H2020-INFRADEV-2019-2
Grant Agreement No 871043

DiSSCo Prepare WP D6.3

A generalised set of API specifications for interaction with the DiSSCo core architecture

Work package lead: Claus Weiland

Version 1.0 [20.01.2023]

Authors:

Sam Leeflang	0000-0002-5669-2769	Naturalis Biodiversity Center
Claus Weiland	0000-0003-0351-6523	Senckenberg Nature Research Society
Sharif Islam	0000-0001-8050-0299	Naturalis Biodiversity Center
Matthias Dillen	0000-0002-3973-1252	Meise Botanic Garden
Soulaine Theocharides	0000-0001-7573-4330	Naturalis Biodiversity Center

Abstract

Application Programming Interfaces (APIs) form the glue between all the different DiSSCo services. They are the main vehicle for data exchange between machines. Additionally, users may want to request or build their own applications on top of the APIs. It is therefore essential to the success of DiSSCo that it provides stable, simple and well documented APIs. This deliverable describes the strategy through which DiSSCo aims to achieve this goal.

DiSSCo will leverage established industry standards as much as possible. Adhering to generic specifications such as the JSON:API specification helps to build predictable and reliable APIs. DiSSCo recognizes that different types of users and software agents might require different types of endpoints. Hence, in addition to simple JSON response, we will also provide the option for users to request JSON-LD. DiSSCo might also implement an asynchronous API for larger datasets, where we will notify the user when their request is ready to be collected.

Documentation will leverage the power of OpenAPI v3 in combination with a Swagger endpoint for human readability. FDO Types and openDS term descriptions will provide documentation on the data model both for machines and for users.

APIs are entry points into DiSSCo's system and therefore prime points of attacks. To mitigate these attacks, DiSSCo will implement up-to-date security following the latest guidelines. While the data in DiSSCo will be as open as possible, we will store non-public information. This information will only be accessible for authenticated and authorized users. For the implementation, DiSSCo will follow the guidelines of OAuth2 in combination with OIDC.

By building on top of existing standards DiSSCo hopes to provide predictable and stable APIs. These APIs will be the building blocks used to build DiSSCo's services providing services for our end users.

Keywords

FAIR Digital Object, Distributed System of Scientific Collections, DiSSCo RI, openDS, Digital Specimen, Application Programming Interface, JSON:API, JSON-LD, DOIP, data exchange, security.

INDEX

Introduction and objectives

The goal of DiSSCo is to improve the digitization, mobilization and subsequent use of collection data in the natural history domain. This generates a massive amount of specimen data which we will store, annotate and present. All this information needs to be findable and easily accessible for both humans and machines. That is where deliverable 6.3 comes into play.

In this deliverable, we will describe our strategy regarding exposing data through Application Programming Interfaces (APIs). DiSSCo will provide a set of generalised APIs through which data can be requested. These APIs will be used as a backend for several of DiSSCo services such as UCAS and ELViS, but can also be used by new service providers.

For the generalised set of APIs we try to build upon various standards. Some are domain specific, others are generic specifications. We will lean heavily on the recommendations written in BiCIKL Deliverable 1.3 “Best practice manual for findability, re-use and accessibility of infrastructure” (Addink 2022). This deliverable describes a set of best practices for APIs based on the input of the major infrastructure in the biodiversity domain. DiSSCo will follow these guidelines and only diverge from them where there is an explicit need.

It is important to remember that, as with all technical documentation, this document is a living document. During development of both the pilots and the final infrastructure, implementation might drift from this document. New insights gained during development might indicate that we have missed certain issues, problems, specifications and that changes make sense.

Technology Stack

The DiSSCo technology stack for data exposure consists of several components each with its specific functionality. Knowing the underlying technical implementation might help to understand the possibilities as well as the limitations of the DiSSCo infrastructure. During the development of DiSSCo this set of tools might change, depending on the use cases and the experience gained.

Relational database

DiSSCo’s data storage is based around the openDS which is a specification of Digital Specimen and other related object type definitions which are essential to mass digitization of natural science collections. For the initial implementation of the storage of the openDS data a document store was used. This type of storage uses a key, in this case the PID, and a value, the openDS, to store and lookup data.

After piloting this setup it was discovered that with the growing of the object became increasingly hard to manage. More and more information was added to the single object which made retrieval difficult. It was also difficult to search on information contained in the object, you could only search when the PID was known. Eventually it was decided to separate the different parts of the Digital Specimen (such as the Digital Media Objects, the Annotations and the Provenance) and to store these separately.

This made us rethink the data architecture and move DiSSCo's data to a relational database. However, not all our data is structured and normalizing all information will be unfeasible. That is why we will make use of the JSON column functionality implemented in most relational databases. This effectively creates a hybrid variant between a document store and a relational database. It has all the advantages of a relational database with the benefits of the stored capacities of a document store.

This approach does mean that the openDS information is split up over several tables. They are separate FAIR Digital Objects (FDO's) with their own PID and their own versioning. For example, one will hold the main Digital Specimen information while another will hold the Digital Media Object information. This makes working with the data much simpler, but also means we will have to join information from multiple tables.

Indexing Solution

In addition to stable persistent storage, we want an indexing solution for fast full-text searches. This is a functionality traditional databases lack and which will require a specific optimization solution. Within the indexing engine, we will only hold the last version of the FDO. Queries regarding full text searches or aggregated queries will be run using the indexing solution.

Additionally, the indexing solution can be used to provide data to dashboards. The dashboards will show data aggregations displaying for example the average MIDS level of a dataset or the distribution of specimens over institutions. By being able to quickly provide aggregated information, we can better monitor our digitization effort.

Data derivatives

Most data inquiries should be covered by the above two solutions. However, there may be use cases for which we require specific data products. An example could be the provision of full data dumps through an API as described in BiCIKL Deliverable 1.3, recommendation 2.5 (Addink 2022). For these data derivatives, the database would form the main source of truth. Products based on this can be created at a regular interval or at the moment of request. Instead of the immediate response coming from the API this is either a batch process running on a regular interval or an asynchronous process.

To store the data derivatives, we would look into options specifically tailored for the data. In the case of a full data dump we would use cloud storage in the form of S3 buckets to store our data. This data can then be exposed both to users and machines.

Java Backend

For serving the APIs, we will use Java as a programming language together with the Spring Boot framework. Java is a statically typed programming language, compiled before use. This enables strict control and minimizes errors due to unexpected types. It has strict exceptional handling which helps to ensure all exceptions are correctly caught and given back to the user. Additionally, it is supported by a large community and is one of the most widely used

programming languages. This means that there is a large developer pool which helps in continuity over a longer period of time. It also helps that there are plenty of examples available. Several other infrastructure or components have been written in Java, such as the Handle Server and the DOIP interface. This eases implementation and lowers the bar for exchanges of code and ideas.

The backend applications handling the requests will be stateless. Statelessness enables us to run multiple instances in parallel without creating conflicts. The main purpose for this is to be able to upscale when the load on the backend increases through extensive usage.

Routing

Incoming calls will be routed with the use of Traefik Proxy. Traefik is an open-source Edge Router which helps publish our APIs. It forms the entrypoint into our DiSSCo infrastructure, from here requests are routed to the specific services which will handle the request. Additionally, Traefik helps with securing our endpoints and enabling rate limiting, white/black listing IP's and many more options. One of the main benefits of Traefik Proxy is the dynamic configuration. This means that we can update the configuration without having to restart the proxy, limiting downtime.

General set of APIs

DiSSCo will provide a generalised set of APIs to our users. These APIs will be used by humans and machines and also by other DiSSCo services. The APIs form the backbone of the data exposure and is the main point of access into DiSSCo's data.

Although we aim to create a consistent set of APIs, we need to cater to different groups of users with different requirements. As one of our aims is to keep both the barrier to usage low and provide expert users with sufficient information, we decided to create different views of the data. For example, we provide a set of simple APIs based on the JSON:API standard for the frontend and groups of users interested in a simple JSON structure. We created a separate API with a slightly different structure for Linked Data that provides more contexts. This API follows the JSON-LD principles where it provides context on the terms used.

API for JSON:API

For the DiSSCo services and user requests we will implement a set of APIs based on the JSON:API specification. The JSON:API specification provides a set of shared conventions based on best practices. It provides clear guidelines on how endpoints should be structured and how request and response should look like. Following this specification is one of the recommendations made in BiCIKL Deliverable 1.3, see 3.4.3 and 3.7 (Addink 2022).

JSON:API endpoints will be provided for all objects and all actions. Not only read functionality will be implemented but also all necessary Create, Update and Delete actions will be based on the JSON:API specification. Examples of the objects are Digital Specimen, Organisations, Digital Media Objects and Annotations.

The data model with JSON:API is required to have at least one of the following top-level members: data, errors, metadata, or a member defined in an JSON:API extension. When data is requested, this will contain the 'data' member. This 'data' member must have a resource object which has a type and an identifier. This is where we will make the connection with FAIR Digital Object (FDO).

FDO's have the same requirements, they need to have an identifier and an FDO Type. We will use the Persistent Identifier (PID) from the FDO as the value for the JSON:API resource identifier field. The FDO Type we will use in the JSON:API resource type field. The rest of the data will go into the JSON:APIs attributes field. Below, we provide an example based on a Digital Specimen.

Example


```
{
  "data": {
    "type": "20.500.1025/ZoologyInvertebrateSpecimen",
    "id": "20.5000.1025/5TP-ELC-19L",
    "attributes": {
      "ods:primarySpecimenData": {
        "ods:midsLevel": 1,
        "dwc:county": "Helsinki",
        "dwc:country": "Finland",
```

```
      "dwc:eventID": "http://tun.fi/JX.161565#1"
    }
   },
   "links": {
    "self": "https://sandbox.dissco.tech/api/v1/specimens/20.5000.1025/5TP-ELC-19L"
   }
  }
}
```


API for JSON-LD

JSON-LD builds on top of JSON and Linked Data. It provides the ease of JSON combined with the standards of LinkedData by providing additional metadata on the attributes used. By connecting the attributes to general concepts, ontologies, it enhances interoperability and machine-readability. JSON-LD is mentioned in BiCIKL Deliverable 1.3 under section 3.4.1 as a recommendation for serialisation if the use case makes sense to conform to LinkedData (Addink 2022).

However, it adds additional complexity to the data model as it enforces a mapping to an ontology. For a user, this might overcomplicate a simple data request. Compatibility with the JSON:API specification is limited, which further complicates a single integrated endpoint. In the DiSSCo openDS work sessions, it was therefore determined that we would provide separate endpoints for several data standards/specifications. The JSON-LD endpoint won't provide the full range of CRUD functionality, users will only be able to read data following this specification. Users will have to use the JSON:API endpoints if they want to add or modify information.

The data model of JSON-LD has three important attributes. It uses the "@id" attribute for the unique identifier value. This is the attribute which we will use to provide the PID. The "@type" will be used to provide for the FDO Type. Additionally, JSON-LD uses "@context" to provide the context (namespaces) of the ontologies. Here we will refer to other ontologies so that information about the attributes can be requested and connections can be made. Some existing ontologies under discussion are W3C PROV-O for provenance data model, W3C Annotation (Ref DPP 6.4 and 5.4). Within the wider biodiversity domain Biological Collections Ontology is used which has been also under discussion (see Walls et al. 2014 for a more detail discussion on BCO and related ontologies and inadequacy of Darwin Core terms).

Example


```
{
 "@context": {
  "hdl": "https://hdl.handle.net/",
  "ods": "http://github.com/DiSSCo/openDS/ods-ontology/terms/",
  "dwc": "http://rs.tdwg.org/dwc/terms/"
 },
 "@id": "hdl:20.5000.1025/5TP-ELC-19L",
 "@type": "20.500.1025/ZoologyInvertebrateSpecimen",
 "ods:primarySpecimenData": {
  "ods:midsLevel": 1,
  "dwc:county": "Helsinki",
```

```
    "dwc:country": "Finland",
    "dwc:eventID": "http://tun.fi/JX.161565#1"
  }
}
```


API for (Cloud)events

DiSSCo intends to use an event driven design approach for data exchange as different content management systems and other databases will interact with DiSSCo (Glöckler 2022). DiSSCo systems and services need to be aware and often react to changes happening in other systems. We will use the event based approach both for exchanging information with external parties such as the data providers and data aggregators such as GBIF and GeoCase as well as internally for the exchange of information between different DiSSCo services.

For this event driven data exchange, we will build on top of existing industry standards. One of these standards is CloudEvents. In DiSSCo Prepare deliverable 6.1, CloudEvents has been introduced and examples are available there (Glöckler 2022). The deliverable also provides API Guidelines for implementation of the event driven exchange. It combines both the JSON:API specification and the CloudEvents data modeling of the event data.

Example


```
{
  "data": {
    "type": "event",
    "id": "A234-1234",
    "attributes": {
      "specversion": "1.0",
      "type": "org.dissco.event.object.created",
      "source": "https://collection.myinstitution.com/dissco/event",
      "subject": "item 123",
      "id": "A234-1234",
      "time": "2022-02-06T17:31:00Z",
      "datacontenttype": "text/json",
      "data": "{ // put the payload here }"
    },
    "links": {
      "self": "https://collection.myinstitution.com/dissco/event/A234-1234"
    }
  }
}
```


DOIP

The Digital Object Interface Protocol specifies a standard way for clients to interact with digital objects (DOs).[1] It has been built on top of the TCP specification. This means that it will

[1] https://www.dona.net/sites/default/files/2018-11/DOIPv2Spec_1.pdf

differ from the above discussed endpoint, which are all available over HTTP. DOIP is aimed at machine interoperability and requires technical knowledge to implement.

DOIP consists of a set of basic operations which cover the basic CRUD functionality. Additionally, it provides support for extending this set of basic operations with custom object type specific operations.

DiSSCo will use the Java library provided by the DONA foundation to implement the DOIP protocol.[2] Using the library, we ensure that DiSSCo is correctly implementing DOIP, and we align with other users.

Example


```
{
  "status": "0.DOIP/Status.1",
  "output": {
    "id": "20.5000.1025/5TP-ELC-19L",
    "type": "ZoologyInvertebrateSpecimen",
    "attributes": {
      "content": {
        "id": "20.5000.1025/5TP-ELC-19L",
        "ods:primarySpecimenData": {
          "ods:midsLevel": 1,
          "dwc:county": "Helsinki",
          "dwc:country": "Finland",
          "dwc:eventID": "http://tun.fi/JX.161565#1"
        }
      }
    }
  }
}
```


Asynchronous endpoints

Asynchronous endpoints are endpoints which will trigger an action but will not return the response immediately. This happens primarily when it takes time to gather the requested data. APIs are aimed to return a response from the server immediately. If this response takes minutes or even hours, the API will time out and the response is lost. In these instances, it is better to collect the data and reach out to the user when the request has been completed. There are different ways to inform the user that the requested data is ready, notifying them by email is one example.

[2] https://www.dona.net/sites/default/files/2020-09/DOIPv2SDKOverview.pdf

Within DiSSCo, we might have asynchronous endpoints. DiSSCo might provide large datasets or calculated data products which take too long or are too big to provide through a regular API. In these cases, we will indicate that the request has been successfully received, and the data processing has started. We will process the request and store the result. The user will then be informed that the data product has been completed and is ready to be downloaded. The user can now download the product through a URL provided by DiSSCo.

Versioning

Stability is incredibly important for the APIs on which other DiSSCo services are dependent. For API versioning, DiSSCo will follow the guidelines in BiCIKL deliverable 1.3, section 4 (Addink 2022). We will have a major version in the URL path of the API, for example dissco.eu/api/v1/… Breaking changes will be minimized, but when they do happen the major version in the path will be incremented. Older versions will stay functional until all users have been informed and been given time to migrate.

In general, we want to preserve backward compatibility as much as possible. This means that when we make changes, we won't remove fields but can add fields to the API. The reader will be able to use the old schema for reading and parsing data. The writer, however, can add additional data in the schema as long as older properties are not removed. This enables DiSSCo to add new information in a current version without breaking functionality of dependent APIs. When we introduce new fields for APIs where clients send us a document, CREATE/UPDATE endpoints, the new fields should be optional. This way the client will not be forced to make a change but can slowly transition to the new schema.

Documentation

Open and up-to-date documentation has been indicated as critical in BiCIKL deliverable 1.3, 1.4 (Addink 2022). Within DiSSCo we will try to keep the threshold for using the APIs as low as possible. This will include extensive documentation and examples on how to use the endpoints. DiSSCo will provide documentation in several forms.

OpenAPI v3

OpenAPI v3 has become the industry standard for documenting APIs. DiSSCo will use OpenAPI v3 documentation and expose it through our public endpoint. The documentation will not be manually written, but generated by the code. This way, the documentation will always reflect the code and will always be up-to-date. It gives guarantees towards our users and frees up time from our developers.

DiSSCo will have different applications with different sets of endpoints. Each will have its own OpenAPI documentation page. To indicate where all the OpenAPI pages live, we will provide a set of link and descriptions on our DiSSCo websites.

Swagger

OpenAPI v3 documentation is a great way to inform machines on the API structure but it is difficult to read as a human. For humans, we will create a Swagger endpoint. Swagger

provides a visual and easy to understand documentation for humans. Public endpoint can be directly tested and response objects can be reviewed. Just as the OpenAPI documentation, the Swagger endpoint will also be generated based on the code to ensure full and up-to-date documentation.

FDO Types

Almost all objects within the DiSSCo infrastructure will be FAIR Digital Objects (FDOs). One of the core aspects of an FDO is the FDO Type. The FDO Type are attributes which describe the data object. It tells the machines how to process the FDO's internal structure and payload. This enables machine actionability. Within DiSSCo all FDO Types will receive a PID which is resolvable to the FDO Type, which contains a description of the object. For these descriptions, JSON Schema will be used. JSON Schema is a declarative language which describe the data format and provides documentation for both human and machine.

Term documentation

The data model for the object will be based on the openDS data specification. This data specification will be created and maintained by DiSSCo. OpenDS will contain several new terms for describing the data properties. These terms will be extensively documented. In DiSSCo Prepare Deliverable 5.3 is described how and where the terms will be described (Fichtmueller 2023).

Metrics

DiSSCo will monitor the status of the different services. To provide insight into the current state and response time of the API, DiSSCo will provide a service monitor page for its users. This page will contain metrics about the API and other DiSSCo services and will check service availability in line with recommendation 2.3.2 in the BiCIKL Deliverable 1.3 (Addink 2022).

Environments

Within DiSSCo we use several environments: Test, Acceptance, and Production. We do this to test our code through various stages and get the approval of the stakeholders before releasing new features. The test environment is used by developers to quickly run their code in a production-like environment. It always runs the latest version of the main branch of the code. When a complete feature has been finished, the code will be released to the acceptance environment. Acceptance is used for testing by the stakeholders. They can check if the requested feature is conforming to expectations and if no issues are overlooked.

When the stakeholders agree and possible acceptance tests have been completed successfully, the code will be released to production. Code in production always reflects a specific moment in time, a specific version of the code. This way, it can be easily rolled back to an earlier version if an issue is encountered.

As APIs are part of the code base, they will follow the same path. Additionally, the acceptance environment can be used to test data imports. If users provide data, and they are

unsure if the data is correct, the acceptance environment can be used. While the acceptance environment is production-like, it is completely separated from it. This means that if adding data creates an issue, this can easily be reversed. Through the acceptance environment, DiSSCo implements recommendation 2.4 of the BiCIKL recommendations.

Security

DiSSCo uses several methods to guarantee secure APIs. APIs from an opening in the DiSSCo infrastructure and therefore need to be well protected and monitored. This protection extends to the data in which some of DiSSCo's data might be non-public and needs to be protected against unwarranted access.

Certificate/ciphers

The API will be secured by using Transport Layer Security (TLS). TLS makes sure that the communication between the user and the API is impossible to read. This means that all information will be encrypted and can only be accessed by the party that has the correct certificate. TLS certificates also enables Hyper Text Transfer Protocol Secure (HTTPS). This means that all communication between two websites is based on the TLS certificates and is therefore secured. DiSSCo will enforce HTTP Strict Transport Security (HTST) which force all connection to use the HTTPS protocol. A request to the HTTP protocol will automatically be redirected to the secure endpoint. DiSSCo will also enforce additional security headers to prevent further malicious attacks to get into the system.

For encrypting the traffic between user and server, DiSSCo only supports the latest versions of TLS, TLS 1.2 or TLS 1.3. This ensures that vulnerabilities in earlier versions of TLS cannot be exploited. This also enables the use of modern cipher suites for encrypting the traffic. DiSSCo will run regular security reviews to check whether we need to upgrade to a newer version. This might mean that older browsers will no longer be supported.

AAI

Within DiSSCo we use the data policy "As open as possible, as closed as necessary" (Hardisty 2019). This means that we will try to keep all data accessible to everyone but we might have some data that can only be reviewed by specific users. To enforce access control, we use an Authorisation and Authentication Infrastructure (AAI). This AAI enables us to authenticate users and given them authorisation to certain parts of the data.

For authorisation we will make use of the OAuth2 protocol. This industry-standard protocol creates a standardized and simple way to ensure safe access to restricted resources. On top of OAuth2, we will use OpenID Connect (OIDC) for user authentication. OIDC provides information about the user that has logged in, for example his/her email address. OIDC uses encoded JSON Web Tokens (JWT) which contain information about the users. DiSSCo will use OIDC mainly as a way to extract the users' identity from the token. Other information about the user will be stored within DiSSCo's own data storage.

Within OAuth2, bearer tokens are used to secure the endpoint. First, a user needs to request a token with an identity and access management (IAM) tool. DiSSCo will use Keycloak as

IAM tooling managed by GRNET. After the user has received the bearer token, they can make a request to the non-public API with the bearer token as a request header. Within OAuth2 this is called an “Authorization Code Flow”.[3]

There is a problem with this flow for machine to machine interaction. The machine is not an actual user, so it doesn't make sense for the machine to have a username and a password. For machine to machine interaction, we will use the “Client Credentials Flow”. We will create a client, with certain authorisation, which will generate a clientId and a clientSecret. These credentials can be used by the machine to request a token at our IAM tool. The received token can then be used by the machine to request the API.[4]

Privacy policy

DiSSCo will store certain user information in the system, such as first and last names or email addresses. It will do this to provide a better user interaction and experience. For example, we would like to know who added certain annotations to a digital specimen. Based on the name of the user, other users might assign a certain value to these annotations. We might also want to email the user when certain changes were made to his/her annotations. To store this user information, DiSSCo will request the user to accept a privacy policy. In the privacy policy, DiSSCo will state which data it will gather, how it is used, disclosed and managed.

Rate limiting

Rate limiting is generally used to prevent Distributed Denial-of-Service (DDoS) attacks. By limiting the amount of requests a single user can send to the server per time period, we can prevent a single user overloading the service. DiSSCo will evaluate if rate limiting is needed and what the rate limit needs to be to prevent malicious use of the APIs while maintaining optimal functionality for our users.

Discussion and Outlook

In this DiSSCo Prepare Deliverable we have presented the results of our work regarding specifications for a standardised set of APIs. We tried to build on industry standards and create simple, modular yet powerful endpoints. As indicated in the introduction, this document is a living document. Certain decisions might need to be reevaluated after further discussion and implementation. In particular, the decision to implement both the JSON:API specification as well as a different set of endpoints for JSON-LD might need further discussion. Use of both JSON and JSON-LD has its merits and it would be nice to combine them. However, the current added complexity made us decide to develop them separately. New versions of the specifications might improve interoperability and make us revise this decision.

During the project we tried to immediately test and implement our ideas adhering to Agile and DevOps practices. Over 35 endpoints have been written during this stage and more are added regularly. Agile and DevOps methods keep feedback loops short, assess technical

[3] https://www.rfc-editor.org/rfc/rfc6749#section-4.1
[4] https://www.rfc-editor.org/rfc/rfc6749#section-4.4

feasibility, and help to gather valuable experience with the different specifications. The main group of endpoints for the DiSSCo test environment is available through the DiSSCo sandbox. Swagger endpoint can be found at: https://sandbox.dissco.tech/api/swagger-ui.html

The next step would be to implement the specification written in this document and enforce it on all our API endpoints. Several endpoints, such as the DOIP, have not yet been implemented and will be tested before we start DiSSCo Construction phase in 2024.

Acronyms, terms, and definitions

Acronym	Term	Definition in the DiSSCO context
API	Application Programming Interface	A way for two or more applications to communicate with each other.
BiCIKL	Biodiversity Community Integrated Knowledge Library	European project that aims to connect infrastructures to enable researchers to access services across the biodiversity data lifecycle
DOA	Digital Object Architecture	DOA introduces the concept of a digital object, which forms the basis for the architecture by specification of three key components: Identifier/resolution system, repository system, and registry system.
DOIP	Digital Object Interface Protocol	A simple but powerful conceptual protocol for software applications interacting with services.
DMF	DiSSCo Modelling Framework	An instance of Wikibase where the DiSSCo data model is being developed.
DDoS	Distributed Denial-of-Service attack	A malicious attempt to disrupt the normal traffic of a targeted server, service, or network by overwhelming the target or its surrounding infrastructure with a flood of internet traffic.
ELViS	European Loans and Visits System	
FAIR	The FAIR Data principles	Guidelines to improve and foster reuse of digital research assets with respect to the four foundational principles Findability, Accessibility, Interoperability, and Reusability.
	FAIRification	Process describing the stepwise transformation from an initially non-FAIR dataset to deployment as FAIR data resource.
FDO	FAIR Digital Object	FDOs are abstracted data objects encapsulating content, descriptive metadata and globally resolvable and persistent identifiers in compliance with the FAIR principles.
FDO Type	FAIR Digital Object Type	Attribute assigned to FDOs that signals machines how to autonomously process the FDO's internal content (key requirement for machine actionability).

GRNET	Greek Research and Technology Network	Greek national infrastructure which provides e-infrastructures to academic and research institutions.
JSON	Java Script Object Notation	Lightweight, text-based, language-independent interchange format for structured data.
JSON-LD	JSON for Linking Data	JSON-LD is a syntax to serialize Linked Data in JSON and can therefore be used as an RDF syntax.
JSON:API	JSON API specification	A Specification for building APIs in JSON
JWT	JSON Web Token	An open standard for representing claims securely between two parties
LD	Linked Data	Set of methods to publish structured and connected data involving i.a. controlled vocabularies and ontologies to enable machine-interpretability of data.
MIDS	Minimum Information about a Digital Specimen	Specification defining information elements for graded digitization levels of physical specimens.
openDS	Open Digital Specimen	Specification providing the set of elements to model DES (and other curated biodiversity objects) as typed data objects compliant with FDO specifications.
OAuth2	OAuth 2.0	An industry-standard protocol for authorization
OIDC	OpenID Connect	An identify layer on top of the OAuth 2.0 protocol
RDF	Resource Description Framework	Standard data model of the Semantic Web to model resources and statements about those as triples in a knowledge graph.
	Schema.org	Comprehensive widely spread structured data vocabulary for web services with the aim to improve the findability of resources on the web.
TLS	Transport Layer Security Protocol	Protocol to provide privacy and data integrity between two communicating applications.
UCAS	Unified Curation and Annotation Service	A user-friendly website where users can search Digital Specimen and annotate them.
W3C	World Wide Web Consortium	W3C is an international community which

		develops and issues essential Web standards including HTML5, the Semantic Web stack, XML, and SPARQL.

References

Addink W, Hardisty AR (2020) ‘openDS’ – Progress on the New Standard for Digital Specimens. Biodiversity Information Science and Standards 4: e59338. https://doi.org/10.3897/biss.4.59338

Addink, W., Kyriakopoulou, N., Penev, L., Fichtmueller, D., Norton, B. & Shorthouse, D. (2022). Best practice manual for findability, re-use and accessibility of infrastructures. Deliverable D1.3 EU Horizon 2020 BiCIKL Project, Grant Agreement No 101007492.

Fichtmueller D. (2022) Milestone Report MS 5.7 "Compilation of data standards forming the basis for the initial version of the DiSSCo Digital Specimen Object Specification". https://doi.org/10.34960/jemd-0y02

Fichtmueller D., Weiland C., Grieb J., Islam S., Dillen M., von Mering S. & Güntsch A. (2023) DiSSCoPrepare Deliverable 5.3 "DiSSCo Digital Specimen Object Specifications". https://doi.org/10.34960/vn64-ws93

Glöckler F., Pim Reis J., von Mering S., Petersen, M., Weiland, C., Dillen, M., Leeflang, S., Haston, E., Addink, W., Fichtmüller, D. (2022), DiSSCo Prepare report D6.1 Harmonization and migration plan for the integration of CMSs into the coherent DiSSCo Research Infrastructure, https://doi.org/10.34960/366d-sf49

Hardisty, Alex. (2019). Provisional Data Management Plan for DiSSCo infrastructure. Deliverable D6.6. ICEDIG. https://doi.org/10.5281/zenodo.3532937

Hardisty AR, Addink W, Glöckler F, Güntsch A, Islam S, Weiland C (2021) A choice of persistent identifier schemes for the Distributed System of Scientific Collections (DiSSCo). Research Ideas and Outcomes 7: e67379. https://doi.org/10.3897/rio.7.e67379

Leeflang, S, Weiland, C, Grieb, J, Dillen, M, Islam, S, Fichtmueller, D, Addink, W, & Haston, E (2022). DiSSCo Prepare D6.2 Implementation and construction plan of the DiSSCo core architecture (1.2). Zenodo. https://doi.org/10.5281/zenodo.6832200

Walls, R.L., Deck, J., Guralnick, R., Baskauf, S., Beaman, R., Blum, S., Bowers, S., Buttigieg, P.L., Davies, N., Endresen, D. and Gandolfo, M.A., 2014. Semantics in support of biodiversity knowledge discovery: an introduction to the biological collections ontology and related ontologies. PloS one, 9(3), p.e89606. https://doi.org/10.1371/journal.pone.0089606