

# Training documentation for Geoscience Australia for the Digital Atlas Project

## 1. ETL processing

### 1.1. RDF generation

Python based row processing utilising python pandas, SQLAlchemy to connect to postgres. Python's pandas library provides a good wrapper for ingestion of data from a range of relational sources, including Postgres databases as currently used at GA. It is a widely used well supported library. Once read in to a pandas Dataframe (an in memory table), the data may be iterated over row by row with any data manipulation performed using Python functions; for example built in string functions, and geospatial conversion functions through third party libraries such as Shapely. This provides a high degree of flexibility for any necessary data manipulation, including mapping of codes, logic, spelling corrections etc. which can be difficult or expensive operations with other methods. The Python code is readable for new users, and can easily be updated to suit any new requirements, or fix bugs.

RDFLib is utilised to create valid RDF from each row of data, this ensures no syntax errors are introduced into the RDF graph. The data is serialised using RDFLib, to a set of RDF files. The exception to this is the GeoFabric and GNAF datasets, for which record by record processing is done from database dumps. This is due to the size of these datasets, which could not be loaded in to memory on most desktop computers.

The ETL scripts are collected in the <digital atlas etl repo>

#### 1.1.1. Local development, testing

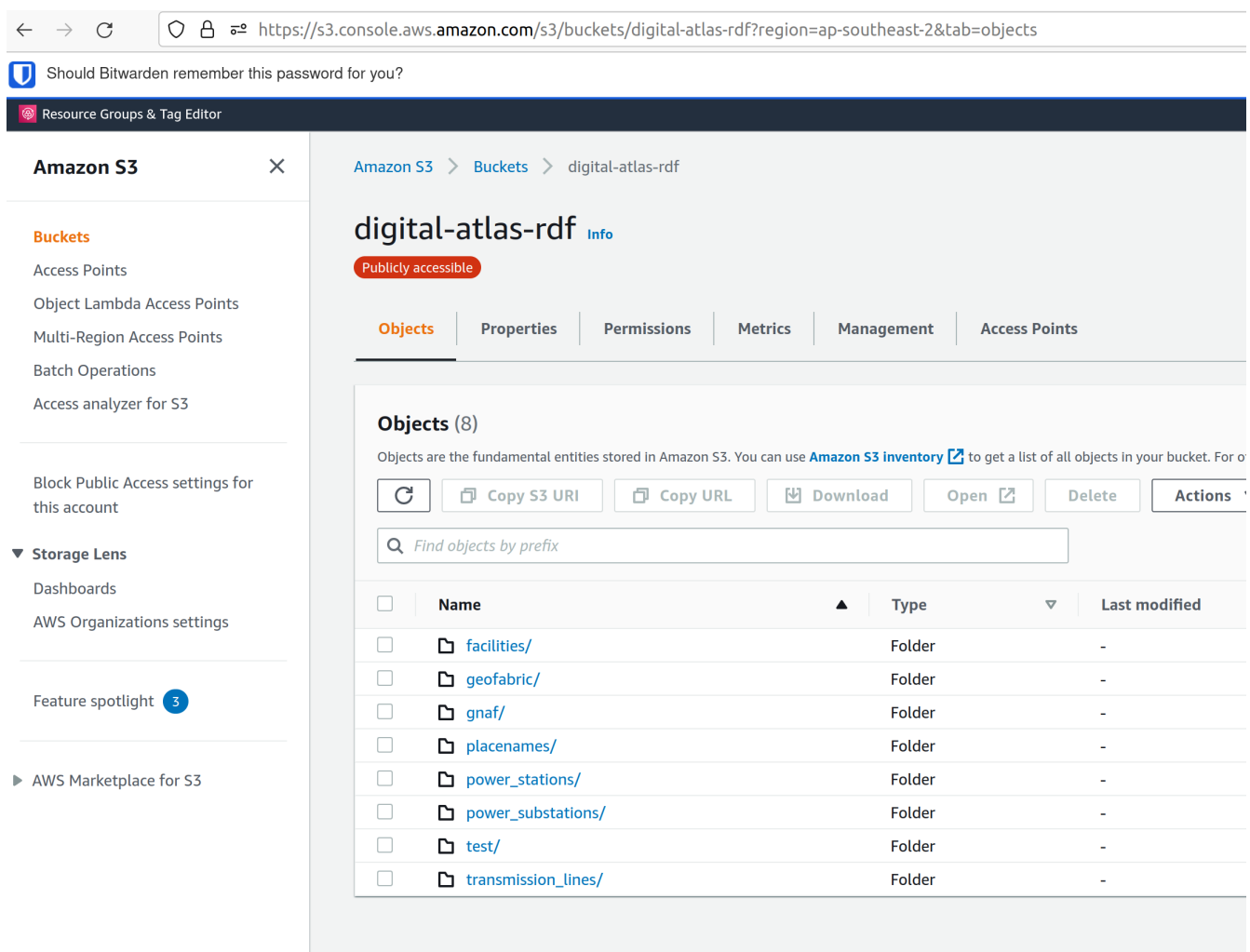
To make updates to the ETL scripts, the python scripts in `app/processing/{dataset}_block_convert_to_rdf.py` within the [Digital Atlas ETL repo](#) should be updated. The changes can then be tested locally by running the python code directly (option 1 below), and inspecting the output. Local integration testing can be performed using Docker Compose, both for the ETL process, and to confirm the output displays correctly in Prez.

Options:

1. Run python script directly, either through the terminal, or an IDE run configuration.
2. Docker container - allows testing of the processing code only
3. Docker Compose - allows testing of the processing code, loading it to Jena Fuseki, and displaying it through Prez. This facilitates rapid iteration over the processing code, allowing issues to be identified and resolved early in the development process. *Detailed instructions for these methods of running the ETL process and testing are described in the repository's [Readme](#).*

## 1.1.2. Cloud based processing

The ETL Process utilises ECS and EFS when deployed in AWS via Terraform, see [AWS infrastructure](#) and [Infrastructure management](#) below. The Terraform script creates a Task Definition for each dataset to be processed, which can be run manually through the AWS console by navigating to "ECS", and finding the relevant task under "Task Definitions". The tasks are configured to write to the S3 Bucket specified using the "OUTPUT\_LOCATION" environment variable. This environment variable is currently set in the task definition JSON template (used for all processing tasks), to `s3://digital-atlas-rdf`. The ECS task has permission to write to this bucket through a statement attached to the `ecs_task_definition_role_policy_document` data definition in `data.tf`. The processing code will prepend the `DATASET` environment variable to the key of each object stored in this bucket, effectively creating a 'folder' structure when viewed through the AWS Console. This structure also allows easy pattern matching when attempting to read specific datasets from S3 to process in to TDB2 datasets, as detailed below. The current contents of the bucket are shown below, with all of the RDF data that has been processed by Surround currently in the bucket:



The screenshot shows the Amazon S3 console interface for the bucket 'digital-atlas-rdf'. The bucket is publicly accessible. The 'Objects' tab is selected, showing a list of 8 objects, all of which are folders. The folders are: facilities/, geofabric/, gnaf/, placenames/, power\_stations/, power\_substations/, test/, and transmission\_lines/.

<input type="checkbox"/>	Name	Type	Last modified
<input type="checkbox"/>	facilities/	Folder	-
<input type="checkbox"/>	geofabric/	Folder	-
<input type="checkbox"/>	gnaf/	Folder	-
<input type="checkbox"/>	placenames/	Folder	-
<input type="checkbox"/>	power_stations/	Folder	-
<input type="checkbox"/>	power_substations/	Folder	-
<input type="checkbox"/>	test/	Folder	-
<input type="checkbox"/>	transmission_lines/	Folder	-

## 1.2. TDB2 generation and spatial indexing

TDB2 is the second generation on-disk database format utilised by Apache Jena. It provides high scalability, indexation, and works in conjunction with extensions including GeoSPARQL and Full Text Search. TDB2 datasets can be created directly, by utilising the `tdb2.tdbloader` command line utility packaged with Apache Jena; through the Fuseki UI; or through Assembler files, which are parsed on Jena startup.

### 1.2.1. Local development, testing

To test "end to end" ETL processing, in a similar manner to as it would occur on AWS, it is recommended to utilise the Docker Compose scripts, as the outputs of one container process can be reused in another. This allows the outputs of the RDF generation steps to be loaded (via a common volume) to the TDB2 generation scripts in a dependent docker container. NB the ETL compose file utilise a **Jena** image, which includes the **tdb2.tdbloader** command line utility, but does not include the Fuseki UI.

Should you wish to test RDF that has already been generated, or is from some source other than the RDF generation scripts, you can directly run the **Jena Fuseki** container image and create a TDB2 dataset through the Fuseki UI, by going to "manage", then "new dataset". NB This TDB2 dataset will not persist beyond the life of the docker container, unless an external mount or volume has been configured, as docker containers are ephemeral.

The Jena spatial index is created through a jar file, repackaged from Apache Jena by Zazuko. This has been added to a multi-stage docker container by Surround, so as to include TDB2 generation along with spatial indexation in one image. This was done as for spatial datasets, the two processes both need to occur, and the output of the TDB2 generation (a TDB2 dataset) is the input of the spatial indexer.

In addition to the TDB2 generation and spatial indexation, a prior step is to validate the input data is valid RDF. For clarity, this validation is to determine whether the files are valid RDF, not whether they conform to any profiles (e.g. the OGC Linked Data API profile), meaning the validation will only provide a guarantee that a TDB2 dataset can be created from them, not that the data itself conforms to any useful model. The validation is completed utilising the Apache Jena RIOT command line utility. Surround has written a short bash script which calls the validator on all nquads files which are to be loaded, and should any of them contain any invalid RDF, prints out the problematic lines and errors, and renames the files with an extension of **.error**. This renaming prevents **tdb2.tdbloader** attempting to load them to a TDB2 dataset. Successfully validated files will also write a short message confirming their valid format. Should a file fail validation, you should try to identify the problematic code which generated invalid RDF, fix this, and the invalid RDF files (or parts of them, and apply a patch). The TDB2 dataset can either be appended to by re-running processing on just the problematic files, or the whole TDB2 dataset can be regenerated if this is easier.

### 1.2.2. Cloud based processing

Task definitions have been created to allow running the TDB2 generation on AWS. This is useful (and arguably necessary) for larger datasets. A task has been created for each TDB2 dataset, which reads data from the relevant RDF Datasets (as RDF files in the output RDF bucket), and writes the output TDB2 dataset to a mounted EFS volume. These are:

TDB2 Dataset	RDF Datasets (dcat:Dataset)
cache	power_substations, power_stations, transmission_lines, facilities, placenames, gnaf, asgs, geofabric
fsdf	power_substations, power_stations, transmission_lines, facilities, placenames

TDB2 Dataset	RDF Datasets (dcat:Dataset)
gnaf	gnaf
asgs	asgs
geofabric	geofabric

To run the TDB2 generation task on AWS, the task must specify and EFS volume that can be mounted. Either an existing TDB2 dataset can be used, or a new one must be created. These scenarios are described in detail below:

1. There is an existing TDB2 dataset on an existing EFS volume which you would like to add additional data to. You will need to:
  - Ensure any Jena Fuseki instances currently using the volume are stopped. You can do this by setting the "Desired tasks" count to zero on the relevant "Service" in the **ld-digitalatlas-nonprod** cluster on AWS; and
  - Remove the network mount points for the EFS volume. Go to the EFS homepage in AWS, select the relevant EFS volume (as it already exists, it should be mapped to the relevant TDB2 dataset in the API Terraform) navigate to network, click edit, and remove the mount points.
2. A new TDB2 dataset is required, requiring an EFS volume to be created. You will need to:
  - Go in to the AWS console (or through the CLI or otherwise) and create an EFS volume in the appropriate region (ap-southeast-2); and
  - AWS will automatically generate network mount points, delete these in the console so terraform can create its own network mounts in the right subnets for the task to use. To do this, navigate to network, click edit, and remove the mount points.

At this point, you should have an EFS volume listed under the EFS File Systems page, as in the screenshot below:

Name	File system ID	Encrypte d	Total size	Size in Standard / One Zone	Size in Standard-IA / One Zone-IA	Provisioned Throughput (MiB/s)	File system state	Creation time	Av ty
da_gnaf_v3	fs-031a24d42e66d2673	Encrypted	226.16 GiB	226.16 GiB	0 Bytes	-	Available	Mon, 20 Jun 2022 05:12:37 GMT	Re
da_gnaf_v2	fs-0e2e227caa2a4527d	Encrypted	24.12 GiB	24.12 GiB	0 Bytes	-	Available	Sun, 19 Jun 2022 12:38:26 GMT	Re
da_geofabric_v2	fs-0cd2dc084a3a06eb8	Encrypted	4.40 GiB	4.40 GiB	0 Bytes	-	Available	Sun, 19 Jun 2022 06:21:44 GMT	Re
DigitalAtlas_Geofabric	fs-0696030a59e1faa1e	Encrypted	318.00 KiB	318.00 KiB	0 Bytes	-	Available	Thu, 16 Jun 2022 07:10:50 GMT	Re
DigitalAtlas_ASgs	fs-06b4f9f34dfd2b3e2	Encrypted	6.00 KiB	6.00 KiB	0 Bytes	-	Available	Thu, 16 Jun 2022 07:10:30 GMT	Re
DigitalAtlas_FSDf	fs-02e81c662e2b7e840	Encrypted	6.00 KiB	6.00 KiB	0 Bytes	-	Available	Thu, 16 Jun 2022 07:10:23 GMT	Re
DigitalAtlas_Cache	fs-064e637727633fd9c	Encrypted	6.00 KiB	6.00 KiB	0 Bytes	-	Available	Thu, 16 Jun 2022 07:09:56 GMT	Re
PID_TDB	fs-055e3a1efeeae0a8b0	Encrypted	10.51 MiB	10.51 MiB	0 Bytes	-	Available	Tue, 14 Jun 2022 09:39:49 GMT	Re
DA_asgs	fs-07f2e27878f7f3fb4	Encrypted	4.10 GiB	4.10 GiB	0 Bytes	-	Available	Thu, 02 Jun 2022 01:50:12 GMT	Re
DA_geofabric	fs-0bff4dfae71c2d7a	Encrypted	188.70 MiB	188.65 MiB	48.00 KiB	-	Available	Fri, 27 May 2022 04:56:24 GMT	Re
DA_saml_datasets	fs-0a72a975e1e8fd789	Encrypted	318.00 KiB	222.00 KiB	96.00 KiB	-	Available	Fri, 27 May 2022 04:55:43 GMT	Re

Specify the EFS volume in the da-etl-terraform repository's `terraform.tfvars` file, under `efs_id`. This will set the task up to utilise the appropriate volume when deployed. Run Terraform apply, and wait for the changes to propagate.

Log in to the AWS console, navigate to EFS, select the relevant EFS volume, and make a note of the Subnet IDs the EFS mount targets are in (under the Network tab), as shown in the screenshot below:

Amazon EFS > File systems > fs-031a24d42e66d2673

## da\_gnaf\_v3 (fs-031a24d42e66d2673)

**General**

Performance mode  
General Purpose

Throughput mode  
Bursting

Lifecycle management  
Transition into IA: 30 days since last access  
Transition out of IA: On first access

Availability zone  
Regional

Automatic backups  
✔ Enabled

Encrypted  
3e077ebf-45d4-423b-b294-319c

File system state  
✔ Available

DNS name  
fs-031a24d42e66d2673.efs.i

Metered size

Monitoring

Tags

File system policy

Access points

**Network**

Replication

**Network**

Availability zone ▲	Mount target ID ▼	Subnet ID ▼	Mount target state ▼	IP address
ap-southeast-2a	fsmt-0eb3a8e8685f52014	subnet-090072efff3ce1ef6	✔ Available	175.32.9.24
ap-southeast-2b	fsmt-039a71223eeacbe65	subnet-0b7a436f637796ad4	✔ Available	175.32.10.181
ap-southeast-2c	fsmt-0b0673bb82a76bf0d	subnet-002f615541daf76d2	✔ Available	175.32.11.175

The task can now be run and will mount the correct volume, as specified in Terraform. To run the task, navigate to ECS in the AWS console, click on "Task Definitions", find the relevant task (they are all prepended with "tdb2\_generation\_\*"), select "Deploy" and then "Run task".

← → ↻ 🔒 https://ap-southeast-2.console.aws.amazon.com/ecs/v2/clusters/da-etl-nonprod/ 90% 🔍 Search

aws Services 🔍 Search for services, features, blogs, docs, and more [Alt+S] 📧 🔔 ⓘ Sydney surround\_David @ 0496-4885-

Resource Groups & Tag Editor

New ECS Experience Tell us what you think ✕

**Amazon Elastic Container Service**

Clusters Task definitions Account settings

Amazon ECR Repositories

Documentation 🔗 Discover products 🔗 Subscriptions 🔗

Amazon Elastic Container Service > Clusters > da-etl-nonprod > Tasks

da-etl-nonprod 🔄 Edit cluster Delete cluster

📄 arn:aws:ecs:ap-southeast-2:049648851863:cluster/da-etl-nonprod

**Cluster overview**

Services		Tasks	
Draining	Active	Pending	Running
-	-	-	-

Services **Tasks** Infrastructure Metrics Tags

**Tasks (0)** 🔄 Stop ▼ Run new task

🔍 Filter tasks by property or value < 1 > ⚙️

Task	Last status	Task definition	Revision	Health status	Started at
No tasks No tasks to display. <span>Run new task</span>					

Select the relevant ECS Cluster (currently figured as **da-etl-nonprod**), and select only the relevant subnets that you made a note of above, as shown in the screenshot below:

## ▼ Networking

### VPC [Info](#)

Choose the Virtual Private Cloud to use.

vpc-0eb799c7f639c8da4  
VPC

### Subnets

Choose the subnets within the VPC that the task scheduler should consider for placement.

Choose subnets

subnet-090072efff3ce1ef6 X  
ap-southeast-2a | Private Subnet

subnet-0b7a436f637796ad4 X  
ap-southeast-2b | Private Subnet

subnet-002f615541daf76d2 X  
ap-southeast-2c | Private Subnet

### Security group [Info](#)

Choose an existing security group or create a new security group.

☒ Use an existing security group

☐ Create a new security group

#### Security group name

Choose an existing security group.

sg-013d3d54301fe6bdc X  
default | SG-default

### Public IP [Info](#)

Choose whether to auto-assign a public IP to the task's elastic network interface (ENI).

☒ Enabled

To view the status of the task, click on the task ID, and then click on "logs". It can take a minute for the task to be registered/deployed. Once the task has finished, logs will not be available for viewing under the ECS pages, however logs have been configured to be sent to cloudwatch logs, so they can be viewed here until the retention period ends, currently set to 1 day.

## 1.3. Display of linked data through a web interface

### 1.3.1. Web application

The linked data API used for display is a Surround built application, Prez, which is based on FastAPI. FastAPI is an asynchronous web framework written in Python. It suits the task of a linked data API well as regular Python code can be used directly in the functions which generate pages, allowing 'on the fly' manipulation of RDF for presentation in different formats, including HTML for the web. Python's RDFLib library is used for any manipulation of RDF prior to display. Data is retrieved from a backend SPARQL endpoint, which in the case of the systems implemented for Geoscience Australia is provided by a Jena Fuseki instance. Prez is a proprietary product that is licensed under [BSD-3](#).

The data used by Prez, while all RDF, can be classified in to three categories:

1. Instance data. This is the output of the ETL processing, and includes data at the Feature, Feature Collection, and Dataset level.
2. Context data. This includes ontologies for the dataset(s), and any related ontologies needed to understand the data, for example the GeoSPARQL ontology. Prez only requires labels and descriptions from these ontologies; though sometimes the entire ontologies are included, as they are small in size and can provide useful context for direct queries to the SPARQL endpoint.

Prez is available as a [docker image on dockerhub](#), and the source code is available on [Github](#).

### 1.3.2. Triplestore

- The triplestore used is a combination of three open source Apache Jena related technologies:
  1. Jena (Java triplestore)
  2. TDB2 (Persistent store for Jena)
  3. Fuseki (Webserver providing UI and SPARQL endpoints)
- Public docker images for Jena and Jena with Fuseki have been created by a Jena user, Stain, and are available on Docker Hub
- The Jena Fuseki image includes both these components (and the ability to work with TDB2)
- The Jena image includes a set of TDB2 command line utilities, which can be used to load RDF data to TDB2, and then query/update/delete directly in TDB2. This is the preferred approach for creating large datasets, or performing updates across large numbers of triples.

## 1.4. Source code management

### 1.4.1. Processing code (source data to RDF)

The ETL processing code, described in [RDF generation](#), is available in the [digital-atlas-etl](#) repository on GA's BitBucket. This repository includes a bitbucket pipeline script which automatically builds and pushes a docker image to the [digital-atlas/etl](#) repository in GA's AWS Account.

### 1.4.2. TDB2 generation

A multi-stage build docker image is used to generate TDB2 datasets, including a spatial index, as described in [TDB2 generation and spatial indexing](#). This dockerfile is automatically built in a bitbucket pipeline, and the resulting image pushed to the [tdb-generation](#) repository in GA's AWS Account.

### 1.4.3. Prez

Surround Australia maintains Prez through a [github repository](#). If there are issues with Prez, or feature suggestions, it is recommended to add these as issues to the issue tracker at <https://github.com/surroundaustralia/Prez/issues>. A suite of unit tests are maintained, including tests for the different endpoints. A docker image is built for each git tagged version (built with tag



'latest'), and versioned images are built when a release is made in Github. We recommend pinning the production version of Prez you use to a particular version, and performing user acceptance testing on any upgrades to Prez prior to utilising these in production.

At present, the themes (CSS) and static template pages (jinja templates) made specifically for GA are added in to the base Prez docker image provided by Surround, to produce a "themed" version of Prez. This theming represents a minimal Docker script, available in the [ga-themed-prez](#) repository. *This repository has not yet had the bitbucket pipeline enabled as Surround doesn't currently have permission to do so*

In a future release, it is planned to enable dynamic loading of themes at API startup, configured through an environment variable. This will remove the need to build/maintain a separate themed docker image.

#### 1.4.4. Jena (TDB2 generation, spatial indexation, RDF validation)

Surround has created a docker image which combines the downloading of RDF files from an S3 bucket, RDF validation, TDB2 generation, and Spatial Indexation. This image is in the [TDB Generation](#) repository. *This repository has not yet had the bitbucket pipeline enabled as Surround doesn't currently have permission to do so*

#### 1.4.5. Jena Fuseki

Jena Fuseki includes both Apache Jena, the triplestore, and Fuseki, the frontend web framework including SPARQL endpoint functionality. The version of Jena Fuseki used includes the GeoSPARQL extension. The image used is based on one provided at [Zazuko's Fuseki GeoSPARQL Github repository](#). This image simply downloads the relevant Jena binaries and adds them to a docker image. To this image Surround adds a Jena dataset config, and push the resulting image to the [fuseki ECR repository](#).

## 1.5. Continuous integration and continuous deployment

- Bitbucket pipelines for application packaging as docker images, and pushing to container registries.
- Updates to application processing and API code will be automatically built in to new images and **available** for deployment, however manual deployment is required, in order to facilitate User Acceptance testing prior to deployment. Learning Resources: bitbucket-pipelines.yml file at <https://bitbucket.org/geoscienceaustralia/digital-atlas-etl/addon/pipelines/home>

## 1.6. AWS infrastructure

A brief description of the AWS resources ECS services and ECS tasks are used to run the docker containers.

- Elastic Container Service (ECS) - Runs the docker containers as services (for Prez and Jena) or as one off jobs (ETL processing).

- Elastic File System (EFS) - provides persistent storage for the docker containers running on ECS.
- Elastic Load Balancer (ELB) - provides load balancing and traffic direction for the ECS services.
- Elastic Container Registry (ECR) - provides a registry for the docker images used by the ECS services.
- Simple Storage Service (S3) - For storage of RDF and raw data. Learning resources:
- [AWS documentation](#)

## 1.7. Infrastructure management

Terraform was used as this is the GA infrastructure management tool. This contains: - AWS infrastructure (ECS, EFS, ELB) - Definitions of the docker images used by the ECS services - AWS security groups and rules

# 2. Modelling, Ontologies, Dataset and Feature Collection definitions

## 2.1. Ontologies and associated modelling

Modelling related to FSDF is in the GA Supermodel, as documented here.

## 2.2. Feature collection and Dataset definitions

Feature collection and dataset definitions are available in the <DA-ETL-Processing repo>