Enterprise Data Asset Management (EDAM)

EDAM Community Edition v0

2022-12-17 – Draft v0

Last Updated: 2023-01-21

Table of Contents

[1. Motivation 2](#_Toc125184190)

[1.1 Asset Schemas 2](#_Toc125184191)

[2.0 Sample Projects 2](#_Toc125184192)

[2.1 Communicable Diseases Assets Management 3](#_Toc125184193)

[2.1.1 EDAM Disease Surveillance (DS) Project 3](#_Toc125184194)

[2.1.2 Getting Started 3](#_Toc125184195)

[2.1.3 RVCT Document Sample 3](#_Toc125184196)

[3. EDAM Features 3](#_Toc125184197)

[4. EDAM UI Screens and Forms 5](#_Toc125184198)

[4.1 EDAM App Main Screen 5](#_Toc125184199)

[4.2 EDAM Asset Dictionary 6](#_Toc125184200)

[4.3 EDAM Asset Data Output Options 7](#_Toc125184201)

[4.4 EDAM Asset Tree Structure 9](#_Toc125184202)

[4.5 EDAM Use Case – Mappings 10](#_Toc125184203)

[4.5.1 Use Case Definitions 10](#_Toc125184204)

[4.6 EDAM Use Case – Book – Booklets 11](#_Toc125184205)

[4.6.1 Understanding Books and Booklets 11](#_Toc125184206)

[4.6.2 Execute a Booklet 14](#_Toc125184207)

[4.6.3 Execute Book 14](#_Toc125184208)

[4.6.4 EDAM Use Case Mappings – Map Play (JSONata) 14](#_Toc125184209)

[Appendix A. EDAM Projects Structure 15](#_Toc125184210)

[A.1 EDAM App Data Templates and Projects 15](#_Toc125184211)

[A.2 Projects Folder 15](#_Toc125184212)

[A.3 Understanding the Project Process Arguments 15](#_Toc125184213)

# 1. Motivation

After many years of struggling with spreadsheets, word documents and other forms of documentation while trying to work with Business Analysts (BA) and Architects and collaboratively define an “Enterprise Data Model” it was clear to us that we should strive for a better way to capture and maintain Data Assets information.

There are excellent commercial tools to help is this area still we have not found any tool that offers a substantial set of features to do data assets management that we could rely on without some kind of financial burden to our customers and projects. Not every customer or project has the budget to manage the expenses of a workable “Data Asset Management” (DAM) tool and for this reason BA’s and Architects resolve to use spreadsheets, word documents or other forms of documentation to manage solutions data assets.

After many years of development without any budget and limited time only the hope that we could find a way to provide a tool with some level of usable DAM features to help in the documentation and generation of Data Assets artifacts Datovy is presenting its “Open Source” EDAM solution. With the possible contributions of the larger “Open Source” community we hope to find a better way to make this effort a suitable alternative to Commercial software specially for those projects with limited budget.

EDAM offers output that that intent to always be verifiable with schema definitions and its output to an Excel Workbook a guaranteed consistent with declared entities that is a substantial improvement over handmade Spreadsheets that can be easily break and extremely difficult to maintain.

This December 2022 EDAM is released in GITHUB with a set of limited functionality but enough to be able to quickly generate documentation, schemas, and other artifacts. The rest of this document will try to provide a glimpse into EDAM and existing features. This is not aimed to compete with Commercial offerings and is work in progress therefore expect incomplete functionality and documentation, by-hand configurations or work, bugs to be fixed and other hurdles that will be found while trying to use this product. We have been users of our own software and we think that those hurdles worth the trouble since we could ease the Business Analysis and Data Architecture by quickly generate useable documentation and artifacts. Hopefully we could find some additional contributors that could spend time with us and improve this product.

## 1.1 Asset Schemas

EDAM is schema agnostic and any provided XSD, JSON Schema, (Database) DDL, or other is translated into the EDAM flat structure representation. Today, existing, inclusive, feature rich, and stable schema language it’s an XSD (XML Schema Definition), JSON is still in draft 12 and more drafts to come. SQL DDL is a good option but lack the richness of an XSD including inheritance, union-types among other. As of the current version the XSD and MS-SQL DDL are the schemas that had been widely tested with large and complex “real” and samples, JSON to a less extend since generally customers/developers and others don’t provide schemas for their JSON documents.

# 2.0 Sample Projects

This section provides sample Projects to help the understanding and use of EDAM. A brief description of each follow.

## 2.1 Communicable Diseases Assets Management

At the same time of this release Datovy is also providing another contribution to the “Open Source” community with the release of a database for “Communicable Diseases” or “Disease Surveillance” (DS) with limited and partial support for related CDC messaging.

The DS database is used in the first release of EDAM as a sample non-trivial collection of data entities and components that show case supported features.

### 2.1.1 EDAM Disease Surveillance (DS) Project

The “Disease Surveillance Project” can be found in:

[esobrino/Datovy.Edam: Datovy Enterprise Data Asset Management (github.com)](https://github.com/esobrino/Datovy.Edam)

There the “Edam.App.Data” folder will be found and as shown in Appendix A, it contains EDAM projects scripts, definition files and other artifacts.

### 2.1.2 Getting Started

The easy way to learn about EDAM is to use an existing project and see how it work and what features the application support (see list in section 3 ahead). The EDAM was originally a command line only tool and a WinUI interface has been built to drive the command line arguments. To start select a project that contain some valid JSON Arguments file and, in this case, use the following:

…/Datovy.Edam/Edam.App.Data/Projects/Datovy.HC.CD/Arguments/0001.HC.CD.ToDictionary.Args.json

EDAM uses the Microsoft Monaco Code Editor to allow editing these files. In section 4.1 the above file has been selected and is visible in this editor. Review the structure of the argument file as explained in Appendix A.3 section.

To read the DS database schema and generate a schema agnostic flat representation press the “Execute” button (see section 4.1). This button is at the right of the “Save” button in the “File” TAB as shown in Figure 4.1.

Once the schema is read the result will show how many data elements had been identified and the flat schema can be viewed in the “Asset” TAB as shown in Figure 4.2.

Now the remaining functionality can be explored by moving around options within the “Asset” TAB.

### 2.1.3 RVCT Document Sample

Datovy is working on the definition of an XSD schema to support the “Report of Verified Case of Tuberculosis” based on the related CDC RVCT form template. Using this schema, the EDAM functionality can be tested.

# 3. EDAM Features

The list of supported features include:

|  |  |  |
| --- | --- | --- |
| **Feature** | **Description** | **See** |
| *EDAM Projects Support* | | |
| EDAM Projects | A project is composed of an expected minimum folder structure that should include an “Arguments” folder that define details about the Asset to be managed. | 4.1 |
| *Schema Reading Support* | | |
| XSD to EDAM | Convert a complex multi-namespace XSD into EDAM assets definition set. |  |
| JSON Schema to EDAM | Convert a complex multi-namespace JSON schema into EDAM assets definition set. |  |
| DDL Definitions to EDAM | Convert MS-SQL, MySql, or Oracle schema definitions into EDAM assets definition set. Support for multiple schemas and related namespace data elements collection sets. |  |
| EDI to EDAM | Convert an enhance EDI definition into EDAM assets definition set (partial support). |  |
| *Schema Writing Support* | | |
| EDAM to XSD | Output to XSD retaining all namespace information. |  |
| EDAM to JSON Schema | Output to most recent JSON Schema Draft. |  |
| EDAM to DDL | For the time being only MS-SQL, others later including multiple namespaces to schemas support. |  |
| EDAM to JSON-LD | Output to JSON-LD Components and definitions. |  |
| *Use Case Definitions and Mappings (on the works)* | | |
| EDAM to Use Case | Identify and document elements from an Asset that are part of a use case. | 4.5 |
| EDAM to Use Case Mappings | Identify and document elements involved in transformations and mappings. | 4.6 |
| EDAM to Use Case Subset | Given a Use Case definition that uses a subset of an asset, output its (XSD) schema. Use the “UseCaseToFile” directive (see Appendix A). |  |
| *Other Supported Features* | | |
| EDAM to Data Dictionary | Excel generated file given an EDAM assets definition set. |  |

Samples for most of the above had been produced for the “Disease Surveillance” database (Visit the related “Datovy Communicable Disease” Open-Source community project).

See the full list of Procedure options (that match the above list of features) in:

Edam.Data.AssetConsole.AssetConsoleProcedure *(enumerator)*

# 4. EDAM UI Screens and Forms

This section contains a few screenshots of the EDAM UI Windows Application (App).

## 4.1 EDAM App Main Screen

The Main Screen display the list of projects (in the left-hand side) and 3 Tabs (Domains, File, and Asset) on the right-hand side (see Figure 4.1). The App projects path is configurable and can be set in the “appsettings.json” by setting the “AssetConsolePath” to the desired folder path.

Graphical user interface, text, application

Description automatically generated

Figure 4.1 Projects and Arguments files.

The App Visual Studio project has a default project configuration and files for the “Disease Surveillance” database that serves as an example and can be used to test existing functionality.

Within the “File” TAB 2 buttons are found, the first to “Save” updated to the selected Arguments file in display, and the second to execute the process to read the defined source data and generate the EDAM data dictionary (see 4.2 for more details).

## 4.2 EDAM Asset Dictionary

After the EDAM dictionary is generated, the results are displayed in the “Asset” TAB. The number of elements is displayed, and the “NAMESPACES” drop down list contains all found namespaces withing the source data (if any).

Graphical user interface, application, table

Description automatically generated

Figure 4.2. Asset Dictionary

The “ASSET” panel has 3 buttons and a dropdown box. The first button is related to the screen as shown in 4.2, the second will display the Asset data components in a Tree structure (see 4.4).

## 4.3 EDAM Asset Data Output Options

Within the “ASSET” panel (on the left) save options are provided (see Figure 4.3).

Graphical user interface, application, table

Description automatically generated

Figure 4.3. EDAM Asset Data Output Options.

The output options include:

|  |  |  |
| --- | --- | --- |
| **Option** | **Description** | **Status** |
| XSD | XML Schema | Available |
| JSON-Schema | Will Output as to latest Draft Specification. | Available |
| GRAPH-SQL (GQL) | TigerGraph GQL. | Testing |
| Database | Output to MS-SQL EDAM database. The connection string should be specify in the “appsettings.json” file. | Available |
| DDL | MS-SQL Schema definition. | Available |
| Excel | Data Dictionary as an Excel Workbook. | Available |
| Data Template File | Data Template file definition. | Testing |

The output is sent to the Projects “Document” folder and for the “Disease Surveillance” EDAM includes:

Graphical user interface, application

Description automatically generated

Figure 4.3.1. EDAM Output folders with generated documentation or schemas.

Once the “Database” output is selected the definitions are stored in the DataElement table (see 4.3.2).

Graphical user interface, application, table, Excel

Description automatically generated

Figure 4.3.2. EDAM Database showing the DataElement table with loaded definitions.

## 4.4 EDAM Asset Tree Structure

The EDAM Asset data set can be displayed in a Tree Structure format by choosing the “folder” icon to the right of the “ASSET” label (see Figure 4.4).

Graphical user interface, application, table

Description automatically generated

Figure 4.4. Asset as a Tree Structure.

Once, in this case the “Disease Surveillance” DDL database definition is read by the application the schemas and tables details are use to create a hierarchy representation of the data as shown in Figure 4.4 whose root element is “Disease\_Surveillance\_Document”. The “\_Document” is automatically appended to the root element.

## 4.5 EDAM Use Case – Mappings

While selecting the “ASSET” mappings option the “Use Cases” TAB is shown. Already defined Use Cases will be listed and can be selected to continue working on those.

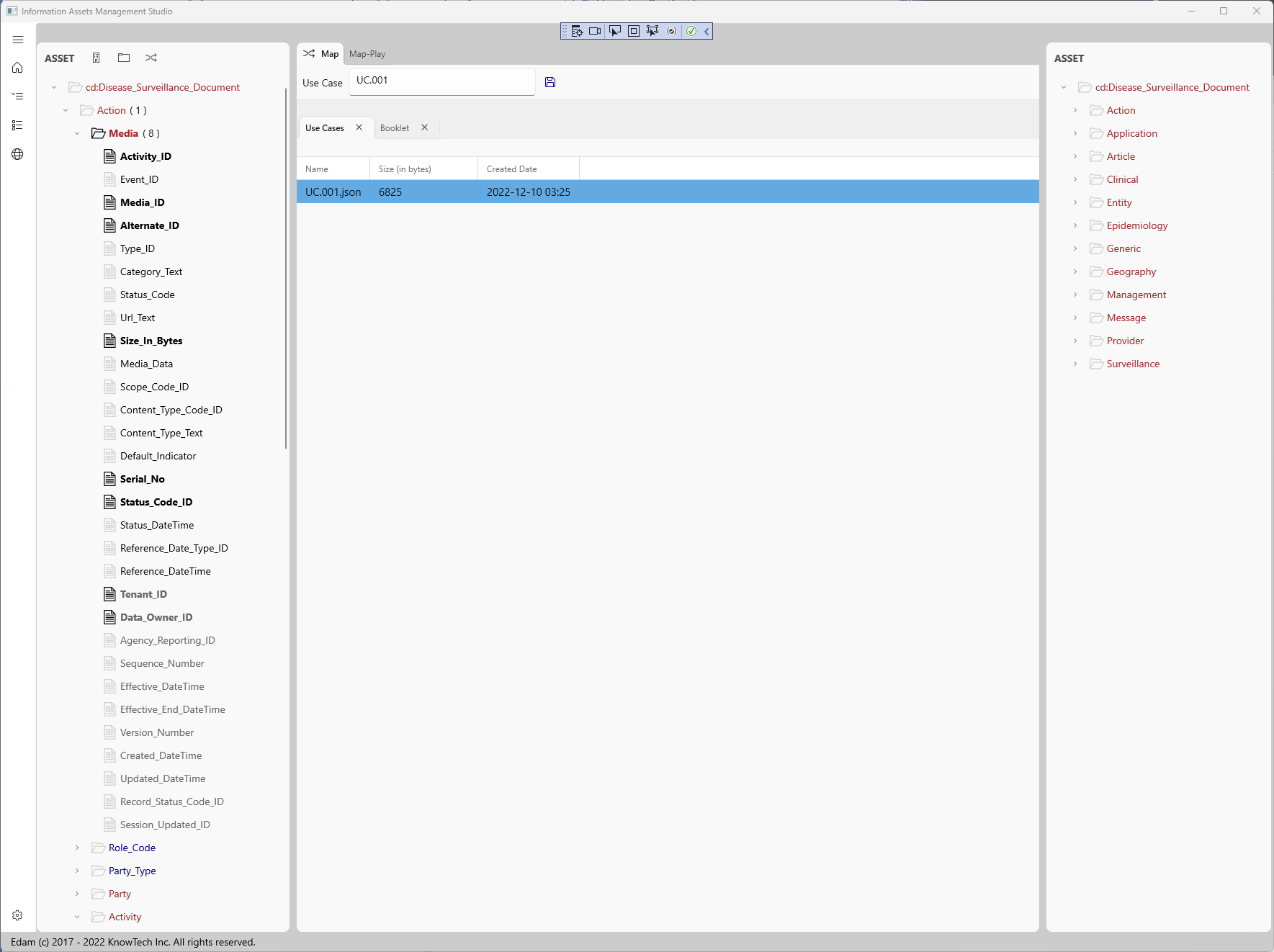


Figure 4.5. EDAM Use Case TAB and source (left) – target (right) Asset Trees.

While selecting items from the Source or Target the Map panels display them and Booklets definitions can be defined (see 4.6).

### 4.5.1 Use Case Definitions

For discussion purposes an Asset is defined here as a collection of data entities that define a business artifact or area such as a the “Disease Surveillance” database.

A Use Case can be defined by identifying the data components and child elements that are relevant to a business scenario. The “Disease Surveillance” database sample shows multiple data components over various schemas such as “Clinical”, “Epidemiology” and others and contain over 3,000 data entities. Within an Asset schema there may be too many elements and it will be impractical to consistently maintain spreadsheets or Word documents by hand to describe the subset of a given Use Case (business scenario). A possible solution is to derive the subset directly from the full Asset (database) that may be needed for information exchanges, reporting or other.

## 4.6 EDAM Use Case – Book – Booklets

As an example, the selection of a data-element on the source will display the mapping panel showing its path withing the document and any related Booklet associated with this reference data-element (see Figure 4.6).

Graphical user interface, text, application, email

Description automatically generated

Figure 4.6. Use Case – Book / Booklet.

Like Jupiter Notebooks the inner Use Case mapping Book can have a booklet per each mapped item allowing the definition of one or more Text or Code cells. In this example the Code cell uses “JSONata” as the language that the code will support. By clicking in the execute button the output will show beneath the code box.

### 4.6.1 Understanding Books and Booklets

The traditional Object mapping tools uses a graphical interface to allow you to select source items, one or more transformations and target items. For small and trivial mappings those tools are an excellent choice, but as the mapping exercise requires large number of mappings those become cluttered, difficult to understand, maintain, and use. The EDAM goes in a different direction and items selected are kept in small collections that are manageable, easy to document and to define needed mappings by using Booklets. On this EDAM version the “Map Items” source and target are not graphically defined, and enhancement that maybe a future feature.

Use Cases are kept by default in the project “Use Cases” folder, for the “Disease Surveillance” sample those will be found in:

…/Edam.AppData/Projects/Datovy.HC.CD/UseCases

Every Use Case has a related mappings “Book” (see Figure 4.6.1). Within the “Map” TAB and while working with a Use Case the Booklet TAB is shown in Figure 4.6. Above the Booklet TAB the configured “Map Item” (MI) is shown. A MI contains one or more selected “source items” (at the left) and “target items” (at the right). This collection of items is uniquely identified and a reference to it is used to identify the corresponding “Booklet” within the Use Case “Book”.

A picture containing graphical user interface

Description automatically generated

Figure 4.6.1 Map Item – Book – Booklets Structure

Inspecting a Use Case JSON file shows the “Map Items” collection and the “Book” with its related “Booklets” (one per each Map Item).

#### 4.6.1.1 Sample Use Case Reports

The existing EDAM version Excel Report Data Dictionary for a project contain the “Use Cases” TAB. A summary of this report is layout (not including all supported columns) as follows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity** | **Element** | **FullPath** | **UseCase** | **MapTo** | **Function** | **Instructions** |
| cr:ClinicalReview | cr:ReviewDate | cr:ClinicalReview/cr:ReviewDate | UC-001 | tr:ReviewReport/tr:ReportYear | year(date) | Fetch year from ReviewDate |
|  |  |  | UC-001 | tr:ReviewReport/tr:ReportMonth | month(date) | Fetch month from ReviewDate |
|  |  |  | UC-001 | tr:ReviewReport/tr:ReportDay | day(date) | Fetch day from ReviewDate |

The previous example maps a Date into its date parts on the (“MapTo”) target. The opposite may look like:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity** | **Element** | **FullPath** | **UseCase** | **MapTo** | **Function** | **Instructions** |
| tr:ReviewReport | tr:ReportYear | tr:ReviewReport/tr:ReportYear | UC-002 | cr:ClinicalReview/cr:ReviewDate | date(year,month,day) | Prepare date from Report Year, Month, Day |
| tr:ReviewReport | tr:ReportMonth | tr:ReviewReport/tr:ReportMonth | UC-002 |  |  |  |
| tr:ReviewReport | tr:ReportDay | tr:ReviewReport/tr:ReportDay | UC-002 |  |  |  |

The above example maps individual date component (Year, Month, Day) into a date.

### 4.6.2 Execute a Booklet

As shown a Booklet is a collection of text and code segments, when executed each code segment will generate a subset of the expected mapping and putting those together by merging those should produce the expected target JSON document for the Booklet Map-Item. As of this version this is experimental and additional functionality may be added to JSONata such as a library of functions that will help in joining and merging those code segments.

### 4.6.3 Execute Book

Similarly, a Book is a collection of Booklets that when executed each will produce a subset of the target document, and by joining and merging those it is expected that the final full target document be produced. Additional functionality may be added later to store the generated document.

### 4.6.4 EDAM Use Case Mappings – Map Play (JSONata)

The “Map Play” TAB offers the opportunity to execute code based on selected Language, for this version only JSONata is supported. Given an instance of a document for the source (left side) Tree a JSON sample is automatically generated by clicking the “Sample” label and display in the panel as shown in Figure 4.6.4.

Graphical user interface, text

Description automatically generated

Figure 4.6.4 Use Case Mappings Play using JSONata.

By default, the generated JSON will have 3 instances of each element whose cardinality is greater than 1 including unbounded occurrences. For this example, some code can be tested by placing it on the “Request” panel and executing it by pressing the “Execute” button. The output of the request is shown in the “Results” panel.

# Appendix A. EDAM Projects Structure

To implement the EDAM an Application (App) folder needs to be identified in the “appsettings.json” configuration file. Within the “AppSettings” the “AssetConsolePath” key value will identify the full folder pathname that by default points to “…/Datovy.Edam/Edam.App.Data/”. After installing the App or code make sure to update this path to that folder location to quickly test the application using the “Communicable Diseases” (“Disease Surveillance”) EDAM Project”.

## A.1 EDAM App Data Templates and Projects

In the “…/Edam.App.Data/” folder the following application resources will be found:

* Arguments – Sample project arguments / parameters script may be found here.
* Documents – Sample output document files may be found here.
* Files – Sample input files may be found here.
* Projects – Keep all your projects here.
* Samples – Additional sample artifacts.
* Templates – Schemas for different DDL SQL variants (My SQL, MS SQL, and Oracle), and other artifacts are provided here.
* TextMaps – Sample text mapping configuration files like XSD types to SQL types are found here.
* Edam.Settings.json – Additional EDAM settings file.

## A.2 Projects Folder

The Project folder contains all projects folders each having a structure like A.1 including:

* Archive – A collection of artifacts that may be used while executing a process as specified by a script found in the Arguments folder.
* Arguments – Sample project arguments / parameters script may be found here.
* Documents – Sample output document files may be found here.
* Files – Sample input files may be found here.
* UseCases – The application uses this folder to store Use Case JSON files (see sample in the “Datovy.HC.CD/UseCase” folder”.

## A.3 Understanding the Project Process Arguments

EDAM was a command prompt tool only and a WinUI interface was built to ease working with the command Arguments list, therefore all processing must have one or multiple …Args.json (or arguments JSON) files. An arguments file will contain a definition of a “Process” that when executed will produce the requested artifacts. The structure of this file as prepared for the “Disease Surveillance” example follows:

{

"@context": {

"edam": "http://www.datovy.com/edam/arguments"

},

"Domain": {

"DomainId": "Datovy.HC.CD",

"Description": "Communicable Diseases"

},

"Namespace": {

"OrganizationDomainId": "datovy.hc.cd",

"Uri": "http://www.datovy.com/hc/cd",

"Prefix": "cd",

"Extension": "",

"RootElementName": "cd:Disease\_Surveillance\_Document"

},

"Project": {

"Name": "Communicable Diseases Database",

"VersionId": "v1r0"

},

"Process": {

"RecordId": null,

"Name": "Datovy.HC.CD.ToAssets",

"OrganizationId": "Datovy",

"OrganizationDomainUri": null,

"ProcedureName": "DdlImportToAssets",

"ProcedureTag": "DDL.DdlImportFileReader",

"SchemaType": 1,

"NextProcess": "",

"NextProcedure": [""]

},

"InputFile": {

"Extension": "xlsx",

"Name": "datovy.hc.cd.mdf",

"Path": "./Files",

"Full": null

},

"OutputFile": {

"Extension": "xlsx",

"Name": "datovy.hc.cd.dictionary",

"Path": "./Documents",

"Full": "./Documents/datovy.hc.cd.dictionary.xlsx"

},

"UriList": [

"./Archive/datovy.hc.cd.schema.xlsx"

],

"InspectArguments": {

"ListLength": "1",

"MaxThreshold": "1"

},

"ConnectionString": "",

"ElementTransform": null,

"TextMapFilePath": "./Archive/DdlTextMap.json"

}

Each section will be explained individually ahead.

Step 1: Establish the JSON vocabulary – context with the following:

"@context": {

"edam": "http://www.datovy.com/edam/arguments"

},

Always include the above at the top of the file.

Step 2: Define the Asset Domain

"Domain": {

"DomainId": "Datovy.HC.CD",

"Description": "Communicable Diseases"

},

EDAM will use this section to manage the (Assets) Domain Catalog. Once an Argument file is selected, the application looks for this section and try to find the “DomainId”, if not found will attempt to add it on the EDAM database.

Step 3: Define the Asset URI

"Namespace": {

"OrganizationDomainId": "datovy.hc.cd",

"Uri": "http://www.datovy.com/hc/cd",

"Prefix": "cd",

"Extension": "",

"RootElementName": "cd:Disease\_Surveillance\_Document"

},

Use this section to specify the Asset URI, its prefix, and “Root Element”. The “Root Element Name” is needed since some Assets don’t support an entry point in the schema that generally will contain many elements. The root can be thought as the schema node that represents a Use Case whose data component contain child relevant data entities for a particular scenario.

As soon as the process of this example is executed and as defined will produce a Root or Document element in the target language (XSD, JSON or other) if it may be relevant to the language. Note that the generated “root” document may not be valid or relevant, but every other schema artifact should be valid and represent the data entities needed to support the Asset definition in the target language.

Step 4: Define the Project

"Project": {

"Name": "Communicable Diseases Database",

"VersionId": "v1r0"

},

Detail the project name and version.

Step 5: Define the Process

"Process": {

"RecordId": null,

"Name": "Datovy.HC.CD.ToAssets",

"OrganizationId": "Datovy",

"OrganizationDomainUri": null,

"ProcedureName": "DdlImportToAssets",

"ProcedureTag": "DDL.DdlImportFileReader",

"SchemaType": 1,

"NextProcess": "",

"NextProcedure": [""]

},

Provide information of the above to define the process. Here the “ProcedureName” to be executed must be provided. Valid values for this name should match existing enumerator values as found in:

Edam.Data.AssetConsole.AssetConsoleProcedure *(file)*

Processes could be join using the “NextProcess” and “NextProcedure” but those will be documented in detail somewhere else.

Step 6. Define the Input and Output File (defaults)

"InputFile": {

"Extension": "xlsx",

"Name": "datovy.hc.cd.mdf",

"Path": "./Files",

"Full": null

},

"OutputFile": {

"Extension": "xlsx",

"Name": "datovy.hc.cd.dictionary",

"Path": "./Documents",

"Full": "./Documents/datovy.hc.cd.dictionary.xlsx"

},

If the WinUI console is used the above may not be relevant since options could be selected in the app.

Step 7. Define the Location of the Input File(s)

"UriList": [

"./Archive/datovy.hc.cd.schema.xlsx"

],

This is a list of paths and in the example relative to “./Archive” within the project folder. If multiple files are provided it will make a composition of all data elements found in all files. If the language supports the use of namespaces, such as an XSD, those will be brough in as defined and separate from others.

Step 8. Define How to Inspect Artifacts

"InspectArguments": {

"ListLength": "1",

"MaxThreshold": "1"

},

TBD

Step 9. Define Additional Arguments

"ConnectionString": "",

"ElementTransform": null,

"TextMapFilePath": "./Archive/DdlTextMap.json"

* If the procedure involves a database put the “ConnectionString” here.
* “ElementTransform” will be documented later.
* “TextMapFilePath” is used to provide text mappings from one language to another, or guide how to traverse through the Asset document structure. Samples will be given elsewhere.