First Look: ObjectScript

Version 2019.4
2020-01-28

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

First Look: ObjectScript
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: ObjectScript

This First Look introduces you to the ObjectScript programming language and gives several examples of how you can use it to store and retrieve data from the InterSystems IRIS® data platform. This First Look does not attempt to provide a thorough overview of the language or its capabilities. Use the sources listed at the end of this document to continue your exploration.

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 Is ObjectScript?

ObjectScript is a programming language designed for rapidly developing complex business applications on the InterSystems IRIS data platform. ObjectScript source code is compiled into object code that is highly optimized for operations typically found within business applications, including string manipulations and database access.

One of the unique aspects of ObjectScript is its underlying storage structure, known as *globals*. Globals can be thought of as persistent multidimensional sparse arrays. ObjectScript allows you to access your data directly from globals, but also allows you to access that data through its native object and SQL support.

While you can write an application on the InterSystems IRIS platform using Java, .NET, node.js, or other languages, by using the platform's many APIs, you can use ObjectScript to write efficient, server-based code that gives you finer control over your data.

2 Try It: Storing and Retrieving Data Using ObjectScript

In this First Look, you will learn how to use ObjectScript to:

- Store and retrieve data in globals.
- Define a class and instantiate, use, and store objects of that class.
- Access the data stored for that class using an SQL query and work with the result set.

As you will see, ObjectScript gives you the ability to store and access your data in multiple ways, for both power and flexibility.

2.1 Before You Begin

To use this First Look, you will need a running InterSystems IRIS instance. Your choices include several types of licensed and free evaluation instances; the instance need not be hosted by the system you are working on (although your system must be able to access the instance over a network). 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*.

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 Learning Labs).

You will also need to download the sample ObjectScript code from the InterSystems GitHub repo: https://github.com/intersystems/FirstLook-ObjectScript.

For more information on how to access the Management Portal or the Terminal, see "InterSystems IRIS Connection Information" in *InterSystems IRIS Basics: Connecting an IDE*. You will not actually need an IDE for these exercises. If you would like to be able to view and edit the sample code, install Atelier and see "Viewing the Sample Code in Atelier".

2.2 Importing the Sample ObjectScript Code

Start by importing the sample ObjectScript code 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 FirstLookObjectScript.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.ObjectScript.cls and FirstLook.Person.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 (ObjectScript or Person). 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.

2.3 ObjectScript and Globals

If you want to play with a few ObjectScript commands, a good way to do this is to use 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, the prompt indicates which namespace you are in. If you are not in the USER namespace, execute the following command:

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

In ObjectScript, you assign a value to a variable by using the **set** command:

```
USER>set x = "Welcome to ObjectScript!"
```

To display the contents of a variable, use the **write** command:

```
USER>write x
Welcome to ObjectScript!
```

The variable you just created exists only in this Terminal session's working memory. If you want to store a value in the database, you can use a global, which looks like a variable preceded by a caret (^):

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

Globals provide persistent storage, meaning that `^Settings` will live on after you close your Terminal session.

This particular global is also an array, with one subscript. Unlike arrays in many other languages, globals can have subscripts that are integers, decimal numbers, or strings. Globals are also sparse, meaning subscripts may or may not be contiguous.

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

```
USER>set ^Settings("Auto1", "Properties", "Color") = "Red"

USER>set ^Settings("Auto1", "Properties", "Model") = "SUV"

USER>set ^Settings("Auto2", "Owner") = "Mo"

USER>set ^Settings("Auto2", "Properties", "Color") = "Green"
```

To display the all of the nodes in a global, you can use the **zwrite** command:

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

To retrieve the value stored at a particular node of a global, you can use the **$get()** function. This function returns the empty string if you try to retrieve a value from a global node that does not exist, avoiding potential *undefined* errors.

Try the following in the Terminal:

```
USER>set ^testglobal(1) = 8888

USER>set ^testglobal(2) = 9999

USER>set globalValue = $get(^testglobal(1))

USER>write "The value of ^testglobal(1) is ", globalValue
The value of ^testglobal(1) is 8888
```

To iterate over the nodes in a global, you can use the **$order()** function, which returns the next subscript in a global. Passing in a global node with a subscript equal to the empty string causes **$order()** to return the first subscript. A return value equal to the empty string indicates that **$order()** has reached the last subscript.

You can write the value returned by a function the same way you wrote the value of a variable or a global:

```
USER>write $order(^testglobal(""))
1
USER>write $order(^testglobal(1))
2
USER>write $order(^testglobal(2))

USER>
```

In practice, you would usually create a **$order()** loop within a *class method* in an ObjectScript class file. The following method has been provided for you in FirstLook.ObjectScript.cls.

```
/// Iterate over global ^testglobal
ClassMethod Iterate() {
  // Start by setting subscript to ""
  set subscript = ""
  // "Argumentless" for loop
  for {
    // Get the next subscript
    set subscript = $order(^testglobal(subscript))
    // When we get to the end, quit the for loop
    quit:(subscript = "")
    // Otherwise, write the subscript and the value
    // stored at ^testglobal(subscript)
    write !, "subscript=", subscript, ", value=", ^testglobal(subscript)
  }
}
```

Note that the “argumentless” for loop specifies no terminating condition and would loop forever, if we did not explicitly quit the loop. The exclamation point in the **write** statement tells ObjectScript to move to the next line before writing the subscript and value of each node in the global.

To run the method in the Terminal, type:

```
USER>do ##class(FirstLook.ObjectScript).Iterate()
```

This yields the output:

```
subscript=1, value=8888
subscript=2, value=9999
```

2.4 ObjectScript and Objects

You can use ObjectScript to create classes that have methods and properties. You can then instantiate objects of that class. The sample class FirstLook.Person.cls defines a class Person and then lets you create instances of that class, such as the person John Smith or the person Jane Doe.

The basic class definition looks like this:

```
Class FirstLook.Person Extends %Persistent
{

Property FirstName As %String [ Required ];

Property LastName As %String [ Required ];

}
```

The Person class extends the built-in InterSystems IRIS class %Persistent, which gives you access to some helpful methods, such as **%New()** and **%Save()**. Then the properties of the class are listed. In this case, you are simply storing a person’s first and last name.

Go ahead and create a new person using the Terminal. If you are not in the USER namespace, execute the following command:

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

To create a new Person object, use the **%New()** method, which returns a “handle” to the new person, more formally known as an object reference, or OREF. Then set the new person’s properties and call the **%Save()** method to store the new person in the database.

```
USER>set person = ##class(FirstLook.Person).%New()

USER>set person.FirstName = "Sarah"

USER>set person.LastName = "Aarons"

USER>set status = person.%Save()

USER>write status
1
```

The **%Save()** method returns a status, which has the value `1` if it succeeds.

When you save an object, InterSystems IRIS stores it in a global for you automatically. The default global name is the class name with a `D` appended to the end, in this case `^FirstLook.PersonD`.

If you display the contents of the global, you can see that the root node (the node without a subscript) holds an ID that is incremented for each new object that is stored. The remainder of the global is subscripted by ID. Each Person node contains a list of property values, where the list is denoted by **$lb** , for "list build."

```
USER>zwrite ^FirstLook.PersonD
^FirstLook.PersonD=1
^FirstLook.PersonD(1)=$lb("","Sarah","Aarons")
```

You can also define *instance methods*, which operate on a specific instance, as opposed to class methods, which are generic to a class. For example, FirstLook.Person.cls contains a **WriteName()** method, which writes a person's name.

```
/// Given an instance of a person, write person's name
Method WriteName() {
  write "The name of this person is:"
  write !, ..FirstName
  write !, ..LastName
}
```

An extra dot in front of a property or method name indicates the current object or class.

Since the variable `person` currently refers to Sarah Aarons, you can write her name like so:

```
USER>do person.WriteName()
The name of this person is:
Sarah
Aarons
```

As an exercise, create, store, and write a few new objects of the Person class, for example, Andrew Shaw, Peter Shaw, and Kate Aarons.

2.5 ObjectScript and SQL

You've just seen how each person you create is stored as a node in a global. As you will see in this section, each person is also a row in a table, accessible using SQL.

InterSystems offers several ways to use SQL from within ObjectScript. For example, you can use a *class query*, which is basically an SQL query inside a class file.

The following class query, in FirstLook.Person.cls, does a SELECT command on all objects in the class:

```
/// Query for all stored names
Query Select() As %SQLQuery [SqlProc]
{
  SELECT %ID, FirstName, LastName
  FROM Person
  ORDER By LastName, FirstName
}
```

To test the query, you can run it from the Terminal:

```
USER>do ##class(%ResultSet).RunQuery("FirstLook.Person", "Select")
```

The output will show a list containing each person you created and stored in the previous exercise, sorted by last name, then first name:

```
ID:FirstName:LastName:
4:Kate:Aarons:
1:Sarah:Aarons:
2:Andrew:Shaw:
3:Peter:Shaw:
```

In real life, you might write a class method, such as the one provided for you in FirstLook.Person.cls, to place the results of the query in a result set and then iterate through each row in the set:

```
/// Run select query and write all names in result set
ClassMethod WriteAllNames() {
  // Create a new ResultSet object
  set resultSet = ##class(%ResultSet).%New()
  // Specify the class query to execute by setting
  // the ClassName and QueryName properties.
  set resultSet.ClassName = "FirstLook.Person"
  set resultSet.QueryName = "Select"
  // Execute the query
  set status = resultSet.%Execute()
  // If status = $$$OK, query is successful
  if (status = $$$OK) {
    // Iterate over results
    while (resultSet.%Next()) {
      // Write person's first and last name
      write !, resultSet.FirstName, " ", resultSet.LastName
    }
  }
}
```

In this class method, **resultSet.%Execute()** returns a status of `1` if it is successful. This is often written in code using the `$$$OK` macro for easier readability. The code loops through the result set using **resultSet.%Next()**, which returns `true` if another row exists.

To run the method in the Terminal, type:

```
USER>do ##class(FirstLook.Person).WriteAllNames()

Kate Aarons
Sarah Aarons
Andrew Shaw
Peter Shaw
```

As you have seen, ObjectScript gives you several options for working with your data. Using globals gives you the most control over how your data is stored, using objects makes it easy to work with single instances of a class, and SQL gives you the power to operate across the rows of a table. How you choose to think about your data is entirely up to you.

3 Viewing the Sample Code in Atelier

If you want to take a closer look at the classes you used in this First Look, you can view or edit the ObjectScript code in Atelier:

1. In Eclipse, click the **Atelier** perspective at the top right corner.
2. In the left pane, click the **Server Explorer** tab.
3. On the Server Explorer tab, expand the connection to your InterSystems IRIS instance.
4. Under that instance, expand the **USER** namespace
5. Under the USER namespace, expand **Classes**.
6. Right-click the FirstLook package, and then click **Copy to Project**.
7. In the Copy dialog box, select a project, for example, the DemoProject project you created in the Atelier First Look.
8. Click **Finish**.
9. In the left pane, click the **Atelier Explorer** tab.
10. On the Atelier Explorer tab, expand the project, and then expand the **Classes** folder.
11. In the FirstLook package, double-click a class file to open it in the right pane of Eclipse.

4 Learn More About ObjectScript

Use the resources listed below to learn more about programming in ObjectScript.

- ObjectScript Tutorial — Provides an interactive introduction to the ObjectScript language.
- Using ObjectScript — Provides an overview of and details about the ObjectScript programming language.
- ObjectScript Reference — Provides reference material for ObjectScript.
- Orientation Guide for Server-Side Programming — Presents the essentials for programmers who write server-side code using InterSystems products.
- InterSystems ObjectScript Basics — An interactive course covering ObjectScript basics.