InterSystems
IRIS Data Platform

First Look: .NET Object Persistence with XEP

Version 2019.4
2020-01-28

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

First Look: .NET Object Persistence with XEP
InterSystems IRIS Data Platform Version 2019.4 2020-01-28

Table of Contents

First Look: .NET Object Persistence with XEP

This First Look introduces the XEP API, which provides support for extremely fast .NET object storage and retrieval on InterSystems IRIS® data platform. It gives you a high level overview of the XEP approach to .NET object persistence, and walks you through a simple scenario that demonstrates the main features of the API.

These activities are designed to use only the default settings and features so that you can acquaint yourself with the fundamentals of XEP without having to deal with details that are beyond the scope of this overview. For the full documentation on XEP, see *Persisting .NET Objects with InterSystems XEP*.

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 Rapid Object Storage and Retrieval

Object-oriented programming is at the core of the .NET framework, so it is natural for .NET applications to model data as objects. However, this can cause problems when the application needs to store that data in a database. If you use ADO.NET to store and retrieve objects, you are faced with the problem of transforming your object data into a set of relational tables, and then transforming query resultsets back into objects. The usual solution to this problem is to use an object-relational mapping (ORM) framework such as Entity Framework to automate the process. InterSystems IRIS does offer an Entity Framework interface, and InterSystems recommends it for large, complex object hierarchies. On the other hand, for applications that perform tasks such as real-time data acquisition, the main problem is speed rather than data complexity. Although Entity Framework is by no means slow (if properly optimized), XEP is a better alternative for tasks that require extremely fast storage and retrieval of simple or moderately complex data. In most cases, it will be considerably faster than either Entity Framework or ADO.NET.

2 How Does XEP Work?

XEP is a lightweight .NET API that projects .NET object data as *persistent events*. A persistent event is an instance of an InterSystems IRIS class (normally a subclass of %Persistent) containing a copy of the data fields in a .NET object. Like any such instance, it can be retrieved by object access, SQL query, or direct global access.

Before a persistent event can be created and stored, XEP must analyze the corresponding .NET class and *import a schema* into the database. A schema defines the structure of the persistent event class that will be used to store the .NET objects. Importing the schema automatically creates a corresponding ObjectScript schema for the persistent event class if one does not already exist. The information from the analysis may be all that XEP needs to import a simple schema. For more complex structures, you can supply additional information that allows XEP to generate indexes and to override the default rules for importing fields.

After the schema has been created for a class, you can use various XEP methods to store, update, or delete events, run SQL queries, and iterate through query resultsets.

3 Try It! Hands-on with XEP

Now it's time for you to try out XEP for yourself. This XepSimple demo is a very small program, but it provides examples for most of the key XEP features and will give you an overview of how the XEP API is used. (Want to try an online video-based demo of InterSystems IRIS .NET development and interoperability features? Check out the .NET QuickStart!)

3.1 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.

3.1.1 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 `netxep`. (Want to try an online video-based demo of InterSystems IRIS .NET development and interoperability features? Check out the .NET QuickStart!)

3.1.2 Adding the Assembly Reference

The XEP API is packaged into the InterSystems.Data.XEP.dll library, which must be installed on your local system. You can obtain it by cloning the repo https://github.com/intersystems/quickstarts-dotnet/tree/master/XEP or downloading the file from the repo. 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.XEP.dll to your 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.XEP.dll file.
4. Select the file and click **Add**.
5. Click **OK**.

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

3.2 Adding a Sample Class

Before creating the program source file, you will add a class that the main program will access to generate sample objects for storage. In Visual Studio, use **Project > Add Class** to add a C# class file to the **netxep** project. Delete the default contents of the class file, and copy and paste the following code into the file.

```
using System;

namespace xep.samples
{
    public class SingleStringSample {
        public string name;
        public SingleStringSample() { }
        internal SingleStringSample(string str) {
            name = str;
        }

        public static SingleStringSample[] generateSampleData(int objectCount) {
            SingleStringSample[] data = new SingleStringSample[objectCount];
            for (int i = 0; i < objectCount; i++)
            {
                data[i] = new SingleStringSample("single string test");
            }
            return data;
        }
    }
}
```

Because this class is very simple, XEP does not need any annotations to generate an effective schema from it. For more complex classes in a real-world application, you can use various options to optimize the import.

3.3 Creating the XEP Demo Program

Now you are ready to create the XEPSimple demo program. In Visual Studio, find the default program.cs file that was created when you created the project. Delete the default contents of the file and paste in the code that follows, substituting the connection information for your InterSystems IRIS instance for the values of the `host`, `port`, `irisnamespace`, `username`, and `password` variables. You can specify the **`USER`** namespace as shown, or you can choose another that you have created on your installed instance.

```
using System;
using InterSystems.XEP;
using xep.samples;

namespace XepSimpleNamespace
{
    public class XepSimple
    {
        public static void Main(string[] args)
        {
            // Generate 12 SingleStringSample objects for use as test data
            SingleStringSample[] sampleArray = SingleStringSample.generateSampleData(12);

            // EventPersister
            EventPersister xepPersister = PersisterFactory.CreatePersister();

            String host = "127.0.0.1"; // InterSystems IRIS host
            int port = 51774; // InterSystems IRIS Superserver port
            String irisnamespace = "USER"; // InterSystems IRIS namespace
            String username = "_system"; // Credentials for InterSystems IRIS
            String password = "SYS"; //Credentials for InterSystems IRIS

            xepPersister.Connect(host,port,irisnamespace,username,password); // connect to localhost
            xepPersister.DeleteExtent("xep.samples.SingleStringSample");   // remove old test data
            xepPersister.ImportSchema("xep.samples.SingleStringSample");   // import flat schema

            // Event
            Event xepEvent = xepPersister.GetEvent("xep.samples.SingleStringSample");

            long[] itemIdList = xepEvent.Store(sampleArray);
            int itemCount = 0;
            for (int i = 0; i < itemIdList.Length; i++)
            {
                if (itemIdList[i] > 0) itemCount++;
            }
            Console.WriteLine("Stored " + itemCount + " of " + sampleArray.Length + " events");


            // EventQuery
            EventQuery<SingleStringSample> xepQuery = null;
            String sqlQuery = "SELECT * FROM xep_samples.SingleStringSample WHERE %ID BETWEEN ? AND
?";
            xepQuery = xepEvent.CreateQuery<SingleStringSample>(sqlQuery);
            xepQuery.AddParameter(3);    // assign value 3 to first SQL parameter
            xepQuery.AddParameter(12);   // assign value 12 to second SQL parameter
            xepQuery.Execute();            // get resultset for IDs between 3 and 12


            xepQuery.Close();
            xepEvent.Close();
            xepPersister.Close();
        } // end main()
    } // end class xepSimple
}
```

The demo program is divided into three sections, each of which demonstrates one of the three main XEP classes, EventPersister, Event, and EventQuery. When run, the XepSimple demo program performs the following tasks:

- First, some sample objects are generated by calling a method of your sample data class, xep.samples.SingleStringSample.
- EventPersister *is the main entry point for XEP, and provides the connection to the database.*

  It creates an instance named *xepPersister*, which establishes a TCP/IP connection to the User namespace in the database and deletes any existing sample data. It then calls **ImportSchema()** to analyze the sample class and send the schema to the database, thus creating an the corresponding ObjectScript schema to hold persistent SingleStringSample objects.
- Event *encapsulates an interface between a .NET object and the corresponding database object.*

  Once the schema has been generated, *xepPersister* can create an Event object named *xepEvent* for the sample class. The **Store()** method stores the array of .NET objects as persistent events.
- EventQuery *is used to prepare and execute a limited subset of SQL queries.*

  An EventQuery<SingleStringSample> object named *xepQuery* is created by passing a query string to the *xepEvent* object's **CreateQuery()** method. The string defines an SQL query that accepts two parameters (the ? characters). The parameter values are defined by calls to **AddParameter()**, and a call to **Execute()** fetches the query resultset.
- When processing is done, it cleans up by calling the **close()** methods for the XEP objects.

3.4 Running the XEPSimple Program

You are now ready to build and run the XEPSimple demo program. Press `Ctrl + F5` to run the program so the command prompt stays open when the program ends.

3.5 Next Steps

The XepSimple demo is designed to give you a taste of XEP without bogging you down in details, and is not a model for real production code — it didn't even bother with exception checking. The most important simplification was in the sample data. You persisted a few tiny .NET objects from a class that doesn't require annotation or other mechanisms to aid in schema generation and optimization. Though you usually need annotations and other options in a real world application, these options are straightforward and easy to use. The APIs are similarly simple and easy to use.

For a comprehensive description of all XEP features, see *Persisting .NET Objects with InterSystems XEP*.

4 Learn More About .NET Support

InterSystems IRIS provides .NET APIs for easy database access via SQL tables, objects, and multidimensional storage. See the following books for detailed information on each type of access:

- *Using the Native API for .NET* for accessing InterSystems IRIS globals from a .NET application.
- *Using ADO.NET Managed Provider Classes* for SQL table access. Allows your .NET projects to access InterSystems IRIS databases with fully compliant versions of generic ADO.NET Managed Provider classes.
- *Using the InterSystems ODBC Driver* for SQL table access. The InterSystems ODBC driver allows InterSystems IRIS to establish ODBC connections to external applications, and provides access to external data sources via SQL.
- *Persisting .NET Objects with InterSystems XEP* for object access. XEP is optimized for transaction processing applications that work with simple to medium complexity object hierarchies and require extremely high speed object data persistence and retrieval.

- *Using the Entity Framework Provider* for object—relational mapping. Provides information about accessing an InterSystems IRIS database using Entity Framework technology.