



First Look: InterSystems IRIS Native API

Version 2018.1
2018-11-30

First Look: InterSystems IRIS Native API

InterSystems IRIS Data Platform Version 2018.1 2018-11-30

Copyright © 2018 InterSystems Corporation

All rights reserved.



InterSystems, InterSystems Caché, InterSystems Ensemble, InterSystems HealthShare, HealthShare, InterSystems TrakCare, TrakCare, InterSystems DeepSee, and DeepSee are registered trademarks of InterSystems Corporation.



InterSystems IRIS Data Platform, InterSystems IRIS, InterSystems iKnow, Zen, and Caché Server Pages are trademarks of InterSystems Corporation.

All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Memorial Drive, Cambridge, MA 02142, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

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.....	1
1 Introduction to Globals	1
2 Why is IRIS Native Important?	1
3 Exploring IRIS Native	2
3.1 Before You Begin	2
3.2 Things to Note about the Sample	2
3.3 Using IRIS Native	2
3.4 Confirming the Changes in the System Management Portal	4
4 Learn More About IRIS Native	4

First Look: InterSystems IRIS Native API

This First Look guide explains how to access InterSystems IRIS 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.

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, by entering the code below into the InterSystems IRIS Terminal, you can associate the value “Red” with the key “Color” using a global named `^Settings`:

```
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.

3.1 Before You Begin

To run the demo, you'll need a machine with a running, licensed instance of InterSystems IRIS, version 1.7 or 1.8 of the JDK, and a Java IDE of your choice. You will also need to add `intersystems-jdbc-3.0.0.jar` to your `CLASSPATH`. You can find this file in the subdirectory `<install-dir>\dev\java\lib\JDK18`, where `<install-dir>` is the installation directory for your instance of InterSystems IRIS. If you are using version 1.7 of the JDK, you can locate this file in the subdirectory `<install-dir>\dev\java\lib\JDK17`.

For instructions on how to install and license a development instance of InterSystems IRIS, see [Quick Start: InterSystems IRIS Installation](#).

3.2 Things to Note about the Sample

The connection string syntax is:

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

Note the following:

- *SuperServerPort* — The SuperServer port is distinct from the Web server port for InterSystems IRIS, and is set at installation time. To find the superserver port number, open the Management Portal for InterSystems IRIS and navigate to **System Administration > Configuration > System Configuration > Memory and Startup**.
- *namespace* — You must connect to a specific existing namespace in your InterSystems IRIS instance. This demo connects to the **USER** namespace.

A *shared memory connection* will offer even better performance for IRIS Native. If you are connecting to a local Windows machine (either using a hostname of `localhost` or an IP address of `127.0.0.1`), then the connection automatically uses a shared memory connection. 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. Create a new Java project named “IRISNative” using an IDE of your choice. Paste in the following code:

```
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());
}
}
}

```

Make sure to edit the *superserverPort*, *username*, *namespace*, and *password* variables to have accurate values.

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 printed 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 System Management Portal

Next, you will want to confirm your results in the System Management Portal. First, ensure that you are in the USER namespace. 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 InterSystems IRIS Native API for Java](#)