



## **irods4j: A new Java client library designed for iRODS 4.3.2+**

Kory Draughn  
Chief Technologist  
iRODS Consortium

June 17-20, 2025  
iRODS User Group Meeting 2025  
Durham, NC

## Challenges with Jargon ...

- It is large and hard to maintain
- Its design makes it difficult to support certain patterns
  - Too high-level and abstract, results in a lack of control
  - API usage isn't obvious

## We want ...

- An easy-to-maintain Java library for developers
- To offer developers features only available in C and C++
- To publish Java code at Maven Central Repository

**Jargon is officially deprecated.**

Applications relying on Jargon need to migrate to irods4j.

New applications need to consider using irods4j.

Jargon is limited to updates in support of Metalnx.

## What does irods4j provide?

- Hosted at [Maven Central Repository](#)
- Easy to use while offering low-level control when needed
- Dedicated implementations for [Java 17](#) and [Java 8](#)
- Offers low-level and high-level APIs
  - Low-level API intentionally mirrors the iRODS C API
  - High-level API intentionally mirrors the iRODS C++ API
- **Documentation for the C API is documentation for irods4j**
- **native** and **pam\_password** authentication schemes are supported
- Socket options are configurable
- Supports SSL/TLS for secure communication



- Primary goals are maintainability and control
- Uses the XML protocol for communication
- Uses the [Jackson](#) JSON library for serialization of packing instructions
- Mirroring the C API future-proofs the library

```
1 var host = "localhost";
2 var port = 1247;
3 var zone = "tempZone";
4 var username = "rods";
5 var errInfo = new RErrMsg_PI();
6
7 RcComm comm = IRODSApi.rcConnect(
8     host, port, username, zone, Optional.empty(),
9     Optional.empty(), Optional.empty(), Optional.of(errInfo));
10 IRODSApi.rcDisconnect(comm);
```

## Examples - Connecting to a server

`IRODSConnection` provides an easy way for developers to connect to an iRODS server.

It manages a single `RcComm` and is inspired by the C++ `irods::client_connection` library.

```
1 try (var conn = new IRODSConnection()) {  
2     conn.connect(host, port, new QualifiedUsername(username, zone));  
3     conn.authenticate("native", password);  
4  
5     // Do work.  
6 }
```

## Examples - Connection Pooling

`IRODSConnectionPool` manages a pool of connections.

Developers can use this with `rcSwitchUser` to write client-side server applications similar to the iRODS HTTP API.

```
1 // Create a pool containing 10 connections.
2 try (var pool = new IRODSConnectionPool(10)) {
3     // Authenticate each connection in the pool.
4     pool.start(host, port, new QualifiedUsername(username, zone), conn -> {
5         try {
6             IRODSApi.rcAuthenticateClient(conn, "native", password);
7             return true; // Let the pool know authentication was successful.
8         } catch (Exception e) {
9             return false; // There was an issue, DO NOT use the pool!
10        }
11    });
12
13    try (var conn = pool.getConnection()) {
14        // Do work.
15    }
16 }
```

## Examples - Listing the contents of a collection

`IRODSCollectionIterator` allows developers to iterate over a collection's contents.

`IRODSRecursiveCollectionIterator` can be used to iterate over subcollections.

```
1 var conn = // Our connection to iRODS.
2 var collection = // Absolute path to a collection.
3
4 for (var e : new IRODSCollectionIterator(conn, collection)) {
5     // Print out the logical path of "e".
6     System.out.println(e.path());
7
8     // Inspect "e" for information about the collection entry.
9 }
```



## Examples - Writing a data object

`IRODSDataObjectOutputStream` gives developers a simple and familiar way to write data to a replica.

Built on top of `IRODSDataObjectStream` and is inspired by the C++ `irods::dstream` library.

```
1  var conn = // Our connection to iRODS.
2
3  var logicalPath = // Absolute path to a data object.
4  var truncate = true;
5  var append = false;
6  var data = "irods4j is easy to use.";
7
8  // Create a new data object and write some data to it.
9  try (var out = new IRODSDataObjectOutputStream(
10      conn, logicalPath, truncate, append)) {
11      out.write(data.getBytes(StandardCharsets.UTF_8));
12  }
```

## Examples - Reading a data object

`IRODSDataObjectInputStream` gives developers a simple and familiar way to read data from a replica.

Built on top of `IRODSDataObjectStream`.

```
1 var conn = // Our connection to iRODS.
2
3 var logicalPath = // Absolute path to a data object.
4 var buffer = new byte[512];
5
6 try (var in = new IRODSDataObjectInputStream(conn, logicalPath)) {
7     // Fill "buffer" with data from the data object.
8     in.read(buffer);
9 }
```

## Examples - Rule Execution

`IRODSRules` exposes a simple interface for executing rules.

```
1 var conn = // Our connection to iRODS.
2
3 var name = "irods4j";
4 var role = "Java client library for iRODS";
5
6 // Set the rule text, inputs, outputs, and rule engine plugin
7 // instance to use.
8 var ruleArgs = new RuleArguments();
9 ruleArgs.ruleText = String.format(
10     "*name = '%s'; *role = '%s'", name, role);
11 ruleArgs.input = Optional.empty();
12 ruleArgs.output = Optional.of(Arrays.asList("*name", "*role"));
13 ruleArgs.ruleEnginePluginInstance = Optional.of(
14     "irods_rule_engine_plugin-irods_rule_language-instance");
15
16 // Execute the rule and print the values of "*name" and "*role".
17 var results = IRODSRules.executeRule(conn, ruleArgs);
18 System.out.println(results.get("*name"));
19 System.out.println(results.get("*role"));
```

## Future Work

- Implement **pam\_interactive** authentication scheme (*in progress*)
- Document public API
- Automate and expand testing
- Leverage GitHub Actions for various tasks

Questions?

0.2.0 is available today!

Give it a try, open issues, help us make it better.

<https://github.com/irods/irods4j>