First Look: InterSystems IRIS Native API for Java

Version 2019.4
2020-01-28

InterSystems Corporation 1 Memorial Drive Cambridge MA 02142 www.intersystems.com

First Look: InterSystems IRIS Native API for Java
InterSystems IRIS Data Platform Version 2019.4 2020-01-28

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

First Look: InterSystems IRIS Native API for Java

This First Look explains how to access InterSystems IRIS® data platform globals from a Java application using the InterSystems IRIS Native functionality. In this exploration, you will first connect to InterSystems IRIS. You will then set and retrieve the value of a global node in InterSystems IRIS and iterate over the nodes of another global. You will also call an InterSystems IRIS class method. All of these activities will be performed from a Java application.

To give you a taste of IRIS Native without bogging you down in details, this exploration is intentionally simple. These activities are designed to only use the default settings and features, so that you can acquaint yourself with the fundamentals of the feature without having to deal with details that are off-topic or overly complicated. When you bring IRIS Native to your production systems, there may be things you will need to do differently. Be sure not to confuse this exploration of IRIS Native with the real thing! The sources provided at the end of this document will give you a good idea of what is involved in using IRIS Native in production.

To browse all of the First Looks, including those that can be performed on a free evaluation instance of InterSystems IRIS, see InterSystems First Looks.

1 Introduction to Globals

Globals provide an easy-to-use way to store data in persistent, multidimensional arrays. A global is a named multidimensional array that is stored within a physical InterSystems IRIS database. Within an application, the mappings of globals to physical databases is based on the current namespace — a namespace provides a logical, unified view of one or more physical databases. As an example, to associate the value “Red” with the key “Color” using a global named ^*Settings*, open the InterSystems IRIS Terminal, using the procedure described for your instance in *InterSystems IRIS Basics: Connecting an IDE*, and enter the following code:

```
set ^Settings("Color")="Red"
```

You can take advantage of the multidimensional nature of globals to define a more complex structure:

```
set ^Settings("Auto1","Properties","Color") = "Red"
set ^Settings("Auto1","Properties","Model") = "SUV"
set ^Settings("Auto2","Owner") = "Mo"
set ^Settings("Auto2","Properties","Color") = "Green"
```

For more information on globals, see *Using Globals*.

2 Why is IRIS Native Important?

IRIS Native is a feature built on top of InterSystems IRIS JDBC functionality that allows you to execute a limited subset of core ObjectScript-like commands and access InterSystems IRIS data using globals, similar to the way you would in the InterSystems IRIS Terminal. This feature takes advantage of the JDBC connection to expose core ObjectScript functionality in Java applications. Importantly, since IRIS Native uses the same connection as JDBC, InterSystems IRIS data is exposed to your Java application as both relational tables through JDBC, and as globals through IRIS Native.

InterSystems IRIS provides a unique set of capabilities to use the same physical connection and transaction context to manipulate data using multiple paradigms: native, relational, and object-oriented.

3 Exploring IRIS Native

The following brief demo shows you how to work with IRIS Native in a Java application. (Want to try an online video-based demo of InterSystems IRIS Java development and interoperability features? Check out the Java QuickStart!)

3.1 Before You Begin

To use the procedure, you will need a system to work on, with version 1.7 or 1.8 of the JDK and a Java IDE of your choice installed, and a running InterSystems IRIS instance to connect to. Your choices for InterSystems IRIS include several types of licensed and free evaluation instances; the instance need not be hosted by the system you are working on (although they must have network access to each other). For information on how to deploy each type of instance if you do not already have one to work with, see Deploying InterSystems IRIS in *InterSystems IRIS Basics: Connecting an IDE*. Connect your IDE to your InterSystems IRIS instance using the information in InterSystems IRIS Connection Information and Java IDEs in the same document.

You will also need to add the InterSystems IRIS JDBC driver, intersystems-jdbc-3.0.0.jar, to your local *CLASSPATH*. You can download this file from https://github.com/intersystems/quickstarts-java/tree/master/lib. If you have installed InterSystems IRIS on your local machine or another you have access to, you can find the file in *install-dir*\dev\java\lib\JDK18 or *install-dir*\dev\java\lib\JDK17, where *install-dir* is the InterSystems IRIS installation directory.

3.2 Connecting to InterSystems IRIS

The InterSystems IRIS connection string syntax is:

`jdbc:IRIS://`*host_IP*`:`*superserverPort*`/`*namespace*`,`*username*`,`*password*

where the variables represent the connection settings described for your instance in *InterSystems IRIS Basics: Connecting an IDE*. (This is the same information you used to connect your IDE to the instance). Set *namespace* to the predefined namespace **`USER`**, as shown in the code that follows, or to another namespace you have created in your installed instance (as long as you update the code).

If you are connecting to an instance on the local Windows machine (using either the hostname **`localhost`** or the IP address **`127.0.0.1`**), the connection automatically uses a special, high-performance local connection called a *shared memory connection*, which offers even better performance for IRIS Native. For more information, see First Look: JDBC and InterSystems IRIS.

3.3 Using IRIS Native

At this point, you are ready to experiment with IRIS Native. In your connected IDE (see Before You Begin), create a new Java project named **`IRISNative`** and paste in the following code. Make sure to edit the *superserverPort*, *username*, *namespace*, and *password* variables to reflect the correct values for your instance.

```
import java.sql.DriverManager;

import com.intersystems.jdbc.IRISConnection;
import com.intersystems.jdbc.IRIS;
import com.intersystems.jdbc.IRISIterator;


public class IRISNative {

    protected static int superserverPort = 00000; // YOUR PORT HERE
    protected static String namespace = "USER";
```

```
    protected static String username = "_SYSTEM";
    protected static String password = "SYS";

    public static void main(String[] args) {
        try {
            // open connection to InterSystems IRIS instance using connection string
            IRISConnection conn = (IRISConnection) DriverManager.getConnection
                    ("jdbc:IRIS://localhost:"+superserverPort+"/"+namespace,username,password);
            // create IRIS Native object
            IRIS iris = IRIS.createIRIS(conn);


            System.out.println("[1. Setting and getting a global]");

            // setting and getting a global
            // ObjectScript equivalent: set ^testglobal("1") = 8888
            iris.set(8888,"^testglobal","1");
            // ObjectScript equivalent: set globalValue = $get(^testglobal("1"))
            Integer globalValue = iris.getInteger("^testglobal","1");

            System.out.println("The value of ^testglobal(1) is " + globalValue);
            System.out.println();



            System.out.println("[2. Iterating over a global]");

            // modify global to iterate over
            // ObjectScript equivalent: set ^testglobal("1") = 8888
            // ObjectScript equivalent: set ^testglobal("2") = 9999
            iris.set(8888,"^testglobal","1");
            iris.set(9999,"^testglobal","2");

            // iterate over all nodes forwards
            IRISIterator subscriptIter = iris.getIRISIterator("^testglobal");
            System.out.println("walk forwards");
            while (subscriptIter.hasNext()) {
                String subscript = subscriptIter.next();
                System.out.println("subscript="+subscript+", value="+subscriptIter.getValue());
            }

            System.out.println();

            System.out.println("[3. Calling a class method]");

            // calling a class method
            // ObjectScript equivalent: set returnValue = ##class(%Library.Utility).Date(5)
            String returnValue = iris.classMethodString("%Library.Utility","Date",5);
            System.out.println(returnValue);

            System.out.println();

            // close connection and IRIS object
            iris.close();
            conn.close();

        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

}
```

The example code is split into three sections:

1. The first section shows how you set the value of a global and later retrieve it. The commands executed in this section are equivalent to the ObjectScript commands **SET** and **GET**.
2. The second section shows how to iterate through the subnodes of a global, similar to the **$ORDER** ObjectScript function.
3. The third section shows how you call an ObjectScript class method from your Java application using IRIS Native.

If the example executes successfully, you should see output with the results of the sample code:

```
[1. Setting and getting a global]
The value of ^testglobal(1) is 8888

[2. Iterating over a global]
walk forwards
subscript=1, value=8888
subscript=2, value=9999

[3. Calling a class method]
Jan 30, 2018
```

3.4 Confirming the Changes in the Management Portal

Next, confirm your results in the Management Portal, using the following procedure.

1. Open the Management Portal for your instance in your browser, using the URL described for your instance in *InterSystems IRIS Basics: Connecting an IDE*.
2. If you are not in the namespace you specified in the code, switch to it.
3. Navigate to the **Globals** page (**System Explorer** > **Globals**). You should see the *testglobal* global created in the example code. Click **View** to see its contents. You should see the two nodes of the global: `^testglobal(1) = 8888` and `^testglobal(2) = 9999`.

4 Learn More About IRIS Native

For more information on IRIS Native, globals, and InterSystems IRIS, see:

First Look: JDBC and InterSystems IRIS

Using Globals

Using the Native API for Java