First Look: Globals

Version 2019.4
2020-01-28

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

First Look: Globals
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: Globals

This First Look introduces you to the concept of globals, the underlying storage structure for the InterSystems IRIS® data platform. It will show you how to access globals using a relational model and an object model, as well as how to access the globals directly.

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 What Are Globals?

One of the hallmarks of the InterSystems IRIS data platform is its ability to store data once and allow you to access it using multiple paradigms. For example, you can use InterSystems SQL to visualize your data as rows and columns, or you can use ObjectScript and think of your data in terms of objects that have properties and methods. Your application can even mix both data models, using whichever model is easiest and more efficient for a given task. But no matter how you access your data, InterSystems IRIS stores it in underlying data structures known as globals.

Globals can be thought of as persistent multidimensional sparse arrays:

- *Persistent* — Globals are stored in the database and can be retrieved at any time, by any process that can access that database.
- *Multidimensional* — The nodes in a global can have any number of subscripts. These subscripts can be integers, decimal numbers, or strings.
- *Sparse* — Node subscripts do not have to be contiguous, meaning that subscripts without a stored value do not use any storage.

Nodes in a global can store many types of data, including:

- Strings
- Numeric data
- Streams of character or binary data
- Collections of data, such as lists or arrays
- References to other storage locations

Even the server-side code you write is ultimately stored in globals!

2 Why Learn About Globals?

While it is possible to write an application on the InterSystems IRIS platform with little or no knowledge of globals, there are several reasons why you may want to learn more about them:

- Some operations may be easier or more efficient if you access globals directly.
- You may want to create custom data structures for data that does not conform to relational or object data models.

- Some system administration tasks are done at the global level, and understanding globals will make these tasks more meaningful to you.

3 Try It: Accessing Globals Three Ways

In this First Look, you will examine the globals used to store data for objects of a class of U.S. states. Properties of the class include the name of the state, its two-letter postal abbreviation, its capital, the year it was established, and its area (in square miles). Then you will access and store objects of this class using relational and object techniques, as well as by manipulating the globals directly.

3.1 Before You Begin

Before starting the exercises, you must have access to an instance of InterSystems IRIS. If you do not have an instance you can use, you can deploy one in one of the following ways:

- Deploy a cloud instance — Provision a free cloud instance of InterSystems IRIS Community Edition on the Google Cloud Platform, Microsoft Azure, or Amazon Web Services public cloud platform. For instructions for deploying and connecting to the instance, see Connecting to InterSystems IRIS in the Cloud.
- Deploy a web instance — InterSystems Labs lets you easily create your own demo instance on the web.
- Install InterSystems IRIS — If you are an InterSystems customer, you can install and license a development instance of InterSystems IRIS on your local machine or another on your network; for instructions, see InterSystems IRIS Basics: Installation. Install with Normal security settings.

You also need to know:

- The URL of the instance's web-based Management Portal, the system administration user interface for InterSystems IRIS.
- How to access the Terminal, the InterSystems IRIS command-line tool.
- Your username and password for the instance (not required for a web instance on InterSystems Labs).

You will also need to download two sample files from the InterSystems GitHub repo: https://github.com/intersystems/First-Look-Globals.

- FirstLookGlobals.xml contains the class definition for the State class.
- FirstLookGlobals.gof contains some sample global data for the original 13 U.S. states.

For more information on how to access the Management Portal or Terminal, see "InterSystems IRIS Connection Information" in *InterSystems IRIS Basics: Connecting an IDE*. You will not actually need an IDE for these exercises.

3.2 Importing and Examining the Class Definition

Start by importing the State class definition into InterSystems IRIS:

1. From the home page in the Management Portal, select **System Explorer** > **Classes**.
2. On the Classes page, look at the left column to make sure you are in the USER namespace. You can think of a namespace as a work space or a directory.
3. Click **Import**.

4. In the Import Classes dialog box:
    a. If your instance of InterSystems IRIS is running on a remote server, specify whether you downloaded the sample files to the remote server or to your local machine.
    b. Under Import from a File or a Directory, click **File**.
    c. Browse for the file FirstLookGlobals.xml, which you downloaded from GitHub.
    d. Select **Compile Imported Items**.
    e. For **Compile Flags**, specify `cuk`.
    f. Click **Next**.
    g. Click **Import**.
    h. When a message saying that the load finished successfully appears, click **Done**.

Now, on the Classes page, you should see FirstLook.State.cls in the list of classes. In InterSystems IRIS, the name of the package containing the class (FirstLook) is appended with the name of the class (State). The extension `.cls` is used to denote class files.

Note: If your namespace contains a lot of classes, filter the list by entering `F*.cls` in the **Class Name** box in the left column of the page.

To the right of the FirstLook.State.cls class, click **Documentation** to view the documentation generated for this class.

The first thing you will notice is that the FirstLook.State class extends the class %Persistent, meaning that the data for this class will be stored in the database. (System-defined classes and methods often start with %.)

```
persistent class FirstLook.State extends %Persistent
```

Extending %Persistent also makes available a number of methods you can use to perform actions on this class. For example, it allows you to create a new object of this class or access an object from the database and load it into memory.

Looking further down the page, you can see that this class has five properties: Area, Capital, Established, Name, and PostalAbbr.

```
property Area as %Integer;
property Capital as %String;
property Established as %Integer;
property Name as %String [ Required ];
property PostalAbbr as %String;
```

This class also has two indices, CapitalIndex and PostalAbbrIndex, which speed up SQL queries and allow you to quickly find states by capital and postal abbreviation.

```
index (CapitalIndex on Capital) [Unique];
index (PostalAbbrIndex on PostalAbbr) [Unique];
```

3.3 Importing the Sample Data and Examining the Globals

To get a feel for the globals that store objects of the State class, start by importing some data:

1. From the home page in the Management Portal, select **System Explorer** > **Globals**. (Or from the Classes page, click the **Globals** button.)
2. On the Globals page, look at the left column to make sure you are in the USER namespace.
3. Click **Import**.
4. In the Import Globals dialog box:

a. If your instance of InterSystems IRIS is running on a remote server, specify whether you downloaded the sample files to the remote server or to your local machine.

b. Browse for the file FirstLookGlobals.gof, which you downloaded from GitHub.

c. Click **Next**.

d. Click **Import**.

e. When a message saying that the load finished successfully appears, click **Done**.

Now, on the Globals page, you should see the globals `FirstLook.StateD` and `FirstLook.StateI` in the list. By default, the data for a class is stored in a global with a `D` appended to the name, and the indices are stored in a global with an `I` appended to the name. Most commonly, you will see globals displayed with a caret (^) in front of the name.

Note: If your namespace contains a lot of globals, filter the list by entering `F*` in the **Global Name** box in the left column of the page.

To the right of the FirstLook.StateD global, click **View** to display a listing of the contents of the global.

```
^FirstLook.StateD = 13
^FirstLook.StateD(1) = $lb("","Delaware","DE","Dover",1787,2489)
^FirstLook.StateD(2) = $lb("","Pennsylvania","PA","Harrisburg",1787,46054)
^FirstLook.StateD(3) = $lb("","New Jersey","NJ","Trenton",1787,8723)
^FirstLook.StateD(4) = $lb("","Georgia","GA","Atlanta",1788,59425)
^FirstLook.StateD(5) = $lb("","Connecticut","CT","Hartford",1788,5543)
^FirstLook.StateD(6) = $lb("","Massachusetts","MA","Boston",1788,10554)
^FirstLook.StateD(7) = $lb("","Maryland","MD","Annapolis",1788,12406)
^FirstLook.StateD(8) = $lb("","South Carolina","SC","Columbia",1788,32020)
^FirstLook.StateD(9) = $lb("","New Hampshire","NH","Concord",1788,9349)
^FirstLook.StateD(10) = $lb("","Virginia","VA","Richmond",1788,42775)
^FirstLook.StateD(11) = $lb("","New York","NY","Albany",1788,54555)
^FirstLook.StateD(12) = $lb("","North Carolina","NC","Raleigh",1789,53819)
^FirstLook.StateD(13) = $lb("","Rhode Island","RI","Providence",1790,1545)
```

You can see that the node for each state is subscripted by an integer called the object ID (or ID, for short), which is generated by the system when you store a new object of this class. The root node of the global, with no subscript, contains a counter that is incremented to generate the next ID.

Data for each node is stored as a list of properties, in the order they were specified in the class definition. If you look at `^FirstLook.StateD(3)`, you can see that the state name is New Jersey, the postal abbreviation is NJ, the capital is Trenton, the state joined the Union in 1787, and its area is 8,723 square miles.

Click **Cancel** to return to the list of globals, and then click **View** next to the global `FirstLook.StateI` to view the indices for the State class.

```
^FirstLook.StateI("CapitalIndex"," ALBANY",11) = ""
^FirstLook.StateI("CapitalIndex"," ANNAPOLIS",7) = ""
^FirstLook.StateI("CapitalIndex"," ATLANTA",4) = ""
^FirstLook.StateI("CapitalIndex"," BOSTON",6) = ""
^FirstLook.StateI("CapitalIndex"," COLUMBIA",8) = ""
^FirstLook.StateI("CapitalIndex"," CONCORD",9) = ""
^FirstLook.StateI("CapitalIndex"," DOVER",1) = ""
^FirstLook.StateI("CapitalIndex"," HARRISBURG",2) = ""
^FirstLook.StateI("CapitalIndex"," HARTFORD",5) = ""
^FirstLook.StateI("CapitalIndex"," PROVIDENCE",13) = ""
^FirstLook.StateI("CapitalIndex"," RALEIGH",12) = ""
^FirstLook.StateI("CapitalIndex"," RICHMOND",10) = ""
^FirstLook.StateI("CapitalIndex"," TRENTON",3) = ""
^FirstLook.StateI("PostalAbbrIndex"," CT",5) = ""
^FirstLook.StateI("PostalAbbrIndex"," DE",1) = ""
^FirstLook.StateI("PostalAbbrIndex"," GA",4) = ""
^FirstLook.StateI("PostalAbbrIndex"," MA",6) = ""
^FirstLook.StateI("PostalAbbrIndex"," MD",7) = ""
^FirstLook.StateI("PostalAbbrIndex"," NC",12) = ""
^FirstLook.StateI("PostalAbbrIndex"," NH",9) = ""
^FirstLook.StateI("PostalAbbrIndex"," NJ",3) = ""
^FirstLook.StateI("PostalAbbrIndex"," NY",11) = ""
^FirstLook.StateI("PostalAbbrIndex"," PA",2) = ""
```

```
^FirstLook.StateI("PostalAbbrIndex"," RI",13) = ""
^FirstLook.StateI("PostalAbbrIndex"," SC",8) = ""
^FirstLook.StateI("PostalAbbrIndex"," VA",10) = ""
```

Take a closer look at the first node in the global `^FirstLook.StateI`. Here, the first subscript (`"CapitalIndex"`) is the name of the index, the second subscript is the value of the property being indexed (`" ALBANY"`), and the third subscript is the ID of the state with the capital of Albany (`11`). If you look back at the data global, you see that `^FirstLook.StateD(11)` is the state of New York.

Globals are automatically stored in sorted order, using the array subscripts.

3.4 Accessing Globals Relationally

Now, take a look at the table InterSystems IRIS generated for the State class:

1. From the home page in the Management Portal, select **System Explorer** > **SQL**.
2. On the SQL page, look at the top of the page to make sure you are in the USER namespace. If you are not, click **Switch** to change namespaces.
3. In the left column, expand the **Tables** section to view the tables in the namespace.
4. Under Tables, click the **FirstLook.State** table.
5. On the **Catalog Details** tab, click **Fields**, and you'll see a field generated for each property of the State class, plus an extra field called ID (sometimes referred to as a RowID). This field is analogous to the object ID you saw in the globals for the class.

   Note: If your namespace contains a lot of tables, filter the list by entering `F*` in the **Filter** box in the left column of the page.

6. Click the **Execute Query** tab to open a text area where you can write queries to run against the FirstLook.State table.
7. Type the following query:

   ```
   SELECT * FROM FirstLook.State WHERE ID = 6
   ```

   and click **Execute**. The query returns the row you requested, for the state of Massachusetts.
8. Type the following query:

   ```
   INSERT INTO FirstLook.State(Area, Capital, Established, Name, PostalAbbr)
   VALUES (9616, 'Montpelier', 1791, 'Vermont', 'VT')
   ```

   and click **Execute** to insert a row into the table for Vermont, the 14th state.

To see the effects of inserting a row into the FirstLook.State table, go back to the Globals page for the namespace USER and view the globals again.

1. From the home page in the Management Portal, select **System Explorer** > **Globals**.
2. On the Globals page, look at the left column to make sure you are in the USER namespace.

Looking at the data global, `^FirstLook.StateD`, you see that the ID counter has been incremented:

```
^FirstLook.StateD = 14
```

And a new node has been added to the global:

```
^FirstLook.StateD(14) = $lb("","Vermont","VT","Montpelier",1791,9616)
```

Looking at the index global, `^FirstLook.StateI`, you see that the indices have been updated with two new nodes:

```
^FirstLook.StateI("CapitalIndex"," MONTPELIER",14) = ""
```

and

```
^FirstLook.StateI("PostalAbbrIndex"," VT",14) = ""
```

3.5 Accessing Globals as Objects

Remember that, when you created the State class, you extended the class %Persistent, which gives you access to some helpful methods that let you store data in the globals for the class and retrieve it again. Next, test some of these methods, by writing some ObjectScript in the InterSystems IRIS Terminal. (If you have not used the Terminal before, see "InterSystems IRIS Connection Information" in *InterSystems IRIS Basics: Connecting an IDE*.)

After you launch a Terminal session, you should see a prompt that indicates which namespace you are in. If you are not in the USER namespace, execute the following command:

```
set $namespace = "USER"
```

Start by loading the data for the state of Vermont, which you just added with your SQL query, into memory. Call the **%OpenId()** method of the FirstLook.State class and assign the return value to the variable `vt`. **%OpenId()** returns a "handle" to the object, more formally known as an object reference, or OREF.

```
USER>set vt = ##class(FirstLook.State).%OpenId(14)
```

You can look at the name of the state you just loaded by accessing the Name property of the object:

```
USER>write vt.Name
Vermont
```

You can get a summary of the all of the object's properties by using the **zwrite** command. The other information provided by the command is beyond the scope of this First Look.

```
USER>zwrite vt
vt=2@FirstLook.State  ; <OREF>
+----------------- general information ---------------
|      oref value: 2
|      class name: FirstLook.State
|           %%OID: $lb("14","FirstLook.State")
| reference count: 2
+----------------- attribute values ------------------
|       %Concurrency = 1  <Set>
|              Area = 9616
|           Capital = "Montpelier"
|        Established = 1791
|              Name = "Vermont"
|         PostalAbbr = "VT"
+-----------------------------------------------------
```

To create a new State object, use the **%New()** method, which returns an OREF to the new object.

```
USER>set newstate = ##class(FirstLook.State).%New()
```

Now, set the properties for the 15th state, Kentucky.

```
USER>set newstate.Name = "Kentucky"

USER>set newstate.PostalAbbr = "KY"

USER>set newstate.Capital = "Frankfort"

USER>set newstate.Established = 1792

USER>set newstate.Area = 40408
```

Inspect all of the properties.

```
USER>zwrite newstate
newstate=4@FirstLook.State  ; <OREF>
+----------------- general information ---------------
|      oref value: 4
|      class name: FirstLook.State
| reference count: 2
+----------------- attribute values ------------------
|       %Concurrency = 1  <Set>
|               Area = 40408
|            Capital = "Frankfort"
|          Established = 1792
|               Name = "Kentucky"
|           PostalAbbr = "KY"
+-----------------------------------------------------
```

If you are happy with the way everything looks, save the object to disk by calling the **%Save()** method of the new object. The **%Save()** method returns a status, which has the value `1` on a successful save.

```
USER>set status = newstate.%Save()

USER>write status
1
```

Once again, check your work by going to the Globals page in the Management Portal. Looking at the data global, `^FirstLook.StateD`, you see that the ID counter has been incremented once more:

```
^FirstLook.StateD = 15
```

And a new node has been added to the global:

```
^FirstLook.StateD(15) = $lb("","Kentucky","KY","Frankfort",1792,40408)
```

Looking at the index global, `^FirstLook.StateI`, you see that the indices have been updated with two new nodes:

```
^FirstLook.StateI("CapitalIndex"," FRANKFORT",15) = ""
```

and

```
^FirstLook.StateI("PostalAbbrIndex"," KY",15) = ""
```

3.6 Accessing Globals Directly

Though data in InterSystems IRIS is most commonly accessed from either SQL or using the object layer, you can also directly access globals for classes that extend %Persistent. This method is trickier because you need to know the structure of the global.

First, find the name of the state with the ID `15`.

From Terminal, assign the variable `ky` to the node in the data global with subscript 15.

```
USER>set ky = ^FirstLook.StateD(15)
```

You can use **zwrite** to examine the value stored at this node:

```
USER>zwrite ky
ky=$lb("","Kentucky","KY","Frankfort",1792,40408)
```

Next, use the **$list()** function to get the second item in the list:

```
USER>write $list(ky, 2)
Kentucky
```

Based upon what you learned from examining the globals after adding a state using SQL or object methods, write some code to add a new state.

First, prepare the data, putting each property (thinking in terms of objects) or field (thinking in terms of SQL) into a different variable.

```
USER>set name = "Tennessee"

USER>set postalabbr = "TN"

USER>set capital = "Nashville"

USER>set established = 1796

USER>set area = 42144
```

Then use the **$listbuild()** function to build a list of properties that can be stored.

```
USER>set properties = $listbuild("", name, postalabbr, capital, established, area)

USER>zwrite properties
properties=$lb("","Tennessee","TN","Nashville",1796,42144)
```

With a single ObjectScript statement, increment the ID counter of the `^FirstLook.StateD` global and assign the new value to the variable `id`.

```
USER>set id = $increment(^FirstLook.StateD)

USER>write ^FirstLook.StateD
16
USER>write id
16
```

Now, store the data in the data global.

```
USER>set ^FirstLook.StateD(id) = properties

USER>zwrite ^FirstLook.StateD(id)
^FirstLook.StateD(16)=$lb("","Tennessee","TN","Nashville",1796,42144)
```

Now we need to manually update the indices, or an SQL query with a WHERE clause on Capital or PostalAbbr will not include the state of Tennessee. This process is a bit more involved than storing the data.

For indices on string values, InterSystems IRIS converts the strings to uppercase and prepends a space character, to allow for easier sorting. Concatenate a space to the front of the capital using the _ operator and then convert it to uppercase with the **$zconvert()** function. Then do the same for the postal abbreviation.

```
USER>set capital = $zconvert(" "_capital, "U")

USER>set postalabbr = $zconvert(" "_postalabbr, "U")
```

Finally, store the index entries in the index global.

```
USER>set ^FirstLook.StateI("CapitalIndex", capital, id) = ""

USER>set ^FirstLook.StateI("PostalAbbrIndex", postalabbr, id) = ""
```

To make sure you've done your work correctly, go to the Globals page in the Management Portal and view the `^FirstLook.StateD` and `^FirstLook.StateI` globals. You can also use `zwrite ^FirstLook.StateD` and `zwrite ^FirstLook.StateI` from Terminal to display the complete contents of each global.

3.7 More About Globals

3.7.1 Creating Globals Using SQL

In this First Look, the globals you looked at were created by storing objects of a class specified using a class definition. You could have created globals with a very similar structure by creating a table and indices on the table using SQL. InterSystems IRIS would then have generated a class for you, based on the table you created.

As an exercise, go to the SQL page in the Management Portal for the USER namespace and execute each of the following statements using the Execute Query tab.

```
CREATE TABLE FirstLook.SQL (%CLASSPARAMETER USEEXTENTSET 0, %CLASSPARAMETER DEFAULTGLOBAL =
'^FirstLook.SQL',
Name CHAR(30) NOT NULL, PostalAbbr CHAR(2), Capital CHAR(30), Established INT, Area INT)
CREATE UNIQUE INDEX CapitalIndex ON FirstLook.SQL (Capital)
CREATE UNIQUE INDEX PostalAbbrIndex ON FirstLook.SQL (PostalAbbr)
INSERT INTO FirstLook.SQL (Name, PostalAbbr, Capital, Established, Area)
VALUES ('Maine', 'ME', 'Augusta', 1820, 35380)
```

In the left column of the SQL page, you should now see the table FirstLook.SQL.

Go to the Classes page, find the class FirstLook.SQL.cls and view its documentation. Do you notice any difference? The class has an additional index, called a Bitmap Extent Index. Then go to the Globals page and find the `^FirstLook.SQLD` and `^FirstLook.SQLI` globals. Take a look at `^FirstLook.SQLI` and see if you can find the additional index.

3.7.2 Creating Custom Globals

So far, you've looked at the globals that are created when you extend the %Persistent class. However, globals can be used to store schema-less data that may not lend itself to a class or relational paradigm. For example, using Terminal or the Management Portal, switch to the %SYS namespace, and examine the `^CONFIG` global, which stores some of the InterSystems IRIS configuration settings. A portion of a sample `^CONFIG` global is shown below:

```
^CONFIG("Cluster","CommIPAddress") = ""
^CONFIG("Cluster","JoinCluster") = 0
^CONFIG("ConfigFile","Version") = "2018.20"
^CONFIG("Conversions","LastConvertTime") = "2019-03-13 08:39:45"
^CONFIG("Databases","ENSLIB") = "/usr/irissys/mgr/enslib/"
^CONFIG("Databases","IRISAUDIT") = "/usr/irissys/mgr/irisaudit/"
^CONFIG("Databases","IRISLIB") = "/usr/irissys/mgr/irislib/"
^CONFIG("Databases","IRISLOCALDATA") = "/usr/irissys/mgr/irislocaldata/"
^CONFIG("Databases","IRISSYS") = "/usr/irissys/mgr/"
^CONFIG("Databases","IRISTEMP") = "/usr/irissys/mgr/iristemp/"
^CONFIG("Databases","USER") = "/usr/irissys/mgr/user/"
```

If you want to create and store your own custom data structures, you can easily do so using ObjectScript. Using the example below, you can write a few lines of code to define a directed graph to store the airfare between airports:

```
USER>set ^Fares("BOS", "ORD") = 87

USER>set ^Fares("ORD", "BOS") = 63

USER>set ^Fares("BOS", "LAX") = 143

USER>set ^Fares("LAX", "BOS") = 143

USER>set ^Fares("ORD", "LAX") = 57

USER>set ^Fares("LAX", "ORD") = 94

USER>zwrite ^Fares
^Fares("BOS","LAX")=143
^Fares("BOS","ORD")=87
^Fares("LAX","BOS")=143
^Fares("LAX","ORD")=94
^Fares("ORD","BOS")=63
^Fares("ORD","LAX")=57
```

4 Accessing Globals with InterSystems IRIS APIs

If you are writing an application in Java, .NET, Node.js, or Python, InterSystems IRIS provides APIs that allow you to manipulate your database using the three models discussed in this First Look:

- Relational access through JDBC, ADO.NET, or PyODBC API
- Object access through the InterSystems XEP API
- Direct access to globals through the InterSystems Native API

Now you know that, no matter how you decide to store or access your data, what you're doing is using globals.

Note: Not all forms of access are supported for all languages.

5 Learn More About Globals

Use the resources listed below to learn more about globals and how to access them.

5.1 Globals and Their Structure

- Globals QuickStart — Provides a visual look at global structures and gives an example of storing data in a custom global structure.
- Using Globals — Discusses the structure of globals, how to manage them, and how to access them using SQL or ObjectScript.
- Globals — This section in *Defining and Using Classes* explains the naming conventions for globals created for classes that extend the %Persistent class.

5.2 Multi-Model Development

- Multi-Model QuickStart — Describes how InterSystems IRIS allows you to use the data models of your choice with a variety of different languages, including Java, .NET, and ObjectScript.
- Java QuickStart — Shows you how to use the Java APIs for relational, object, and direct access to InterSystems IRIS databases.
- .NET QuickStart — Shows you how to use the .NET APIs for relational, object, and direct access to InterSystems IRIS databases.
- Python QuickStart — Shows you how to use the Python APIs for relational and direct access to InterSystems IRIS databases.

5.3 InterSystems Native API

- Using the Native API for Java (interactive course) — Shows you how to use the Native API for Java to access globals and call class methods and routines.
- Using the Native API for Java (book) — Shows you how to use the Native API for Java to access globals and call class methods and routines.
- First Look: InterSystems IRIS Native API for Java — Demonstrates how to access InterSystems IRIS globals from a Java application.
- Using the Native API for .NET (interactive course) — Shows you how to use the Native API for .NET to access globals and call class methods and routines.
- Using the InterSystems Native API for .NET (book) — Shows you how to use the Native API for .NET to access globals and call class methods and routines.
- First Look: InterSystems IRIS Native API for .NET — Demonstrates how to access InterSystems IRIS globals from a .NET application.

- Node.js QuickStart — Shows you how to use the Native API for Node.js to access globals and call class methods and routines.
- Using the Native API for Node.js — Shows you how to use the Native API for Python to access globals and call class methods and routines.
- First Look: InterSystems IRIS Native API for Node.js — Demonstrates how to access InterSystems IRIS globals from a Node.js application.
- Using the Native API for Python (interactive course) — Shows you how to use the Native API for Python to access globals and call class methods and routines.
- Using the Native API for Python (book) — Shows you how to use the Native API for Python to access globals and call class methods and routines.
- First Look: InterSystems IRIS Native API for Python — Demonstrates how to access InterSystems IRIS globals from a Python application.