchy information. Once more, Handle ID serves as the primary representation of the item id in the context of the Digital Library Data Curation domain.TargetField column also defines three


Source: The Portal to Texas History

- 1920
  - January
    - January 1
      - Edition 1
        - Page 1
        - Page 2
        - Page 3
        - Page 4
        - Page 5
        - Page 6
        - Page 7
        - Page 8
    - January 8
    - January 15
    - January 22
    - January 29
  - February
  - March

Level 1 (Intermediate Level
Level 2 (Intermediate Level
Level 3 (Intermediate Level)
Level 4 (Intermediate Level
item Level / Leaf Level
Root Level/Top Parent Level is to be mentioned
Level 2

Figure 41: Newspaper stitching Pattern

types of elements that need to be remembered: Mandatory or Optional Metadata Fields, sortKey, displayTitle. Each elememt will be briefly discussed below.

- **(Mandatory/Optional Metadata Fields)** This column lists out the metadata fields DC desires to be populated for the respective levels. However, these metadata fields can be mandatory or optional. For example, it is mandatory for every node except for the <item> level to have a "dc.title" field whose value will be derived from the corresponding metadata field name mentioned in the column "SourceField". However, <item> node will not have this "dc.title" field mentioned in the stitching template as it is the actual content that already has the title and need not to be derived. Similarily, every node from Level 1.....item Level is mandate to have "sortKey" except for the 0 Level. Further, some other necessary metadata fields may be dc.description.searchVisibility, dc.source, dc.source.uri, dc.type, dc.rights.accessRights as shown in Figure 36

- **sortKey (Optional)**
  This is a reserved key that derives values to follow the ordering rule for every node. If a particular Level is to be arranged in ascending or descending order, the order is defined by using the reserved keyword <sortKey>. <sortkey> derives its values from its respective

metadata field as mentioned in the column "sourceField". The criteria to sort should be defined in the "comments" column against that sort key. There could be different sorting scenarios depending on the data. For example, in Figure 39 <sortKey> defines Level 1 (volume) and 2 (issue) to be ordered in descending order respectively. In Figure 41 <sortKey> defines Level 1 to be ordered in descending order by year while Level 2, 3, 4, and item level to be ordered in ascending order. Though <sortKey> is an optional element but preferrable as it allows the logical ordering or arrangement of data in a hierarchy.

Key points to be noted while defining sortKey:

1. If sortKey derives its values from the "sourceField" that stores textual string value, then the sorting can be done alphabetically in ascending or descending order. The respective expression syntax in the "comments" column will be defined as **Text:descending:bottom or Text:ascending:bottom**
2. If sortKey derives its values from the "sourceField" that stores Integer value, then the sorting can be done numerically in ascending or descending order. The respective expression syntax in the "comments" column will be defined as **Integer:descending:bottom or Integer:ascending:bottom**
3. If the value for sort key stores decimal values, then the sorting can be done numerically in ascending or descending order. The expression in the "comments" column will be defined as **Real:descending:bottom or Real:ascending:bottom**
4. If sortKey derives its values from the "sourceField" that stores date value, then the sorting can be done numerically in ascending or descending order. The respective expression syntax in the "comments" column will be defined as **Date:descending:bottom or Integer:ascending:bottom**

- **displayTitle (Optional):**
  This reserve key gives the freedom to the DC to display a desired title in the stitching pane as compared to stored in the respective "dc.title" field. This is quite useful feature when the titles of the contents are too lengthy to be displayed in the stitching pane. Refer Figure 42

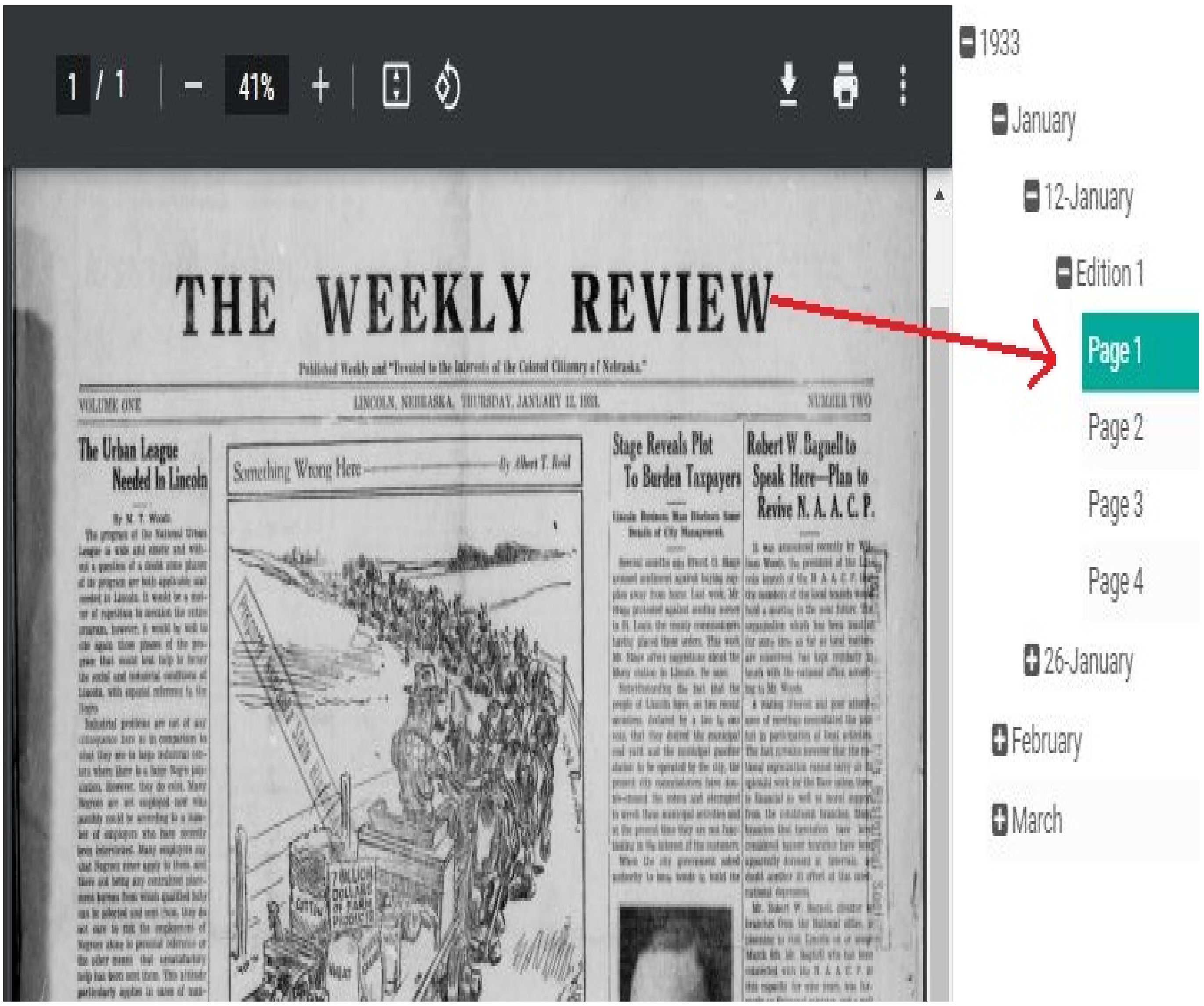

1 / 1
41%
THE WEEKLY REVIEW
The Urban League Needed In Lincoln
Something Wrong Here
Stage Reveals Plot To Burden Taxpayers
Robert W. Bagnell to Speak Here—Plan to Revive N. A. A. C. P.
1933
January
12-January
Edition 1
Page 1
Page 2
Page 3
Page 4
26-January
February
March

Figure 42: DisplayTitle Usage

6.4.3 SourceField

Usage: Optional
One of the two columns, sourceField or sourceValue, must be present. They make reference to the Stitching Template's targetField column mapping information. This column indicates the metadata fields from the originating data whose values will be dynamically used to annotate their corresponding TargetField. The mapping information is indicated in the case of the SourceField column, where values from metadata fields are identified as the values of the hierarchy fields. A field in the targetField column can be mapped to several source fields of the data item.These mappings can be of a single entry or multiple entries. The first value assigned to the target field would stay if the targetField column had a single value. For example, in Table 10 for **Level 1 and 2**, the TargetField **dc.title** may be mapped to the sourceField such as **Title_Meta2.addInfo** and **Title_Meta3.addInfo** respectively from the originating data. Incase of SourceField, there is also an interesting concept of **Dummy Fields** that need to be remembered and has been explained below.

Dummy Field is an interesting feature of the SourceField column in Stitching Template. Sometimes, a SourceField may be temporarily created in the originating data to store values that may be used to annotate their corresponding TargetFields at the different levels. These SourceFields, once annotate their respective TargetFields can be deleted later and are called Dummy Fields. For example, in Table 10 at the **item Level**, "displayTitle" in the TargetField Column may derive its value from a dummy field (Title _ Meta6.addInfo) temporarily created in the SourceField column. Similarly for hierarchy levels 1, 2, and 3 **dc.title** in the TargetField column may be mapped to three respective dummy fields in the SourceField column. The TargetField **sortKey** may also use values mapped to a dummy field.

6.4.4 SourceValue

Usage: Optional
As stated in the brief explanation of the column SourceField, it is required to have one between sourceField and sourceValue.A hierarchy characteristics field has the ability to store externally assigned static values in addition to data from other fields. However, depending on the analysis and support provided by a domain curation expert, such targetFields may contain both mappings from a sourceField and a sourceValue. It's also crucial to remember that numerous mapping entries would be needed if sourceValue were to be assigned to a targetField in a multivalued manner.This column defines the static values that will be used for static annotation of their corresponding TargetField. These values are case sensitive. For example, in Figure 36 the TargetField **dc.description.searchVisibility** has the value **false** for every virtual node except for Root Level and **true** for every real node. Similarily, the TargetField **dc.rights.accessRights** may also have the values **open, subscribed, authorised or limited** depending on the content undergoing the curation process.

6.4.5 Comments

Usage: Optional
One important column in the template is Comments, which gives information on how a hierarchical level is sorted. Each level in the collection hierarchy is represented by **dc.title** in the TargetField column. **dc.title** for each level would depend on the SourceField or possibly SourceValue mapping for that specific level. It is also possible to map a sourceValue to a level, which is frequently used to establish the hierarchy's root level. For every level in a collection hierarchy, there can be more than one entry. As we've already seen, the primary goal of data content stitching is to create a consolidated display that shows items at a specific level in a hierarchical fashion. As a result, sorting the values for a level may occasionally become necessary. The column Comments indicates the triplet structure that specifies this sorting information. The triplet sorting structure is mentioned as **<dataType>:<sorting_order>:<position_of_orphans>**. Each element of the triplet sorting structure will be discussed in detail further

- **Datatype (Mandatory):**

  The first element **<datatype>** specifies the class of the data that are stored in the "sortKey" field. These datatypes may be either of date, integer, real or text categories. Hence the following datatype keyword can be used in the syntax.

Datatype	Syntax Keyword
Integer	Integer
Text or String	Text
Date	Date
Real or Decimal or Float	Real

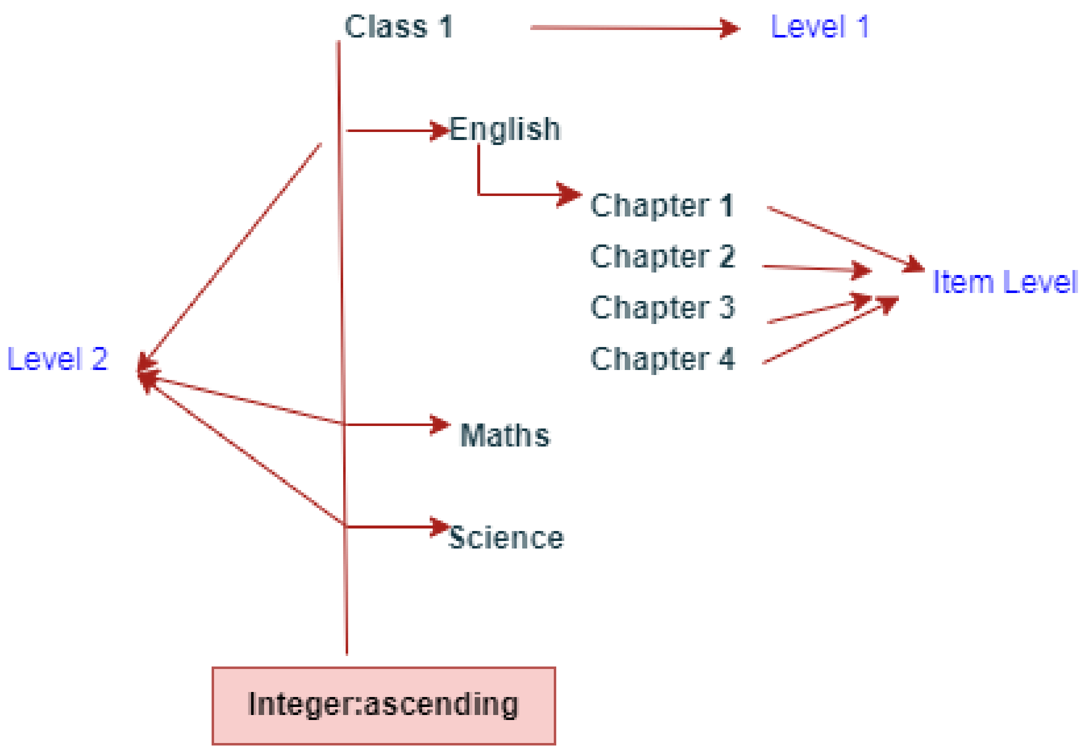


Figure 43: Integer:ascending

It is to be mentioned that a field can hold one single type of value. Hence they can only be sorted in one way across the data. If the hierarchy requires sorting of mixed types of data then these need to be brought into a singular format through curation and then proceed for stitching operation. For example, in Figure 41 Levels 1, 2, 3, are ordered by the "sortKey" of the datatype **date**, hence the datatype as "Date" whereas item level is ordered by page number of the datatype "Integer".

- **Order (Mandatory)** <order> can be ascending or descending. For example, in Figure 41, Level 1 is ordered by year of the datatype "date" and order "descending", hence the comment **"date:descending"** whereas Levels 2, 3 are ordered by the datatype "date" and order "ascending" hence the comment **"date:ascending"**. Levels 3 and item are ordered in ascending order by edition number and page number respectively, hence the comment **"Integer:ascending"**. Similarily, Figure 43 shows a hierarchy collection of book chapters that are arranged subject and class wise. Class 1 English Chapters are arranged by ascending order of chapter number, hence the comment **"Integer:ascending"**.

- **Orphan Item Position [Top or Bottom] (Optional):** Sometimes a content may not have a value for the immediate parent and get randomly sorted in the hierarchy. However, this element allows the placement of a content at the top or bottom of in the hierarchy as defined. Figure 44 clearly illustrates the application of the discussed component, where an orphan content has been given a "displayTitle as "Unknown" and placed at the top or bottom of the hierarchy. Consider there is a chapter "Unknown" that does not have the chapter name. Such a content will be placed at the top if the comment is **"Integer:ascending:top"** Figure 44(a) and placed at the bottom if the comment is **"Integer:ascending:bottom"** Figure 44(b).

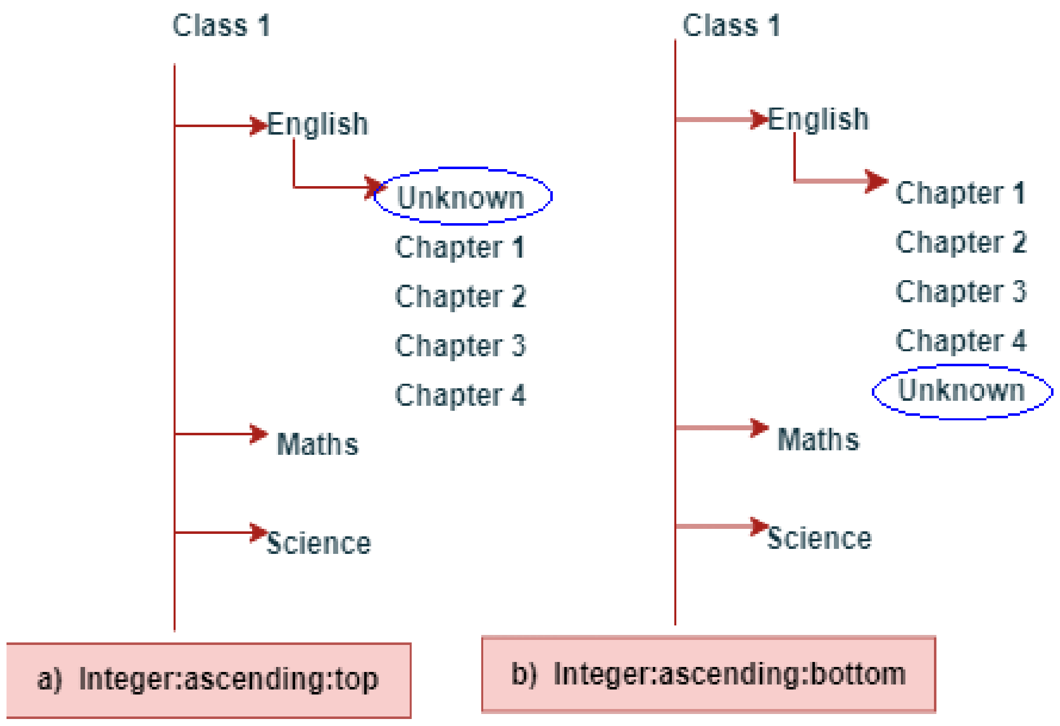


Figure 44: Placement of Orphan Nodes in a Stitching Hierarchy

Note to Remember: It is significant to remember that the stitching template is executed without any curation. Pre-stitching is when all the fields and parameters needed for stitching are fully curated. The mapping of the pertinent parameters is all that is required to execute the stitching template. Within the metadata management architecture, relationships between objects are established using the Content Stitching approach. This allows for the consolidated display of the desired content without compromising the dependencies and integrity of the connected objects. Out of the three approaches for data curation as mentioned in 3, the Stitching Level Curation is the most advanced.

7 Data Logging and Reporting

Data Logging and Reporting is an essential part of any system performing an action based on user input. The ADCT system is no exception and contains an exhaustively built logging and reporting module. However both features are optional and only enabled by user's extensive requests, i.e., they are not default enabled.

7.1 Data Logging

The data logging feature is called the auditing in the ADCT system. They are enabled at the run.properties configuration file to audit the data translation which has happened to specific items. The parameter which enables the Data Audit action ADCT is called **audit-handle**. For each handle a log file is generated which contains the transformations that happened to the fields.

8 Enhancements and Bug Fixes

By the nature of its purpose, the ADCT system is supposed to and has already gone through several Enhancement Requests and Bug Fixes. Since the system is in its very nascent state the Bug/Enhancement management system is not fancy, but has been an integral part of the tool and is sufficient. The Google Sheet has done quite well to serve the purpose. Any request for Enhancement, Change or Bug Fix needs to be entered in this Spreadsheet and the solution concerning the request ID will be posted and updated in this section.

9 Appendix

9.1 Appendix I: ADCT Tool Demo

This is a ADCT Tool Demo template.

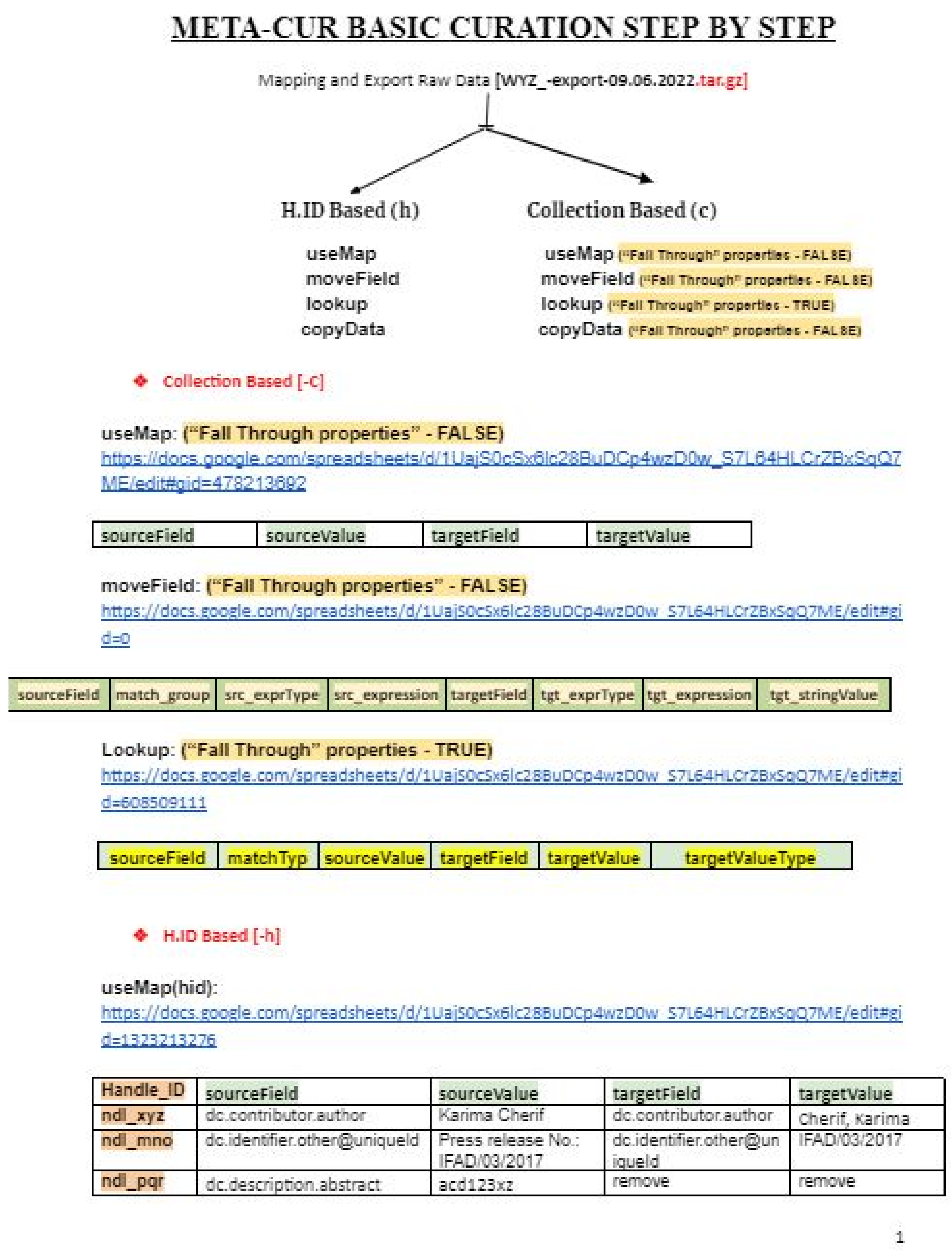
9.2 Appendix II: Script Example

This is how a Script template would look like.

{
"Fields": {
"dc.coverage.temporal": {
"action": [
"deleteField"
]
},
"dc.coverage.spatial": {
"action": [
"deleteField"
]
},
"dc.description.sponsorship": {
"action": [
"deleteField"
]
}
}
}
	Delete Field

"{
""Fields"": {
""dc.contributor.author"": {
""action"": [
""useMap"",
""copyData""
],
""useMap"": {
""inputFile"": ""useMap.xlsx"",
""delimiter"": ""|"",
""action"": [
""moveField""
],
""moveField"": {
""inputFile"": ""moveField.xlsx""}
}
}
}
}"
	Nesting

"{ ""Fields"": { ""dc.subject.ddc"": { ""validation"": true }, ""dc.title"": { ""action"": [""moveField"", ""copyData""], ""moveField"": { ""filter"": [""transcript_filter""], ""transcript_filter"": { ""fields"": [""field1""], ""field1"": { ""fieldName"": ""ndl.sourceMeta.additionalInfo@authorInfo"", ""criteria"": ""equals"", ""fieldValue"": ""transcript"" }, ""expr"": ""$field1"", ""inputFile"": ""moveField.Title_Trascript.xlsx"" } } } } }"	Full Script Example
"{ ""Fields"": { ""dc.title.alternative"": { ""action"": [""copyData""], ""copyData"": { ""filter"": [""Lecture Notes""], ""Lecture Notes"": { ""fields"": [""field1""], ""field1"": { ""fieldName"": ""dc.title"", ""criteria"": ""equals"", ""fieldValue"": ""Lecture Notes"" }, ""expr"": ""$field1"", ""targetField"":""dc.title"", ""targetValue"":""*value* : Lecture Notes"" }, ""targetField"":""dc.title.alternative"" } } } }"	Script for filter (copyData)



META-CUR BASIC CURATION STEP BY STEP

Mapping and Export Raw Data [WYZ_-export-09.06.2022.tar.gz]

H.ID Based (h)	Collection Based (c)
useMap	useMap ("Fall Through" properties - FALSE)
moveField	moveField ("Fall Through" properties - FALSE)
lookup	lookup ("Fall Through" properties - TRUE)
copyData	copyData ("Fall Through" properties - FALSE)

- Collection Based [-C]

useMap: ("Fall Through properties" - FALSE)
https://docs.google.com/spreadsheets/d/1UajS0cSx6lc28BuDCp4wzD0w_S7L64HLCrZBxSqQ7ME/edit#gid=478213692

sourceField	sourceValue	targetField	targetValue

moveField: ("Fall Through properties" - FALSE)
https://docs.google.com/spreadsheets/d/1UajS0cSx6lc28BuDCp4wzD0w_S7L64HLCrZBxSqQ7ME/edit#gid=0

sourceField	match_group	src_exprType	src_expression	targetField	tgt_exprType	tgt_expression	tgt_stringValue

Lookup: ("Fall Through" properties - TRUE)
https://docs.google.com/spreadsheets/d/1UajS0cSx6lc28BuDCp4wzD0w_S7L64HLCrZBxSqQ7ME/edit#gid=608509111

sourceField	matchTyp	sourceValue	targetField	targetValue	targetValueType

- H.ID Based [-h]

useMap(hid):
https://docs.google.com/spreadsheets/d/1UajS0cSx6lc28BuDCp4wzD0w_S7L64HLCrZBxSqQ7ME/edit#gid=1323213276

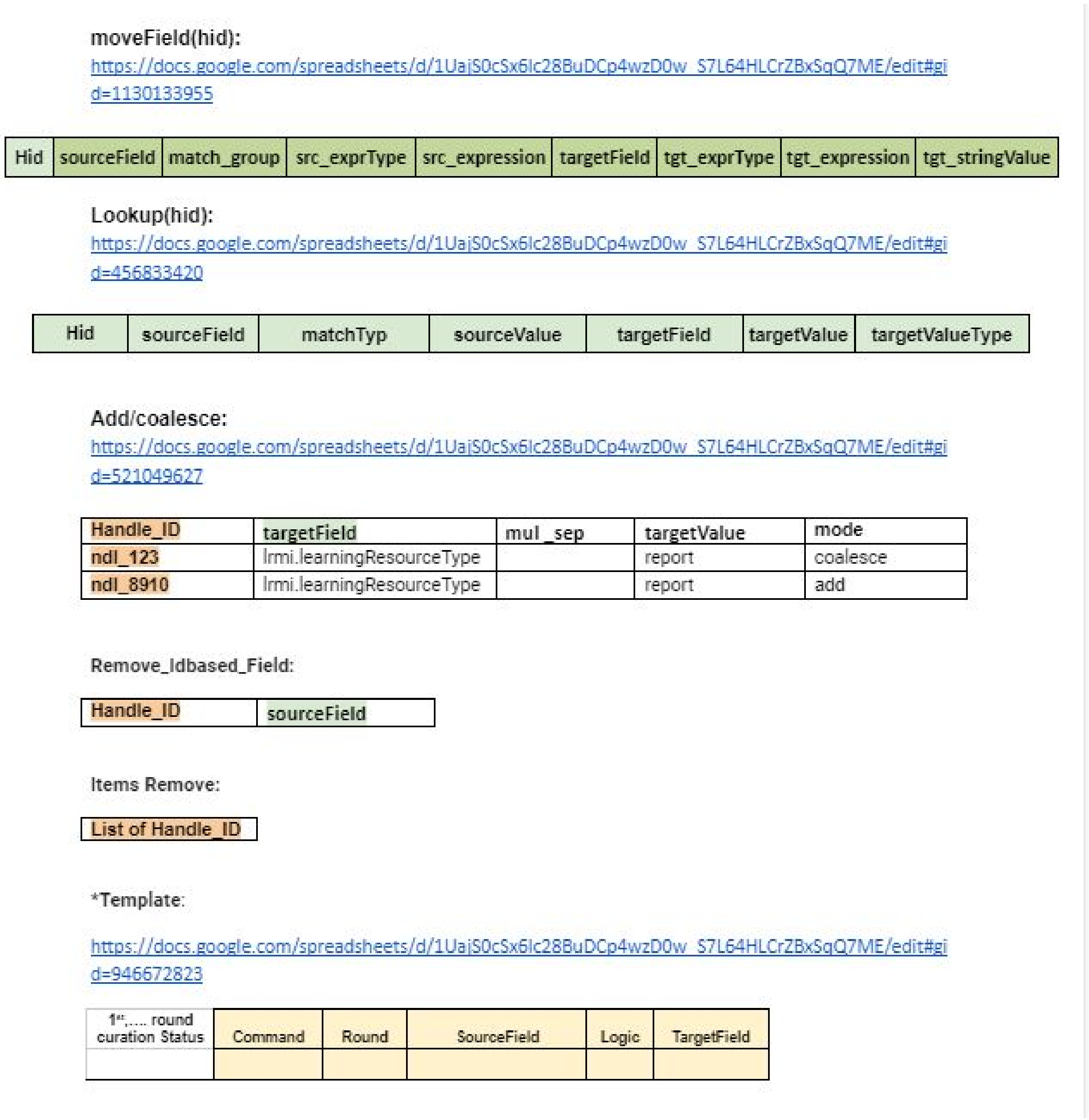
Handle_ID	sourceField	sourceValue	targetField	targetValue
ndl_xyz	dc.contributor.author	Karima Cherif	dc.contributor.author	Cherif, Karima
ndl_mno	dc.identifier.other@uniqueId	Press release No.: IFAD/03/2017	dc.identifier.other@uniqueId	IFAD/03/2017
ndl_pqr	dc.description.abstract	acd123xz	remove	remove

1

Figure 45: ADCT Tool Demo

9.3 Appendix III: HID_useMap Template

This is how a useMap by HID template would look like.



moveField(hid):
https://docs.google.com/spreadsheets/d/1UajS0cSx6lc28BuDCp4wzD0w_S7L64HLCrZBxSqQ7ME/edit#gid=1130133955

Hid	sourceField	match_group	src_exprType	src_expression	targetField	tgt_exprType	tgt_expression	tgt_stringValue

Lookup(hid):
https://docs.google.com/spreadsheets/d/1UajS0cSx6lc28BuDCp4wzD0w_S7L64HLCrZBxSqQ7ME/edit#gid=456833420

Hid	sourceField	matchTyp	sourceValue	targetField	targetValue	targetValueType

Add/coalesce:
https://docs.google.com/spreadsheets/d/1UajS0cSx6lc28BuDCp4wzD0w_S7L64HLCrZBxSqQ7ME/edit#gid=521049627

Handle_ID	targetField	mul_sep	targetValue	mode
ndl_123	lrmi.learningResourceType		report	coalesce
ndl_8910	lrmi.learningResourceType		report	add

Remove_Idbased_Field:

Handle_ID	sourceField

Items Remove:

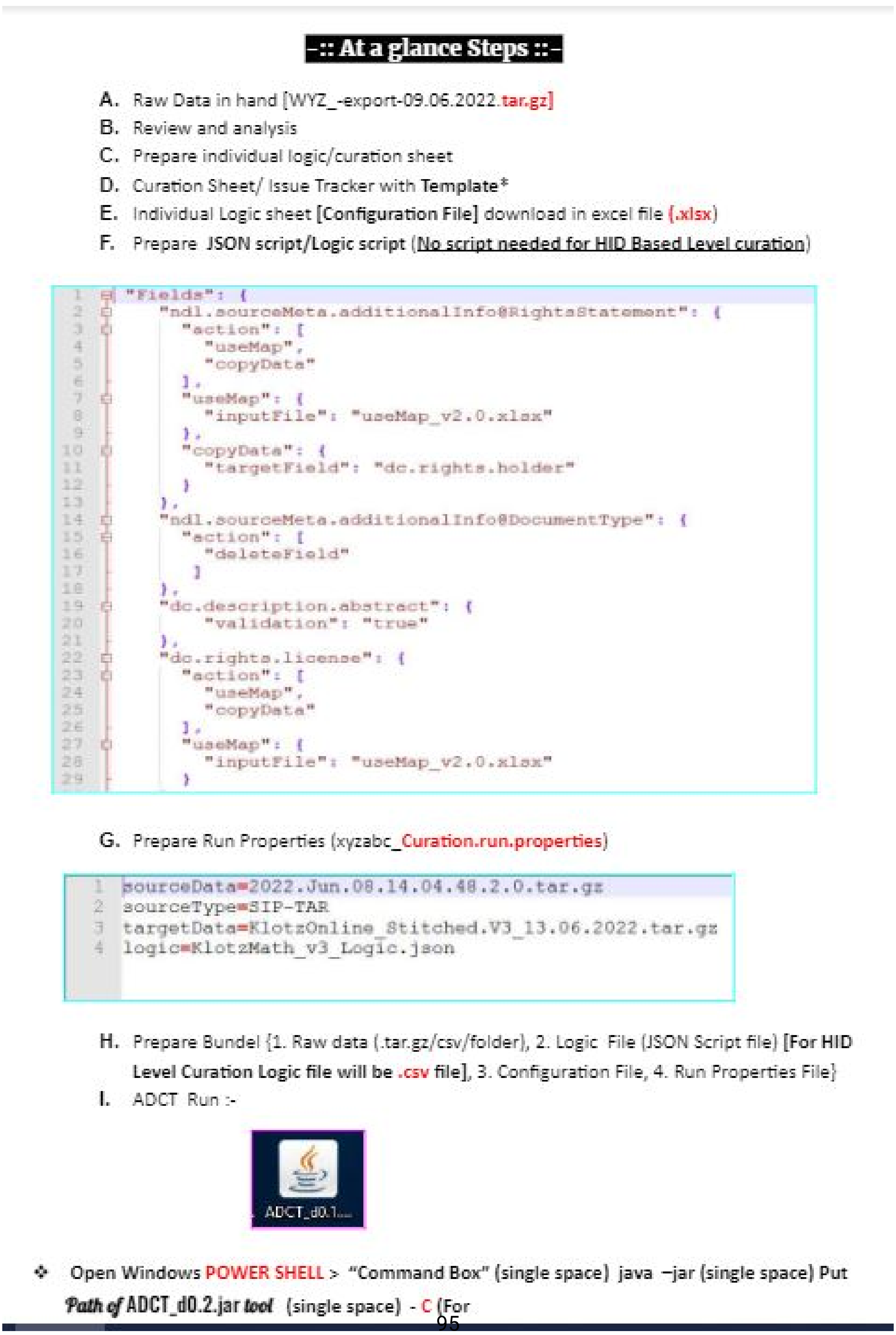
List of Handle_ID

*Template:

https://docs.google.com/spreadsheets/d/1UajS0cSx6lc28BuDCp4wzD0w_S7L64HLCrZBxSqQ7ME/edit#gid=946672823

1st,.... round curation Status	Command	Round	SourceField	Logic	TargetField

Figure 46: ADCT Tool Demo



-:: At a glance Steps ::-

A. Raw Data in hand [WYZ_-export-09.06.2022.tar.gz]
B. Review and analysis
C. Prepare individual logic/curation sheet
D. Curation Sheet/ Issue Tracker with **Template***
E. Individual Logic sheet **[Configuration File]** download in excel file (.xlsx)
F. Prepare **JSON script/Logic script** (<u>No script needed for HID Based Level curation</u>)

```
"Fields": {
    "ndl.sourceMeta.additionalInfo@RightsStatement": {
      "action": [
        "useMap",
        "copyData"
      ],
      "useMap": {
        "inputFile": "useMap_v2.0.xlsx"
      },
      "copyData": {
        "targetField": "dc.rights.holder"
      }
    },
    "ndl.sourceMeta.additionalInfo@DocumentType": {
      "action": [
        "deleteField"
       ]
    },
    "dc.description.abstract": {
        "validation": "true"
    },
    "dc.rights.license": {
      "action": [
        "useMap",
        "copyData"
      ],
      "useMap": {
        "inputFile": "useMap_v2.0.xlsx"
      }
```

G. Prepare Run Properties (xyzabc_Curation.run.properties)

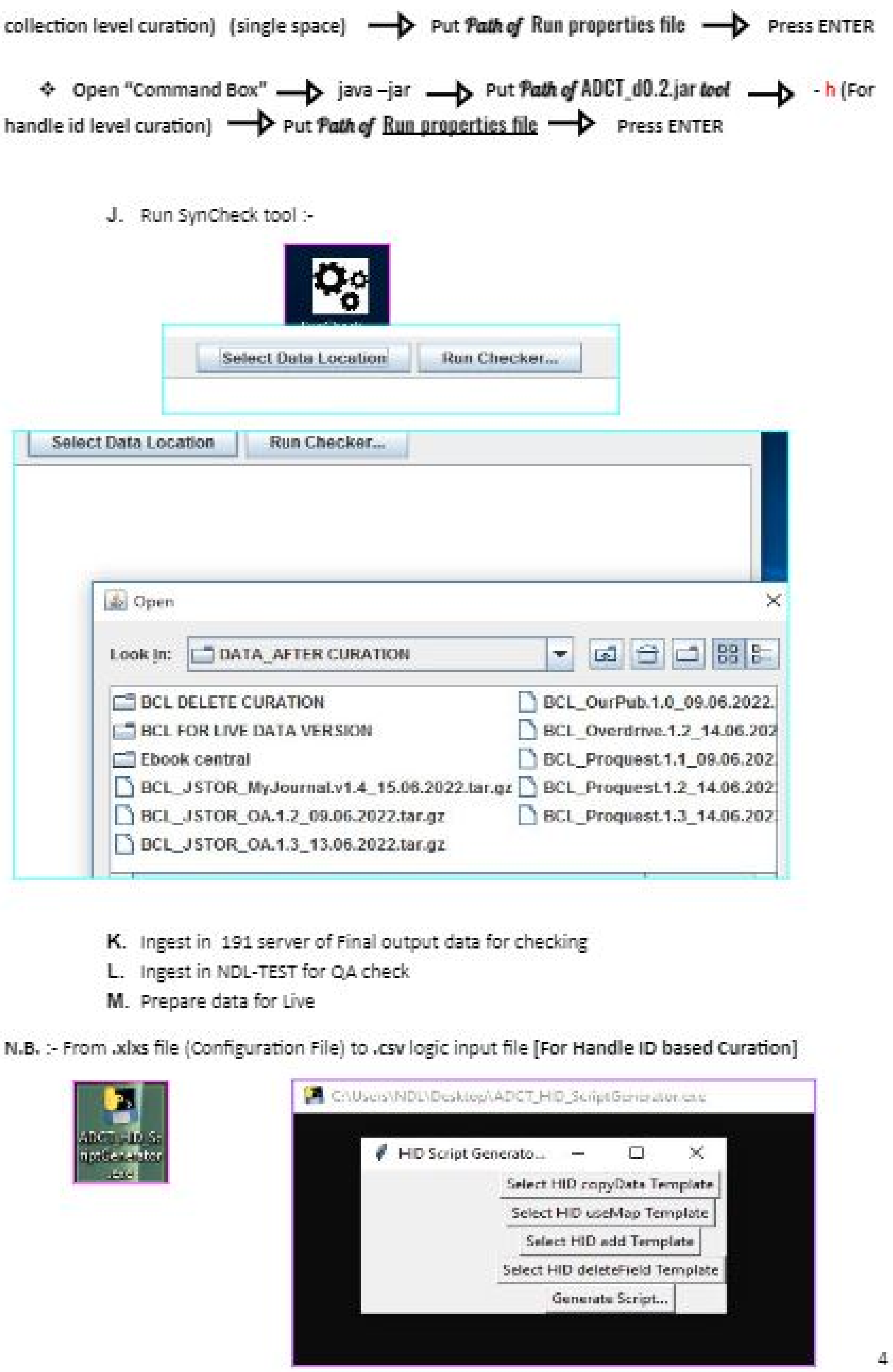
```
sourceData=2022.Jun.08.14.04.48.2.0.tar.gz
sourceType=SIP-TAR
targetData=KlotzOnline_Stitched.V3_13.06.2022.tar.gz
logic=KlotzMath_v3_Logic.json
```

H. Prepare Bundel {1. Raw data (.tar.gz/csv/folder), 2. Logic File (JSON Script file) **[For HID Level Curation Logic file will be .csv file]**, 3. Configuration File, 4. Run Properties File}
I. ADCT Run :-

❖ Open Windows POWER SHELL > "Command Box" (single space) java –jar (single space) Put ***Path of* ADCT_d0.2.jar *tool*** (single space) - C (For

95

Figure 47: ADCT Tool Demo


collection level curation) (single space) ⟶ Put *Path of* Run properties file ⟶ Press ENTER

- Open "Command Box" ⟶ java –jar ⟶ Put *Path of* ADCT_d0.2.jar *tool* ⟶ - h (For handle id level curation) ⟶ Put *Path of* Run properties file ⟶ Press ENTER

J. Run SynCheck tool :-

Select Data Location | Run Checker...

Open

Look In: DATA_AFTER CURATION

```
BCL DELETE CURATION                          BCL_OurPub.1.0_09.06.2022.
BCL FOR LIVE DATA VERSION                    BCL_Overdrive.1.2_14.06.202
Ebook central                                BCL_Proquest.1.1_09.06.202
BCL_JSTOR_MyJournal.v1.4_15.06.2022.tar.gz   BCL_Proquest.1.2_14.06.202
BCL_JSTOR_OA.1.2_09.06.2022.tar.gz           BCL_Proquest.1.3_14.06.202
BCL_JSTOR_OA.1.3_13.06.2022.tar.gz
```

K. Ingest in 191 server of Final output data for checking

L. Ingest in NDL-TEST for QA check

M. Prepare data for Live

N.B. :- From .xlxs file (Configuration File) to .csv logic input file **[For Handle ID based Curation]**

ADCT_HID_ScriptGenerator.exe

C:\Users\NDL\Desktop\ADCT_HID_ScriptGenerator.exe

HID Script Generato...

Select HID copyData Template
Select HID useMap Template
Select HID add Template
Select HID deleteField Template
Generate Script...

4

Figure 48: ADCT Tool Demo

Hid	sourceField	sourceValue	targetField	targetValue
12345	dc.contributor.author	Karima Cherif	dc.contributor.author	Cherif, Karima
12346	dc.identifier.other@uniqueId	Press release No.: IFAD/03/2017	dc.identifier.other@uniqueId	IFAD/03/2017
12347	dc.description.abstract	This report is the most...	remove	remove
12348	dc.description.abstract	This report is the fruit ...	remove	remove
12349	dc.description.abstract	This report is the most...	remove	remove

Table 11: Hid_useMap

9.4 Appendix IV: moveField Template

This is how a moveField template would look like.

sourceField	src_exprType	src_expression	targetField	tgt_exprType	tgt_e
dc.contributor.editor	containsToken	editor	dc.contributor.editor	**replaceToken**	editor
dc.contributor.editor	containsToken	Mrs	dc.contributor.editor	**replaceToken**	Mrs
dc.contributor.author	containsToken	Mr.	dc.contributor.author	**replaceToken**	Mr.
dc.identifier.isbn	contains	doi:	identifier.other@doi	**replace**	doi:
dc.identifier.isbn	contains	pdf,	dc.format.mimetype	**move**	
dc.language.iso	contains	[Farsi]	remove	remove	
dc.subject	count:=1	,	dc.subject	regexReplace	**(.*),(.***
dc.subject	matches	.*	dc.subject	regexReplace	**,\s*$**
dc.contributor.editor	matches	**.*\d+.***	dc.contributor.editor	regexReplace	**\d+.***
dc.contributor.editor	matches	.*	dc.contributor.editor	regexReplace	**[]+**
dc.contributor.author	contains	**–**	dc.contributor.author	regexReplace	**(.*)–.***
dc.contributor.author	matches	.*	dc.contributor.author	regexReplace	**([^,]*,[**
dc.contributor.author	matches	**.*[].***	dc.contributor.author	regexReplace	**[]+**
dc.publisher	contains	**name**	dc.publisher	regexReplace	**.*nam**
dc.contributor.author	matches	.*	dc.contributor.author	regexReplace	**^(.*\s)**
dc.contributor.author	endsWith	**.*?(\d+)$**	remove	remove	
dc.identifier.isbn	matchs	\s	dc.identifier.isbn	regexReplace	\s
dc.subject	contains	;	dc.subject	split	;
dc.relation.ispartof	matches	.*	dc.relation.ispartof	replace	****
dc.relation.haspart	matches	.*	dc.relation.haspart	replace	****

Table 12: Move Field

9.5 Appendix V: lookUp Template

This is how a lookUp template would look like.

sourceField	matchTyp	sourceValue	targetField	targetValue	targetValueType
dc.language.iso	contains	storybook	lrmi.learningResourceType	book	value
dc.identifier.isbn	contains	pdf,	dc.format.mimetype	application/pdf	value
lrmi.learningResourceType	equals	Book	dc.format.mimetype	application/pdf	value
lrmi.learningResourceType	equals	News	dc.type	text	value

Table 13: lookup

9.6 Appendix VI: HID_add template

Hid Add/coalesce

1. Must be maintained Columns Order i.e. Hid>targetField>mul _sep>targetValue>Mode 2. Where targetValue is separated by any separator (i.e. |, ||, ; etc.), must be indicated in mul_sep column. 3. If there is no separator in the list, mul_sep column is blank but not deleted the column.				
Hid	**targetField**	**mul _sep**	**targetValue**	**mode**
12345	lrmi.learningResourceType		report	coalesce
12346	lrmi.learningResourceType		report	add

Table 14: Hid Add/coalesce

9.7 Appendix VII: Hid_Delete Field

Hid_Delete FieldS

hid	sourceField
scilit_medknow/a42b5369079def51bc4b3f0d7c162a7a	ndl.sourceMeta.uniqueInfo@relatedContentUrl
scilit_medknow/f1e9e9b84ac5f64f6aed458652b561cc	dc.description.uri
scilit_medknow/a9c6a0d0889993ad06860bd10b0828ae	dc.publisher.place
scilit_medknow/a5a3a44d34121b5771ac550e4dcda232	dc.description.uri
scilit_medknow/c19a4e5f10bc77df3aa102c252f12ce4	ndl.sourceMeta.uniqueInfo@relatedContentUrl
scilit_medknow/c19a4e5f10bc77df3aa102c252f12ce4	dc.description.uri
scilit_medknow/494d5952911584aebebfdf024a9528f8	dc.description.uri
scilit_medknow/271a7d32dc7faac96da3c644ce8e13bb	dc.description.uri
scilit_medknow/5e6d24dab9dcde82a9c84eece73195c7	dc.description.uri
scilit_medknow/3c4ee16c7086164ecc29b05da341d873	dc.description.uri
scilit_medknow/cc8e155d212d86a3583731e61f55e462	dc.publisher.place
scilit_medknow/f1e9e9b84ac5f64f6aed458652b561cc	dc.description.uri
scilit_medknow/485d06204d5c82f1387fa25a1e55c6d6	dc.description.uri
scilit_medknow/6918a5e65e71ae4af307fe8d3a2a4a5b	dc.publisher.place
scilit_medknow/6918a5e65e71ae4af307fe8d3a2a4a5b	dc.rights.uri
scilit_medknow/a42b5369079def51bc4b3f0d7c162a7a	ndl.sourceMeta.uniqueInfo@relatedContentUrl
scilit_medknow/79c1662bd486134e3275ea0f6e830ec1	ndl.sourceMeta.uniqueInfo@relatedContentUrl
scilit_medknow/b9a157f5f28239831652688d4f46b1d7	dc.rights.uri
scilit_medknow/07bf97b8745b46042d010d19e866ea77	dc.rights.uri
scilit_medknow/be31785820ae0e345a718474e54f3a31	dc.description.uri
scilit_medknow/2a9bb9681c341c24bded57458e30954a	dc.rights.uri

Table 15: Hid_Delete Field