First Look: InterSystems IRIS Native API for Node.js

Version 2019.4
2020-01-28

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

First Look: InterSystems IRIS Native API for Node.js
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 Node.js

This First Look guide explains how to access InterSystems IRIS® globals from a Node.js application using the Native API. The Native API also allow you to run ObjectScript methods, functions, and routines. 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 Node.js application.

To give you a taste of the Native API 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. The sources provided at the end of this document will give you a good idea of what is involved in using the Native API 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 InterSystems IRIS Storage Structures

InterSystems IRIS provides 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 Exploring Native API for Node.js

At this point, you are ready to experiment with the Native API. This brief demo shows you how to work with the Native API in a simple Node.js application.

Want to try an online video-based demo of the InterSystems Native API for Node.js? Check out the Node.js QuickStart!

2.1 Before You Begin

To use the procedure, you will need a system to work on, with your favorite Node.js IDE 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 Node.js IDEs in the same document.

2.2 Setting Up Your Project

Next you'll need to set up your project using npm, the package manager for JavaScript that is distributed with Node.js. As part of this process, you'll install the Native API module intersystems-iris-native.

use this procedure:

1. At a command prompt, create a new folder called IRISNative for the demo, and change to that directory. For example:

   ```
   mkdir IRISNative
   cd IRISNative
   ```

2. Initialize the new project using the npm package manager:

   ```
   npm init
   ```

3. Follow the prompts to create the package.json file.
4. To install the IRIS Native API module:
   - If your InterSystems IRIS instance is installed on the system you are working on, enter the command

     ```
     npm install --save <install-dir>\dev\nodejs\intersystems-iris-native
     ```

     where <*install-dir*> is the directory where InterSystems IRIS is installed, for example C:\InterSystems\myIRIS.
   - If your instance is not installed on the system you are working on, follow these steps:
     a. Download or clone the following repo into your IDE:

        https://github.com/intersystems/quickstarts-nodejs

     b. If you downloaded the repo, unzip it.
     c. Change to the Solutions directory of the repo you just cloned or downloaded, for example:

        ```
        cd quickstarts-nodejs/Solutions
        ```

     d. To install the IRIS Native API module, enter:

        ```
        npm install --save intersystems-iris-native
        ```

2.3 The Native API Application

Now that you've created your project, you will create a small application that demonstrates a few of the features of the Native API.

1. In your IDE, create a new source file in the IRISNative directory, saving the file as IRISNative.js.

2. Paste the following code into IRISNative.js, substituting the connection information for your InterSystems IRIS instance for the values in `connectionInfo`. You can specify the **USER** namespace as shown, or you can choose another that you have created on your instance::

```
const irisnative = require('intersystems-iris-native')

// Modify connection info based on environment
let connectionInfo = {
    host: '127.0.0.1',
    port: 51773,
    ns: 'USER',
    user: '_SYSTEM',
    pwd: 'SYS'
};
// create database connection
const connection = irisnative.createConnection(connectionInfo);

//create IRIS instance
const dbnative = connection.createIris();

console.log('[1] Setting and getting a global');
// setting and getting a global
// ObjectScript equivalent: set ^testglobal("1") = 8888
dbnative.set(8888, 'testglobal', '1');

// ObjectScript equivalent: set globalValue = $get(^testglobal("1"))
let globalValue = dbnative.get('testglobal','1');
console.log('The value of testglobal is ' + globalValue);
console.log();

console.log('[2] Iterating over a global');
// modify global to iterate over
dbnative.set(7777, 'testglobal', '1');
dbnative.set(8888, 'testglobal', '2');
dbnative.set(9999, 'testglobal', '3');

let subscriptIter = dbnative.iterator('testglobal');
console.log('walk forwards');
for ([key,value] of subscriptIter) {
    console.log('subscript='+ key +', value=' + value);
};
console.log();

console.log('Iterate backwards a different way');
let revIter = dbnative.iterator('testglobal').reversed();
let node = revIter.next();
while (!node.done) {
    console.log('subscript='+ node.value[0] +', value='+ node.value[1]);
    node = revIter.next();
}
console.log();

console.log('[3] Calling a class method');
// calling a class  method
// ObjectScript equivalent: set returnValue = ##class(%Library.Utility).Date(5)
let returnValue = dbnative.classMethodValue("%Library.Utility", "Date", 5);
console.log(returnValue);

// close connection
connection.close();
```

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 Node.js application using the Native API.

Note: Globals in ObjectScript begin with the caret character (^). This is not a requirement in your Node.js applications that use the Native API.

2.4 Running the Exercise

Now you are ready to run the demo application. 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 is 8888

[2] Iterating over a global
walk forwards
subscript=1, value=7777
subscript=2, value=8888
subscript=3, value=9999

Iterate backwards a different way
subscript=3, value=9999
subscript=2, value=8888
subscript=1, value=7777

[3] Calling a class method
Apr 11, 2019
```

2.5 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`.

3 Learn More About IRIS Native

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

- *Using the Native API for Node.js*
- *Using Globals*