First Look: InterSystems IRIS Native API for .NET

Version 2019.4
2020-01-28

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

First Look: InterSystems IRIS Native API for .NET
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 .NET

This First Look explains how to access InterSystems IRIS® data platform globals from a .NET application using the InterSystems IRIS Native functionality. IRIS Native also allows 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 .NET 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!

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

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 IRIS Native for .NET

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

2.1 Before You Begin

To use the procedure, you will need a Windows system to work on, with the .NET framework and Visual Studio 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 Visual Studio to your InterSystems IRIS instance using the information in InterSystems IRIS Connection Information and .Net IDEs in the same document.

2.1.1 Configuring the Visual Studio Project

To begin, open Visual Studio and create a new console app project, selecting the **Visual C#** and **Console App (.NET Framework)** options. For the **Name** field, enter `netnative`.

2.1.2 Adding the Assembly Reference

The InterSystems.Data.IRISClient.dll assembly must be present on your local system; you can download it from https://github.com/intersystems/quickstarts-dotnet/tree/master/EFPlay/bin/Debug. If InterSystems IRIS is installed on your local machine or another you have access to, you can find the file in *install-dir*\dev\dotnet\bin\v4.5, where *install-dir* is the installation directory for the instance.

To add an assembly reference to InterSystems.Data.IRISClient.dll to a project:

1. From the Visual Studio main menu, select **Project** > **Add Reference...**
2. In the resulting window, click **Browse...**.
3. Browse to the location of the InterSystems.Data.IRISClient.dll file.
4. Select the file and click **Add**.
5. Click **OK**.

In the Visual Studio Solution Explorer, the InterSystems.Data.IRISClient.dll assembly should now be listed under **References**.

2.2 Using IRIS Native

At this point, you are ready to experiment with IRIS Native. Open the file that was created when you created the Visual Studio project (for example, Program.cs). Delete the default content of the file and paste in the following code, substituting the connection information for your InterSystems IRIS instance for the values in `conn.ConnectionString`. (The value of `logfile` must be a writable path on the local system). You can specify the **USER** namespace as shown, or you can choose another that you have created on your instance.

```
using System;
using InterSystems.Data.IRISClient;
using InterSystems.Data.IRISClient.ADO;

public class IRISNative
{

    public static void Main(String[] args)
    {
        try
        {
            // open connection to InterSystems IRIS instance using connection string
            IRISConnection conn = new IRISConnection();

            // edit this ConnectionString to match your environment
            conn.ConnectionString = "Server=localhost; Port=51773; Namespace=User; Password=SYS; User
 ID=_system; \
              logfile=c:\\sandbox\\dbnative.log";
            conn.Open();
```

```
            // create IRIS Native object
            IRIS iris = IRIS.CreateIRIS(conn);

            Console.WriteLine("[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"))
            Int16? globalValue = iris.GetInt16("^testglobal", "1");

            Console.WriteLine("The value of ^testglobal(1) is " + globalValue);
            Console.WriteLine();


            Console.WriteLine("[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
            Console.WriteLine("walk forwards");
            IRISIterator subscriptIter = iris.GetIRISIterator("^testglobal");
            foreach (var node in subscriptIter)
            {
                Console.WriteLine("subscript=" + subscriptIter.CurrentSubscript + ", value=" + node);
            }
            Console.WriteLine();


            Console.WriteLine("[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);
            Console.WriteLine(returnValue);

            Console.WriteLine();

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

        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }

}
```

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 .NET application using IRIS Native.

If you are using a locally installed instance and the connection therefore uses `localhost` as the server address, the program may use a local shared memory connection, which is even faster than the standard TCP/IP connection.

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

2.3 Running the Exercise

You are now ready to run the demo application by pressing `Ctrl+F5`. 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]
Oct 30, 2018
```

2.4 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`.

3 Learn More About IRIS Native

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

Using the Native API for .NET

First Look: InterSystems IRIS Native API for Java

Using Globals

Using the InterSystems Managed Provider for .NET