EDAM Studio Understanding Projects

2023-02-05

Table of Contents

[1.0 Background 2](#_Toc128371680)

[2.0 EDAM Studio Projects Structure 2](#_Toc128371681)

[2.1 EDAM App Data Templates and Projects 2](#_Toc128371682)

[2.1.1 Adding Project Collections 3](#_Toc128371683)

[2.2 Projects Folder 3](#_Toc128371684)

[2.3 Understanding the Project Process Arguments 4](#_Toc128371685)

# 1.0 Background

At the beginning EDAM was a command line tool and the current version keeps all the resources to manage access to the same functionality but instead using a UI. Once the application is installed there will be an application folder called “Edam.Studio” located in the users “Documents” folder. Within the “Edam.App.Data” folder should be found.

# 2.0 EDAM Studio 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 “Documents/Edam.Studio/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”.

## 2.1 EDAM App Data Templates and Projects

Table

Description automatically generated

Figure 2.1 Installation “Edam.App.Data” folder layout.

In the “…/Edam.App.Data/” folder (see figure 2.1) 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 a project collection here (see 2.1.1 on how to add other collections).
* Samples – Additional sample artifacts.
* Temp – Directory in where resources may be available temporarily.
* 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.

### 2.1.1 Adding Project Collections

You may explicitly add new collections by inserting an entry in the “UriList” in the “Edam.Settings.json” file located in the default collection. The app needs to have this default collection to properly operate. Each project URI information includes:

|  |  |
| --- | --- |
| Name | The name of the collection that should match with the name of its related folder. |
| Type | 1 = Console Path (always use this value) |
| UriText | A file Path. (Eventually the URI may point to some Web/Cloud resource) |

(Proposed) Implicitly, any folder below the “…/Edam.Studio” folder that ends with “.App.Data” will be consider a collection.

## 2.2 Projects Folder

Graphical user interface, application, Word

Description automatically generated

Figure 2.2 Installation “Datovy.HC.CD” Projects Folder.

Graphical user interface, text, application

Description automatically generated

Figure 2.2.1 Datovy.HC.CD Folder layout.

The Project folder (see Figure 2.2, and 2.2.1) 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”.

## 2.3 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": "Datovy.HC.CD",

"VersionId": "v1r0"

},

Detail the project name and version. The name must be the same as the Project folder.

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.

The current version will not switch to the Organization ID of the project but there are plans to do so.

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.