Rapporti Tecnici INAF
INAF Technical Reports

Number	370
Publication Year	2026
Acceptance in OA@INAF	2026-03-16T15:22:43Z
Title	Interoperability for the Gaia use case.
Authors	COSTANTINI, Massimo
Publisher's version (DOI)	https://doi.org/10.20371/INAF/TechRep/370
Handle	http://hdl.handle.net/20.500.12386/46163

Interoperability for the Gaia Use Case

Issue/Rev. No.: 1.1
Date: 7/1/2026
Author: Massimo Costantini
Affiliation: INAF - Istituto Nazionale di Astrofisica
Osservatorio Astronomico di Trieste
Gruppo IA2
Approved by: Cristina Knapic

Abstract

This Technical Report describes the design and implementation of a technically compliant Table Access Protocol (TAP), also exposing a DataLink table, developed to support the Gaia use case within the framework of the Italian National Recovery and Resilience Plan (PNRR) Spoke 3, WP 4 project, funded by the European Union, enabling interoperable access to astronomical datasets within the Virtual Observatory ecosystem.

TAP and DataLink are two examples of protocols defined by the International Virtual Observatory Alliance (IVOA), an international organization that develops standards to ensure seamless discovery, access, and interoperability of astronomical data and services across distributed archives and analysis tools.

The main goals of the Spoke 3 project are to leverage state-of-the-art solutions in High-Performance Computing (HPC) and Big Data processing and analysis, to address challenges in the fields of Astrophysics and Cosmic Observation.

Spoke 3 is structured into several Work Packages (WPs) and this Technical Report focuses on WP 4, which is dedicated to challenges related to Big Data Management, Storage and Archiving.

TAP supports synchronous and asynchronous database queries using Astronomical Data Query Language (ADQL), while DataLink provides mechanisms to associate datasets with related resources such as documentation, calibration files or processing services.

The codeset is built on a functional programming model inspired by Haskell, enforcing a strict separation between pure computations and controlled side effects. Business logic is expressed through language-agnostic abstractions and higher-order functions, ensuring portability across different languages (e.g., JavaScript and Python). This approach facilitates deployment on modern Function-as-a-Service (FaaS) platforms and containerized microservice infrastructures.

By adopting a modular, pure-functional architecture, the system achieves long-term maintainability, code reuse, and cross-language consistency, while delivering a robust foundation for interoperable astronomical data services.

Contents

1 Introduction
2 Philosophy and Design Model
3 Type Signature Examples
4 Code Examples
5 Feature Summary
6 Accessing TAP Services with TOPCAT
7 DataLink Table and Available Actions
8 Language-Agnostic and Portable Design
9 How to Run
10 Interoperability: Bringing It All Together
11 Status and Next Steps
12 References
13 Appendices (Real Code)

1 Introduction

The *Gaia* [1] mission, operated by *ESA* [2], is generating a detailed three-dimensional map of the *Milky Way*. Its data releases include hundreds of billions of measurements. Handling such a dataset requires careful database design, especially for interactive scientific queries.

This report focuses on the development of a TAP (Table Access Protocol) service and a DataLink implementation tailored to the *Gaia* use case.

The goal is to provide standards-based, interoperable access to *Gaia* data, enabling integration with existing *Virtual Observatory* [3] tools and workflows.

2 Philosophy and Design Model

The codebase adopts a *Haskell-like* [4] programming model, embracing **pure functional** programming principles. The design is **language-agnostic**, meaning the business logic can be implemented consistently across different languages and runtimes.

This model enforces an explicit separation between **pure computations** and **impure side effects**, following established functional programming paradigms. Side effects (e.g., file I/O, networking and database access) are modeled in a controlled and isolated way using **implicit continuation-passing style (CPS)**.

All effectful computations are represented as *higher-order functions*[1].

---

[1] A *higher-order function* is a function that:

- takes another function as an input (argument)
- returns a function as its output (result)

This concept comes from functional programming, but it's present in many modern programming languages.

3 Type Signature[2] Examples

This section introduces a set of representative type signatures used to illustrate how side effects are modeled consistently across different programming languages, following the functional design principles described in the previous section.

Haskell version:

```
putStr :: String -> IO ()
```

In this example, the side effect is handled implicitly by the IO monad[3].

JavaScript version:

```
(req: Req) => (IO: Symbol) => Promise<T>
```

Python version:

```
def handler(req: Req) -> Callable[[IO], Awaitable[T]]
```

In these examples, mirroring the behavior of the IO monad:

- **req** contains the pure input data (e.g., HTTP request or SQL parameters)
- **IO** is a token representing the impure world and must be explicitly passed to authorize the execution of side effects
- the return value (Promise<T> or Awaitable[T]) represents an asynchronous computation that eventually produces a value of type T

---

[2] A *type signature* is a concise declaration that specifies:

- the types of inputs (parameters) a function accepts
- the type of output (result) it returns

It's like a contract that describes what a function looks like, without showing how it is implemented.

[3] A *monad* is a structure that sequences computations and controls effects.
In Haskell, the IO monad encapsulates side effects in a pure functional way.

4 Code Examples

This section provides concrete code examples corresponding to the type signatures presented above, showing how the same abstractions are implemented in Haskell, JavaScript and Python.

Haskell version:

```
putStr :: String -> IO ()
putStr msg = putStrLn msg
```

This function encapsulates the side effect in a pure functional way, without requiring an explicit token.

JavaScript version:

```
// @returns {function(Symbol): Promise<Symbol>}
const putStr = (msg) => async (IO) => {
  console.log(msg);
  return IO;
};
```

Python version:

```
def put_str(msg) -> Callable[[object], Awaitable[None]]:
    async def inner(io):
        print(msg)
        return io
    return inner
```

These functions do not perform the side effect immediately. Instead, they return deferred computations that only execute once the **IO token** is provided. Side-effecting operations (console.log and print) remain pure descriptions until explicitly invoked with the IO token.

In JavaScript, the type signature is expressed as a JSDoc[4] comment because the language lacks built-in static typing. This annotation makes the function's contract explicit and mirrors the formal type declaration used in Haskell or in Python through type hints.

Also, in Python anonymous functions cannot contain multiple statements, so the inner function must be declared explicitly, resulting in a slightly more verbose syntax.

[4] *JSDoc* is a comment-based syntax for documenting JavaScript code, allowing developers to specify types, parameters and return values for improved readability and tooling support.

5 Feature Summary

This section summarizes the main features of the TAP and DataLink services, as well as the supporting components used for metadata storage, data retrieval and state management:

- **TAP:** /tap/availability, /tap/capabilities, /tap/sync, /tap/tables

  These endpoints provide information about the current status of the TAP service, allowing clients to check whether the service is operational, describe the features supported (e.g., output formats, query execution modes), enabling clients to automatically adapt to available options, handle synchronous execution of *ADQL queries*, returning results immediately in standard formats such as *VOTable* [5] and expose the database schema, including tables and columns available for querying, so that clients can construct informed queries.

- **DataLink:** /datalink, /datalink/gaiamerger, /datalink/progenitor, /datalink/transits

  These endpoints provide the main entry point of the DataLink protocol, returning links to additional resources or operations associated with TAP query results, access to the *Cut & Merge* service for Gaia datasets, enabling users to generate customized data subsets, allows retrieval of original (*progenitor*) datasets containing the full set of sources or transits and grants access to transit tables associated with individual sources, supporting recursive workflows (e.g., selection, subsetting and new DataLink calls).

- **External links:** gaiaMerger, Rucio

  The *gaiaMerger* [6] application is used for dataset *Cut & Merge* operations, *Rucio* [7] for large-scale data distribution and replication, and *progenitor* datasets for retrieving complete original files.

- **Metadata and Data Management:** PostgreSQL

  The *PostgreSQL* [8] database stores *Gaia* metadata together with the *Rucio Data Identifier (DID)*, which uniquely identifies the corresponding data file. This allows astronomers to query metadata via TAP and then retrieve the actual file through *Rucio*.

- **State/cache:** Redis

  Redis [9], the *NoSQL* in-memory database, manages transient state and per-user caches, improving performance and reducing system load by handling query results and session data efficiently.

  A column named *access_url*, which is required by *TOPCAT* [10] to invoke the DataLink, embeds two *Universally Unique Identifiers (UUIDs)*: the first is used to store the selected rows and queries, the second to create a personal cache for each user accessing *TOPCAT*.

  The stored selections of rows can then be passed to the DataLink service to trigger operations such as *Cut & Merge*, allowing the astronomer to generate and download a customized subset of the original dataset.

6 Accessing TAP Services with TOPCAT

Users can access the TAP schema and the *PostgreSQL* tables directly from *TOPCAT*. The integrated editor also allows query writing and editing:

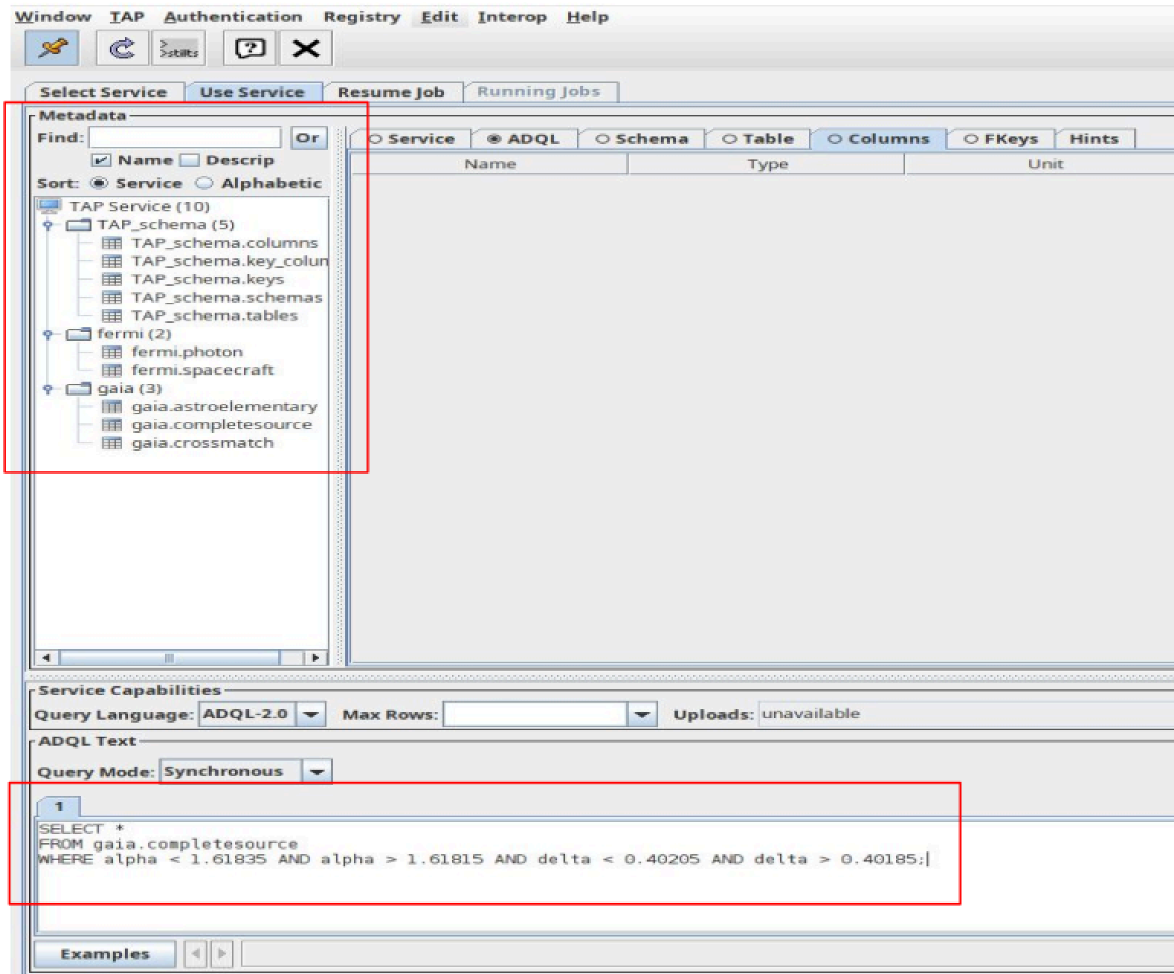
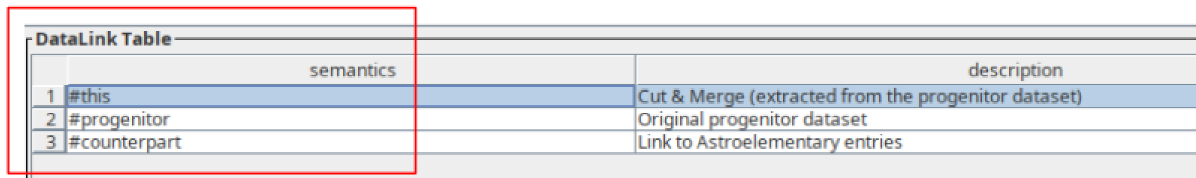


Fig. 1: TOPCAT view.

7 DataLink Table and Available Actions

The DataLink service provides additional actions associated with the query results. Each row in the table exposes links to related resources or operations, such as generating subsets of data, retrieving the original dataset or accessing complementary entries:


DataLink Table

	semantics	description
1	#this	Cut & Merge (extracted from the progenitor dataset)
2	#progenitor	Original progenitor dataset
3	#counterpart	Link to Astroelementary entries

Fig. 2: DataLink view.

Available actions:

- **#this**

  Triggers the *Cut & Merge* operation, producing a reduced dataset extracted from the *progenitor* file. Useful to download only the rows of interest.

- **#progenitor**

  Provides access to the original *progenitor* dataset containing the complete set of sources or transits, before any filtering or merging.

- **#counterpart**

  Links to related *Astroelementary table* entries (e.g., transits associated with a source), enabling recursive workflows where new subsets can be generated and processed again.

8 Language-Agnostic and Portable Design

All core abstractions (e.g., InternalError, IO, Maybe) are designed to be **language-agnostic** and consistent in semantics. Business logic is also **runtime-agnostic**, as it is decoupled from specific platforms like *Node.js* or *Python*.

The code can be easily ported to any language that supports:

- higher-order functions
- asynchronous operations (e.g., Promise<T> in JavaScript or Callable[[IO], Awaitable[T]] in Python)

This design promotes **long-term maintainability**, **code reuse** and **portability** across ecosystems.

Since all effectful computations are modeled as pure higher-order functions, this architecture is naturally suited for *Function-as-a-Service (FaaS)* platforms such as *AWS Lambda* [11] or *Google Cloud Functions* [12]. Each endpoint can be deployed independently, simply by wrapping it in a compatible runtime adapter.

Likewise, the modular design and pure functional interface make it easy to adopt a microservice-based deployment model (e.g., *Kubernetes* [13]), where each service exposes specific functionality via lightweight containers.

9 How to Run

This section outlines the basic steps required to run the TAP and DataLink services in a local development environment, providing minimal setup instructions for both the JavaScript and Python implementations.

JavaScript version:

```
node IVOA.js
```

Python (venv[5]) version:

```
python3 -m venv .venv
source .venv/bin/activate
python3 IVOA.py
```

Make sure *PostgreSQL* and *Redis* are running, with appropriate configuration files under the Conf/ directory.

[5] The *Python virtual environment* (venv) is an isolated workspace that allows installing and managing project-specific dependencies without interfering with the system-wide Python installation.

10 Interoperability: Bringing It All Together

Interoperability, in the context of the *Virtual Observatory*, refers to the ability of different services, tools and workflows to seamlessly interact through shared standards. By adopting *IVOA protocols* such as TAP and DataLink, the system ensures that astronomers can access and manipulate the same underlying data in multiple, complementary ways.

Through a **custom web portal**, users can compose queries via a graphical interface, retrieve metadata and download tailored data products. Alternatively, the same TAP service can be accessed directly from *TOPCAT*, the widely used **VO-aware** client that enables interactive query composition and visualization. Finally, astronomers can exploit programming languages such as Python, using libraries like *Astropy* and *PyVO*, to integrate TAP and DataLink queries within automated workflows, scripts or *Jupyter notebooks*[6].

This interoperability guarantees that data access is not restricted to a single interface but remains consistent and reusable across different platforms and user communities. It lowers the entry barrier for newcomers while enabling advanced users to build complex pipelines and reproducible analyses on top of the same standardized services.

11 Status and Next Steps

The current version works but is still under development: core TAP and DataLink functionality have been implemented, while *ADQL*, async TAP and *UWS (Universal Worker Service)*[7] support, are planned for future extensions.

Furthermore, the clear distinction between the pure and impure worlds also opens the way to future investigations on code correctness, understood as the ability to reason about and formally prove system properties starting from its functional specifications.

---

[6] A *Jupyter notebook* is a web-based interactive computing platform. The notebook combines live code, equations, narrative text, visualizations and more.

[7] *UWS (Universal Worker Service)* is an *IVOA standard* that defines a common pattern for managing asynchronous jobs in *Virtual Observatory services*, including their creation, monitoring, execution status and retrieval of results.

12 References

[1] *Gaia* satellite has made more than three trillion observations of two billion stars and other objects throughout our *Milky Way* galaxy and beyond:
https://www.esa.int/Science_Exploration/Space_Science/Gaia

[2] *ESA* is the European Space Agency, Europe's gateway to space. Its mission is to shape the development of Europe's space capability and ensure that investment in space continues to deliver benefits to the citizens of Europe and the world:
https://www.esa.int

[3] The *Virtual Observatory (VO)* is the vision that astronomical datasets and other resources should work as a seamless whole:
https://www.ivoa.net

[4] *Haskell* is a purely functional programming language that features referential transparency, immutability and lazy evaluation:
https://www.haskell.org

[5] The *VOTable* format is an *XML* standard for the interchange of data represented as a set of tables:
https://www.ivoa.net/documents/VOTable/20230913/WD-VOTable-1.5-20230913.html

[6] *gaiaMerger* is a *Python-based* software providing additional *Cut & Merge* services allowing users to generate files containing only the selected sources or transits extracted from the original *Gaia* catalogs.

[7] *Rucio* helps you to manage your community data. Built on more than a decade of experience, *Rucio* serves the data needs of modern scientific experiments:
https://rucio.cern.ch

[8] *PostgreSQL* is a powerful, open source object-relational database system:
https://www.postgresql.org

[9] *Redis* (Remote Dictionary Server) is a *NoSQL* in-memory database:
https://redis.io

[10] *TOPCAT* is an interactive graphical viewer and editor for tabular data:
https://www.star.bris.ac.uk/~mbt/topcat

[11] *AWS Lambda* is a serverless compute service for running code without having to provision or manage servers:
https://aws.amazon.com/lambda

[12] *Google Cloud Functions* respond to events from *Google Cloud services*, such as *Cloud Storage*, *Pub/Sub* and *Cloud Firestore*:
https://cloud.google.com/functions

[13] *Kubernetes*, also known as *K8s*, is an open source system for automating deployment, scaling and management of containerized applications:
https://kubernetes.io

13 Appendices (Real Code)

The complete source code is available at the project repository:

https://www.ict.inaf.it/gitlab/massimo.costantini/ivoa-services