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

Motivation

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

What does this mean for Jargon?

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

Design

- 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

```
var host = "localhost";
var port = 1247;
var zone = "tempZone";
var username = "rods";
var errInfo = new RErrMsg_PI();

RcComm comm = IRODSApi.rcConnect(
    host, port, username, zone, Optional.empty(),
    Optional.empty(), Optional.empty(), Optional.of(errInfo));
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.

```
try (var conn = new IRODSConnection()) {
    conn.connect(host, port, new QualifiedUsername(username, zone));
    conn.authenticate("native", password);

    // Do work.
}
```

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.

```
// Create a pool containing 10 connections.
try (var pool = new IRODSConnectionPool(10)) {
    // Authenticate each connection in the pool.
    pool.start(host, port, new QualifiedUsername(username, zone), conn -> {
        try {
            IRODSApi.rcAuthenticateClient(conn, "native", password);
            return true; // Let the pool know authentication was successful.
        } catch (Exception e) {
            return false; // There was an issue, DO NOT use the pool!
        }
    });

    try (var conn = pool.getConnection()) {
        // Do work.
    }
}
```

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.

```
var conn = // Our connection to iRODS.
var collection = // Absolute path to a collection.

for (var e : new IRODSCollectionIterator(conn, collection)) {
    // Print out the logical path of "e".
    System.out.println(e.path());

    // Inspect "e" for information about the collection entry.
}
```

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.

```
var conn = // Our connection to iRODS.

var logicalPath = // Absolute path to a data object.
var truncate = true;
var append = false;
var data = "irods4j is easy to use.";

// Create a new data object and write some data to it.
try (var out = new IRODSDataObjectOutputStream(
                    conn, logicalPath, truncate, append)) {
    out.write(data.getBytes(StandardCharsets.UTF_8));
}
```

Examples - Reading a data object

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

Built on top of `IRODSDataObjectStream`.

```
var conn = // Our connection to iRODS.

var logicalPath = // Absolute path to a data object.
var buffer = new byte[512];

try (var in = new IRODSDataObjectInputStream(conn, logicalPath)) {
    // Fill "buffer" with data from the data object.
    in.read(buffer);
}
```

Examples - Rule Execution

`IRODSRules` exposes a simple interface for executing rules.

```
var conn = // Our connection to iRODS.

var name = "irods4j";
var role = "Java client library for iRODS";

// Set the rule text, inputs, outputs, and rule engine plugin
// instance to use.
var ruleArgs = new RuleArguments();
ruleArgs.ruleText = String.format(
    "*name = '%s'; *role = '%s'", name, role);
ruleArgs.input = Optional.empty();
ruleArgs.output = Optional.of(Arrays.asList("*name", "*role"));
ruleArgs.ruleEnginePluginInstance = Optional.of(
    "irods_rule_engine_plugin-irods_rule_language-instance");

// Execute the rule and print the values of "*name" and "*role".
var results = IRODSRules.executeRule(conn, ruleArgs);
System.out.println(results.get("*name"));
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

Thank you!

Questions?

0.2.0 is available today!

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

https://github.com/irods/irods4j