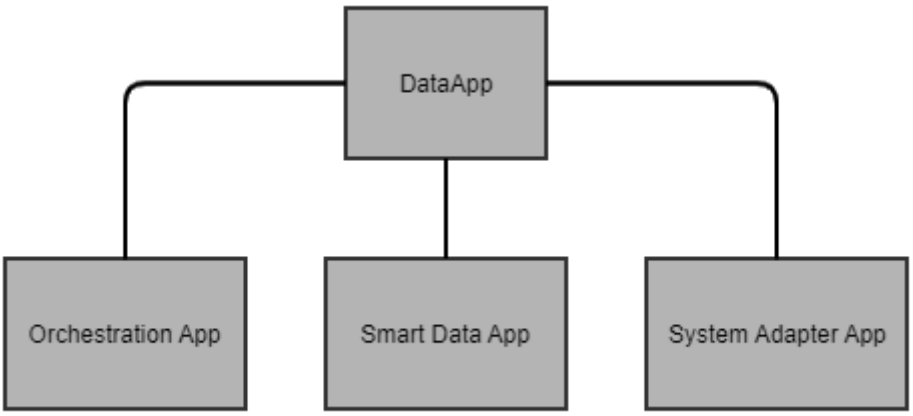
App Development Guide

An App in the IDS ecosystem mainly consists of two components. First, an application needs to be provided as a packaged linux virtualization container (e.g., Docker or Kubernetes) or any other kind of application deployment. Therefore, an app can also be a packaged as an executable or a technology dependent deployment format such as a java jar archive for example. The second part of an app is a metadata description of the application itself, that conforms to the IDS information model. This application metadata is used later for the application deployment inside of IDS connectors and for the interaction with the IDS appstore. The app itself can provide some endpoints for consuming, providing and processing data. In addition to these endpoints, an app can provide interfaces for configuration. If data is processed within an app even endpoints for parameterization of methods can be provided by an app.

App Categories

Within the IDS ecosystem there are three different categories of applications by now, each performing different tasks in the application ecosystem. The following section will give a short overview of the different categories and will explain them in more detail.



System Adapter App (Data Endpoints - Data Source Interfaces / Data Sink Interfaces)

The applications in the system adapter category provide interfaces for reading or writing data from external data sources or external data sinks. System adapter apps can be applied to all types of data sources or data sinks, including enterprise information systems.

System adapter apps can be used to integrate web services, databases, flat files, or even big data systems, just to name a few possible examples.

Another important task of a system adapter app is to transform data models from the data sources into a recommended or standardized data format. A system adapter app can also enrich the data with additional metadata end offer it to other apps.

Smart Data App (Data Processing)

The applications belonging to the smart data category, so called smart data apps, are responsible for any kind of data manipulation or processing. These applications provide functions for data processing or data transformation tasks. Therefore, apps for data quality assurance are described as smart data apps, as well as for example apps for machine learning algorithms. The data a smart data app consumes and the data it provides after processing / transformation is normally already enriched with metadata.

Orchestration App

The different app types can be orchestrated, which makes it possible to build a data processing chain with several apps chained together. The orchestration can be done by the IDS connector itself, in whose runtime environment the apps have been installed. The control over the orchestration is hereby enforced with the help of the configuration model inside the connector.

Another possibility to orchestrate applications is the use of an so called orchestration app, which can also take over this task.

Within the orchestration app the different endpoints of the system adapter apps and the smart data apps can be controlled and connected to each other. Like the orchestration in the IDS connector, routing is realized via camel routes to control the individual endpoints and data flows in the orchestration app.

App Interaction Endpoints

As described above, there are also different endpoints for the apps from the three categories, which should be implemented by an application to guarantee their functionality. Input or output endpoints must be integrated by apps from the system adapter category to connect external data sources and / or data sinks.

Other endpoints that can be implemented by all app categories are the endpoints for querying the status of an app and the endpoint for implementing application specific usage control policies. The status query can provide general information about the current state of the app, including lifecycle or internal connections (database connection, network information).

The endpoints for configuration / parameterization and processing are mainly intended for use in smart data apps. They can be used to pass parameters to an app which are necessary for processing and to finally start the processing itself.

IDS_ENDPOINT_TYPE	App Type	Example Mapping	Metadata Mapping	Description
CONFIG_ENDPOINT	Smart data	/configPath	Port and path mapping as in application code	Endpoint for configuration / parameterization
INPUT_ENDPOINT	Smart data System adapter	/inputPath	Port and path mapping as in application code	Endpoint for consuming data
OUTPUT_ENDPOINT	Smart data System adapter	/outputPath	Port and path mapping as in application code	Endpoint for providing (processed) data
PROCESS_ENDPOINT	Smart data	/processPath	Port and path mapping as in application code	Endpoint to start the data processing
STATUS_ENDPOINT	Smart data System adapter	/statusPath	Port and path mapping as in application code	Endpoint for status information (lifecycle)
USAGE_ENDPOINT	Smart data System adapter	/ids/usage	Path must match the /ids/usage mapping	Endpoint for application internal usage-control limitations

IDS_ENDPOINT	Smart data System adapter Orchestration	/ids	Path must match the /ids mapping	Endpoint to return the ids metadata as a kind of self-description about the app

App Metadata Representation

As already mentioned in the beginning, the app metadata representation is an important part of an in developing an IDS data app. The metadata representation is following the IDS information model version 4.0.2 and above. In the following the different sections of the information model representation will be explained in more detail.

General app information fields

- title → The app title
- version → The application release version number. For simplicity the version number can match the container release number.
- description → The application short description
- keyword → Keywords describing app categories
- created → App creation date
- modified → App modification date
- customLicense → URI to custom license information
- standardLicense → URI to standard license information

```
"ids:title": [
    {
      "@value": "IDSSmartDataAppTemplate",
      "@language": "https://w3id.org/idsa/code/EN"
    }
],
"ids:version": "1.0.0",
"ids:description": [
    {
      "@value": "Example Smart-Data-App for demonstration purposes",
      "@language": "https://w3id.org/idsa/code/EN"
    }
  ],
"ids:keyword": [
    {
      "@value": "Demo",
      "@language": "https://w3id.org/idsa/code/EN"
    },
    {
      "@value": "Example",
      "@language": "https://w3id.org/idsa/code/EN"
    },
    {
      "@value": "Smart-Data-App",
      "@language": "https://w3id.org/idsa/code/EN"
    }
  ],
"ids:created": "2020-11-11T17:42:50.609UTC",
"ids:modified": "2020-11-11T17:42:50.612UTC",
"ids:standardLicense": "https://www.apache.org/licenses/LICENSE-2.0",
"ids:customLicense": "https://www.apache.org/licenses/LICENSE-2.0"
```

Definition of an app “Representation” within the metadata representation

An IDS data app consists at least of one app representation which in the first place describes the runtime environment. If the app is a virtualization container the “dataAppRuntimeEnvironment” should be used to set the virtualization environment here (docker, kn8s). In Addition to this, the “dataAppDistributionService” could be set. This field indicates where the related container is stored. (e.g., if you are using a private or remote container registry)

- DataAppRuntimeEnvironment → The application runtime environment
- dataAppDistributionService → The appstore url or the registry where the app is located

```
"ids:dataAppRuntimeEnvironment": "Docker",
"ids:dataAppDistributionService":
"https://appstore.fit.fraunhofer.de/registry"
```

Definition of “DataAppInformation” within the metadata representation

In this section properties related to the app deployment and the app documentation can be defined. If there is a human readable documentation about the application available, it can be referenced by the “appDocumentation” property. If the application needs some special variables set for configuration purposes, these can be set by using the “appEnvironmentVariables”. The given environment variables should match the ones that are needed to configure the linux container, so if you are using docker for example the environment variables should match the ones defined in the docker file. For the “appStorageConfiguration” it is pretty much the same. If the application needs some volumes to be mapped, these volumes should be given in the “appStorageConfiguration” as well as in the container definition.

If the app needs some environment variables or volume mountings for execution, these can be set within the following fields.

- appDocumentation → The application documentation in human readable text
- appEnvironmentVariables → Environment variables needed by the app (configuration or settings)
- appStorageConfiguration → Volumes to mount for the application execution

```
"ids:appDocumentation": "Place for an app-related human-readable
documentation",
 "ids:appEnvironmentVariables": "dbUser=sa;dbPasswd=passwd",
 "ids:appStorageConfiguration": "-v /data"
```

Definition of “AppEndpoints” within the metadata representation

For defining an app endpoint, it is important to notice that the path mapping, the listening port and the media type need to match to your current app implementation. The ids endpoint is special because it delivers the app metadata in json-ld format as a self-description of the app. To offer the metadata representation can be used for example to realize usage control between apps (purpose and identification) and is therefore mandatory for every app implementation.

- path → The path where the app endpoint is mapped to
- appEndpointProtocol → The protocol the endpoint is listening on
- endpointInformation → Information about the endpoint in human readable text
- endpointDocumentation → Uri to an external endpoint documentation
- appEndpointMediaType → IANAMediatype an endpoint sends or receives
- appEndpointPort → The port number the endpoint listens for connections
- appEndpointType → Descripes wether the endpoint is an INPUT, OUTPUT, CONFIG, PROCESS, STATUS, USAGE_POLICY, IDS endpoint

```
"ids:path": "/input",
"ids:appEndpointProtocol": "HTTP/1.1",
"ids:endpointInformation": [
    {
        "@value": "Endpoint for app input data",
        "@language": "https://w3id.org/idsa/code/EN"
    }
],
"ids:endpointDocumentation": [
    "https://app.swaggerhub.com/apis/app/1337"
],
"ids:appEndpointMediaType": {
    "@type": "ids:IANAMediaType",
    "@id": "https://w3id.org/idsa/autogen/iANAMediaType/27f8fdd6-d214-46c0-
82c2-46d56f5ac464",
    "ids:filenameExtension": "application/json"
},
"ids:appEndpointPort": 8080,
"ids:appEndpointType": {
    "@id": "idsc:INPUT_ENDPOINT"
}
```

Virtualization Container Generation

To be able to publish the data apps, they need to be packaged into a linux virtualization container.

In this example we use docker container as the virtualization technology. The environment variables, port shares and locations required by the application for operation are specified in the dockerfile according to the Docker documentation.

However, it is important to make sure that the information published in the dockerfile is also specified in the app metadata, because the later deployment of an app in a connector depends on the information in the metadata.

To ensure that container specific information is also given in the metdata representation it is worthwhile to have a look at the app representations (Chapter X), the app endpoints (Chapter X) and the following docker file example, which illustrates the relationship between metadata and docker properties. The properties of environment variables and storage configurations must be considered here, as well as the endpoint specifications via the port mappings. The last but necessary property is the “IDS-METADATA” named label. The value of this label is the base64 encoded version of the metadata representation.

The specification of the parameters is therefore subject to a higher priority. When deploying the app via the AppStore, an application specific container template is generated from the given metadata. This template is then used to integrate apps into some container management software (e.g., Portainer, Yacht) within the connectors.

The following example of a dockerfile shows the relation between the dockerfile properties and the properties set in the app metadata representation.

```
# EVIRONMENT VARIABLES MAPPING FOR IDS DATA APP
# Environment variables can be used in the app metadata under the following
property path
# "ids:representation" --> "ids:appEnvironmentVariables":
"$dbUser=sa;$dpPasswd=passwd",
ENV dbUser="sa"
ENV dbPasswd="passwd"

# VOLUME MAPPING FOR IDS DATA APP
# Volumes or files can be used in the app metadata under the following
property path
# "ids:representation" --> "ids:appStorageConfiguration": "-v /data",
# "ids:representation" --> "ids:appStorageConfiguration": "-v
/data/file.tmp",
VOLUME ["/data"]

# PORT MAPPING FOR IDS DATA APP ENDPOINTS
# Ports can be used in the app metadata under the following property path
# Each endpoint has its own port number, so multiple ports per app are
possible
# Each port can be exposed in docker or mapped to a custom host port
# "ids:representation" --> "ids:appEndpoint" --> "ids:appEndpointPort":
"8080"
EXPOSE 8080/tcp

# IDS APP DESCRIPTION ENCODED AS BASE64 ENCODED STRING REPRESENTATION
# The app metadata will be published as an encoded version with the help of
a Docker label named "IDS-Metadata"
# In this app template, the encoded version can be found because of the
/ids/encode endpoint
LABEL maintainer="Dominic Reuter <dominic.reuter@fit.fraunhofer.de>"
LABEL IDS-METADATA="eyJAdHlwZSI6ImlkczpBcHBSZXNvdXJjZSIsIk……."
```