Connecting Your Application to InterSystems IRIS

2024-05-06

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

Connecting Your Application to InterSystems IRIS
InterSystems IRIS Data Platform 2024-05-06

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

Connecting Your Application to InterSystems IRIS

This document explains how to make programmatic connections to an InterSystems IRIS® Data Platform cloud service, instance, or cluster from applications written in Python, Java, .NET, and C++. Such connections encompass multiple options and can be coded in different ways; this document provides illustrative examples that can be quickly examined and understood. In addition to instructions for basic connection code, this document shows you how to protect connections with TLS encryption, using a self-signed certificate provided by the target. Each of the sections listed below contains a link to more comprehensive documentation.

Connect from Python using DB-API	Secure a DB-API connection with TLS encryption
Connect from Java using JDBC	Secure a JDBC connection with TLS encryption
Connect from .NET using ADO.NET	Secure an ADO.NET connection with TLS encryption
Connect from C++ using ODBC	Secure an ODBC connection with TLS encryption

Connecting your Python, Java, .NET, or C/C++ application to InterSystems IRIS involves just three simple steps!

1. Download the InterSystems driver package for DB-API (Python), JDBC (Java), ADO.NET (.NET), or ODBC (C++).
2. Gather the connection information for the InterSystems IRIS target you want to connect to, which includes the host identifier (hostname or IP address) and superserver port of the target instance, credentials to authenticate to InterSystems IRIS with sufficient privileges to execute the desired actions, and the namespace to connect to. You can find the connection information for an InterSystems IRIS cloud service, such as InterSystems IRIS Cloud SQL, on your deployment's Deployment Details page in the InterSystems Cloud Services Portal. For InterSystems IRIS clusters and instances on various platforms, see InterSystems IRIS Connection Information.
3. Add the needed code to your application, as explained in the following sections. You can copy the completed connection code from the listing at the end of each section.

For detailed online learning content about the use of some of these languages with InterSystems products, see Connecting to InterSystems Products with External Languages. For information about all of the connections tools available from InterSystems, see Connection Tools.

Important: One of the required parameters in the connection code shown in this document is *namespace* (with the value USER throughout). InterSystems IRIS makes the distinction between the physical *databases* that store data and the logical *namespaces* used to interact with them, and the relationships between namespaces and databases may differ from one InterSystems IRIS instance to the next. To interact with any given data, therefore, you must determine and specify the appropriate namespace.

Third-party tools and technologies you might use to connect to InterSystems products or services likewise interact with namespaces only, but most use the standard term *database* to refer to them.

1 Connect from Python using DB-API

Before using these instructions, you should make sure that:

- Your development system has the Python IDE of your choice installed and has network access to your InterSystems IRIS target.
- You have downloaded the InterSystems DB-API driver, **intersystems_irispython-***version*-py3-none-any.whl, to your development system.

  If InterSystems IRIS is installed locally or in a container on your development system, you can also find the file in *install-dir*\dev\python, where *install-dir* is the InterSystems IRIS installation directory (*install-dir* in a container is /usr/irissys).
- You have gathered the connection information for the InterSystems IRIS target you want to connect to.

You can copy the completed connection code from the listing that follows the instructions. Be sure to review the instructions in the following section for adding TLS encryption to the connection.

To code a DB-API connection to InterSystems IRIS, follow these steps:

1. Install the DB-API driver:

   ```
   C:\> pip install intersystems_irispython-version-py3-none-any.whl
   ```

2. Import the **iris** module and create a main method:

   ```
   1    import iris
   2
   3    def main():
   ```

   Note: When using embedded Python, the **iris** module is typically imported using **iris.cls**, **iris.gref**, or **iris.sql**. This differs from the `import iris` statement for the DB-API driver, as shown in this code.

3. Use the connection information for the InterSystems IRIS target to define the connection string, in the form *host-identifier*:*superserver-port*/*namespace*, and the credentials:

   ```
   3    def main():
   4        connection_string = "localhost:1972/USER"
   5        username = "Admin"
   6        password = "--------"
   ```

   You can also define the host identifier, port, and namespace separately, as shown for the credentials here, and pass all five to **iris.connect** when creating the connection, as shown for the connection string and credentials in the next step.

4. Finally, create the connection by calling **iris.connect**:

   ```
   1    import iris
   2
   3    def main():
   4        connection_string = "localhost:1972/USER"
   5        username = "Admin"
   6        password = "--------"
   7
   8        connection = iris.connect(connection_string, username, password)
   9
   10       # when finished, use the line below to close the connection
   11       # connection.close()
   12
   13   if __name__ == "__main__":
   14       main()
   ```

You can copy the code used here from the listing below. For more information about using Python and DB-API with InterSystems IRIS, see Python DB-API Support.

```
import iris

def main():
    connection_string = "host-identifier:superserver-port/namespace"
    username = "username"
    password = "password"

    connection = iris.connect(connection_string, username, password)

    # when finished, use the line below to close the connection
    # connection.close()

if __name__ == "__main__":
    main()
```

2 Securing the DB-API Connection with TLS

To extend the instructions above to code a DB-API connection with TLS encryption using a self-signed X.509 certificate provided by the InterSystems IRIS target, do the following:

1. Obtain or download the self-signed certificate as instructed and place it in a secure location. (If necessary, download the InterSystems DB-API driver, **`intersystems_irispython-`***version*-py3-none-any.whl.)
2. Modify the code provided above in these ways:
   a. Import the **`ssl`** module in addition to the **`iris`** module.
   b. Before creating the connection, specify the TLS context and verify the existence of the required certificate.

You can copy the DB-API connection code **including TLS encryption** from the listing below. Before using this minimal TLS configuration in production, please consult TLS/SSL wrapper for socket objects in the Python documentation.

```
import iris
import ssl

def main():
    connection_string = "host-identifier:superserver-port/namespace"
    username = "username"
    password = "password"

    context = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
    context.verify_mode=ssl.CERT_REQUIRED
    context.check_hostname = False
    context.load_verify_locations("path-to-cert/cert-file.pem")

    connection = iris.connect(connection_string, username, password, sslcontext=context)

    # when finished, use the line below to close the connection
    # connection.close()

if __name__ == "__main__":
    main()
```

3 Connect from Java using JDBC

Before using these instructions, you should make sure that:

- Your development system has version 1.8 of the JDK and the Java IDE of your choice installed and has network access to your InterSystems IRIS target.

- You have downloaded the InterSystems JDBC driver, **`intersystems-jdbc-`***version*.jar, to your development system.

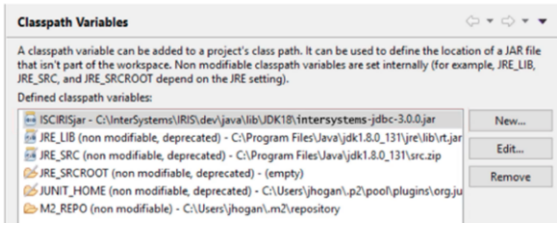
  If InterSystems IRIS is installed locally or in a container on your development system, you can also find the file in *install-dir*\dev\java\lib\JDK18 or *install-dir*/dev/java/lib/1.8, where *install-dir* is the InterSystems IRIS installation directory (*install-dir* in a container is /usr/irissys).

- You have gathered the connection information for the InterSystems IRIS target you want to connect to.

You can copy the completed connection code from the listing that follows the instructions. Be sure to review the instructions in the following section for adding TLS encryption to the connection.

To code a JDBC connection to InterSystems IRIS, follow these steps:

1. Add the InterSystems JDBC driver, **`intersystems-jdbc-`***version*.jar, to your local *CLASSPATH*.



2. Create a main method and import the **`com.intersystems.jdbc.*`** libraries.

```
import com.intersystems.jdbc.*;
import java.sql.Connection;
```

3. Use the connection information for the InterSystems IRIS target to define the connection string, in the form *`host-identifier`*`:`*`superserver-port/namespace`*, and the credentials:

```
public class JDBCConnection {
   public static void main (String[] args) throws Exception {
      String dbUrl = "jdbc:IRIS://127.0.0.1:1972/User"; //replace
      String user = "Admin";
      String pass = "--------";
```

4. Set a new data source using `IRISDataSource` (or the standard driver manager you may have used to connect to other databases). Set the `URL`, `User`, and `Password` using the connection string and credentials for your target.

```
      IRISDataSource ds = new IRISDataSource();
      ds.setURL(dbUrl);
      ds.setUser(user);
      ds.setPassword(pass);
```

5. Finally, create the connection.

```
import com.intersystems.jdbc.*;
import java.sql.Connection;

public class JDBCConnection {
   public static void main (String[] args) throws Exception {
      String dbUrl = "jdbc:IRIS://127.0.0.1:1972/User"; //replace
      String user = "Admin";
      String pass = "--------";

      IRISDataSource ds = new IRISDataSource();
      ds.setURL(dbUrl);
      ds.setUser(user);
      ds.setPassword(pass);
      Connection dbconnection = ds.getConnection();
      System.out.println("Connected to InterSystems IRIS via JDBC.");
   }
}
```

You can copy the code used here from the listing below. For more information about using Java and JDBC with InterSystems IRIS, see Using Java with InterSystems Software.

```
import com.intersystems.jdbc*;
import java.sql.Connection;
public class JDBCConnection{
  public static void main (String[] args) throws Exception {
      String dbUrl =
        "jdbc:IRIS://host-identifier:superserver-port/namespace";
      String user = "username";
      String pass = "password";

      IRISDataSource ds = new IRISDataSource();
      ds.setURL(dbUrl);
      ds.setUser(user);
      ds.setPassword(pass);
      Connection dbconnection = ds.getConnection();
      System.out.println("Connected to InterSystems IRIS via JDBC.");
      }
}
```

4 Securing the JDBC Connection with TLS

To extend the instructions above to code a JDBC connection with TLS encryption using a self-signed X.509 certificate provided by the InterSystems IRIS target, do the following:

1. Obtain or download the self-signed certificate as instructed and place it in a secure location. (If necessary, download the InterSystems JDBC driver, **`intersystems-jdbc-`***version*.jar.)
2. On the operating system command line, issue the following command, supplying a keystore password and confirming `Trust this certificate? [no]: yes` as requested:

   ```
   keytool -importcert -file path-to-cert/cert-file.pem
     -keystore keystore.jks
   ```

3. In the directory containing the Java file in which you are coding the connection, create a configuration file named SSLConfig.properties and including these properties:

   ```
   trustStore=path-to-keystore/keystore.jks
   trustStorePassword=keystore-password
   ```

4. In the code provided above, modify the `IRISDataSource` definition by adding `ds.setConnectionSecurityLevel(10);`, which specifies the use of TLS, to the other connection settings.

You can copy the JDBC connection code **including TLS encryption** from the listing below. For detailed information about connecting with TLS from Java applications, see Configuring Java Clients to Use TLS with InterSystems IRIS.

```
import com.intersystems.jdbc*;
import java.sql.Connection;
public class JDBCConnection{
  public static void main (String[] args) throws Exception {
      String dbUrl =
        "jdbc:IRIS://host-identifier:superserver-port/namespace";
      String user = "username";
      String pass = "password";

      IRISDataSource ds = new IRISDataSource();
      ds.setURL(dbUrl);
      ds.setUser(user);
      ds.setPassword(pass);
     ds.setConnectionSecurityLevel(10);
      Connection dbconnection = ds.getConnection();
      System.out.println("Connected to InterSystems IRIS via JDBC.");
      }
}
```

5 Connect from .NET using ADO.NET

Before using these instructions, you should make sure that:

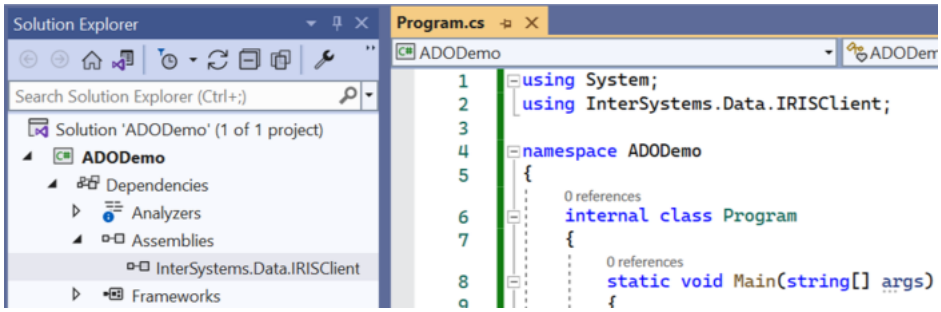
- Your development system has the .NET framework and Visual Studio (or another .NET IDE of your choice) installed, and has network access to your InterSystems IRIS target.
- You have downloaded the InterSystems ADO.NET client assembly, **`InterSystems.Data.IRISClient.dll`**, to your development system.

  If InterSystems IRIS is installed locally or in a container on your development system, you can also find the file in *install-dir*\dev\dotnet\bin\net5.0 (or *install-dir*/dev/dotnet/bin/net5.0 on UNIX®/Linux), where *install-dir* is the InterSystems IRIS installation directory (*install-dir* in a container is /usr/irissys).
- You have gathered the connection information for the InterSystems IRIS target you want to connect to.

You can copy the completed connection code from the listing that follows the instructions. Be sure to review the instructions in the following section for adding TLS encryption to the connection.

To code an ADO.NET connection to InterSystems IRIS, follow these steps:

1. Add the InterSystems ADO.Net client assembly, **`InterSystems.Data.IRISClient.dll`**, as a dependency and declare it in the application, then create a namespace with an internal class that has a main method.



2. Use the connection information for the InterSystems IRIS target to define the connection string:

```
        static void Main(string[] args)
        {
            string connectionString = "Server = localhost; Port = 1972; " +
                "Namespace = User; User ID = Admin; Password = --------";
```

3. Finally, pass the connection string as an argument to the **`IRISADOConnection`** method and create the connection:

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

namespace ADODemo
{
    0 references
    internal class Program
    {
        0 references
        static void Main(string[] args)
        {
            string connectionString = "Server = localhost; Port = 1972; " +
                "Namespace = User; User ID = Admin; Password = --------";
            IRISADOConnection connection = new IRISADOConnection(connectionString);
            connection.Open();
            // when finished, use the line below to close the connection
            // connection.Close();
        }
    }
}
```

You can copy the code used here from the listing below. For more information about using .NET and ADO.NET with InterSystems IRIS, see Using .NET with InterSystems Software.

Note: Other references to this code in the InterSystems documentation may use the **`IRISConnection`** method instead of the **`IRISADOConnection`**; the former is simply an alias to the latter, identical in practice.

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

namespace ADODemo
{
   internal class Program
   {
      static void Main(string[] args)
      {
         string connectionString = "Server = host-identifier; " +
           "Port = superserver-port; Namespace = namespace;" +
           "User ID = username; Password = password";
         IRISADOConnection connection = new IRISADOConnection(connectionString);
         connection.Open();
         // when finished, use the line below to close the connection
         // connection.Close();
      }
   }
}
```

6 Securing the ADO.NET Connection with TLS

To extend the instructions above to code an ADO.NET connection with TLS encryption using a self-signed X.509 certificate provided by the InterSystems IRIS target, do the following. You can copy the completed connection code including TLS encryption from the listing that follows the instructions.

1. Obtain or download the certificate as instructed and place it in a secure location. (If necessary, download the InterSystems ADO.NET client assembly, **`InterSystems.Data.IRISClient.dll`**.)
2. Install the certificate using the instructions for your platform, below. (Both Windows and UNIX/Linux systems provide several ways to install certificates.)
   - On Windows, open a Command Prompt window and enter the following command to add the certificate to the **Trusted Root Certification Authorities** store under **Current User**:

     ```
     certutil -user -addstore Root path-to-certificate\certificate-file.pem
     ```

   - On the UNIX or Linux command line, follow these steps:
     a. Enter the following command to install the Dotnet Certificate Tool, **dotnet-certificate-tool** (see https://github.com/gsoft-inc/dotnet-certificate-tool):

        ```
        dotnet tool install --global dotnet-certificate-tool
        ```

     b. Use the tool to add the certificate to the **Root** store under **CurrentUser** (besure to include **--store-name Root** as shown):

        ```
        certificate-tool add --cert path-to-certificate\certificate-file.pem
          --store-name Root
        ```

        Note: If the certificate is not in PEM format, use the appropriate flag in place of **--cert**; if the certificate is password-protected, include the **--password** flag. For more information, see the README file on GitHub.

3. In the code provided above, modify the connection string definition by adding `SSL = true` to specify the use of TLS.

You can copy the ADO.NET connection code **including TLS encryption** from the listing below. For more detailed information about connecting with TLS from .NET applications, see Configuring .NET Clients to Use TLS with InterSystems IRIS.

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

namespace ADODemo
{
   internal class Program
   {
      static void Main(string[] args)
      {
         string connectionString = "Server = host-identifier; " +
           "Port = superserver-port; Namespace = namespace;" +
           "User ID = username; Password = password; SSL = true";
         IRISADOConnection connection = new IRISADOConnection(connectionString);
         connection.Open();
         // when finished, use the line below to close the connection
         // connection.Close();
      }
   }
}
```

7 Connect from C++ using ODBC

Before using these instructions, you should make sure that:

- Your development system has Visual Studio (or another C++ development environment of your choice) installed, and has network access to your InterSystems IRIS target.
- You have downloaded the platform-specific InterSystems ODBC driver to your development system. (If InterSystems IRIS is installed locally on your development system, the driver is already installed and you do not need to download it.)
- You have gathered the connection information for the InterSystems IRIS target you want to connect to.

You can copy the completed connection code from the listing that follows the instructions. Be sure to review the instructions in the following section for adding TLS encryption to the connection.

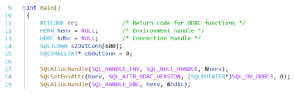
Note: The ODBC driver is also used to connect C applications to InterSystems IRIS.

To code an ODBC connection to InterSystems IRIS, follow these steps:

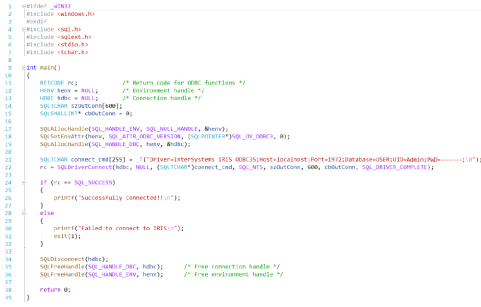
1. If InterSystems IRIS is not installed locally, install the InterSystems ODBC driver as follows:
   - On Windows, execute the downloaded installer, ODBC-*version*-win_x64.exe.
   - On Linux or MacOS:
     a. Create the directory in which you want to install the driver, for example /usr/irisodbc.
     b. Unpack the downloaded .tar file, ODBC-*version-platform*.tar.gz, in that directory.
     c. Execute the ODBCinstall script.

2. Import the required libraries and create a main method:

3. Initialize variables, and allocate the environment and connection handles:



4. Define the connection string, which is a series of key value pairs specifying the connection information for the InterSystems IRIS target (note that the **`Database`** key is used to specify the InterSystems IRIS namespace) and call the method that creates the connection, **`SQLDriverConnect`**, passing in arguments such as the handle variables and connection string.



You can copy the code used here from the listing below. For more information about using C++ and ODBC with InterSystems IRIS, see Using the InterSystems ODBC Driver.

```
#ifdef _WIN32
#include <windows.h>
#endif
#include <sql.h>
#include <sqlext.h>
#include <stdio.h>
#include <tchar.h>

int main()
{
     RETCODE rc;                          /* Return code for ODBC functions */
     HENV henv = NULL;                    /* Environment handle */
     HDBC hdbc = NULL;                   /* Connection handle */
     SQLTCHAR szoutConn[600];
     SQLSMALLINT *cbOutConn = 0;

     SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
     SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER*)SQL_OV_ODBC3, 0);
     SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

     SQLTCHAR connect_cmd[255] = _T("Driver=InterSystems IRIS
ODBC35;Host=host-identifier;Port=superserver-port;Database=namespace;UID=username;PWD=password;\0");
     rc = SQLDriverConnect(hdbc, NULL, (SQLCHAR*) connect_cmd, SQL_NTS, szOutConn, 600, cbOutConn,
SQL_DRIVER_COMPLETE);

     if (rc == SQL_SUCCESS)
     {
                printf("Successfully connected!!\n");
     }
     else
     {
                printf("Failed to connect to IRIS\n");
                exit(1);
     }

     SQLDisconnect(hdbc);
     SQLFreeHandle(SQL_HANDLE_DBC, hdbc);    /* Free connection handle */
     SQLFreeHandle(SQL_HANDLE_ENV, henv);    /* Free environment handle */

     return 0;
}
```

8 Securing the ODBC Connection with TLS

To extend the instructions above to code an ODBC connection with TLS encryption using a self-signed X.509 certificate provided by the InterSystems IRIS target, do the following. You can copy the completed connection code inlcuding TLS encryption from the listing that follows the instructions.

1. Obtain or download the certificate as instructed and place it in a secure location. (If necessary, download the platform-specific InterSystems ODBC driver to your development system.)
2. Choose a location and name for a connection and configuration definitions file and set the environment variable **`ISC_SSLconfigurations`** to the full path (including filename) of this file. The default name and location is C:\Program Files (x86)\Common Files\InterSystems\IRIS\SSLDefs.ini on Windows, however you can place the file where you like as long as you set the environment variable. On UNIX/Linux, you must set the environment variable so the driver can locate the file.
3. Create the definitions file with the following contents:

   ```
   [ConnectionName]
   Address=host-identifier
   Port=superserver-port
   SSLConfig=TLSConfigName

   [TLSConfigName]
   VerifyPeer=1
   VerifyHost=0
   CAfile=path-to-certificate\certificate-file.pem
   TLSMinVersion=minimum-supported-tls-version
   TLSMaxVersion=maximum-supported-tls-version
   ```

   The connection definition includes the host identifier and superserver port for the InterSystems IRIS target that appear in the connection string variable `connect_cmd` in the code above, as well as the name of the TLS configuration definition. In the TLS configuration definition, `VerifyPeer=1` and `VerifyHost=0` indicate that in order to establish an encrypted connection, the client will verify the server's certificate — that is, the self-signed certificate provided by the InterSystems IRIS target, specified in `CAfile` — but will not verify that the `Common Name` or `subjectAlternativeName` fields of the certificate match the host name or IP address as specified in the connection definition. For information about the correct values for `TLSMinVersion` and `TLSMaxVersion`, see Which TLS Versions Does My Instance of InterSystems IRIS Support? For more detailed information about the definitions file and its contents, see Connecting from a Windows Client Using a Settings File.

   Important: Generally speaking, the best practice when using TLS encryption is to require peer verification, so that both server and client must verify the certificate of the other party before establishing an encrypted connection. However, in the limited case presented here, only the client's certificate is verified.

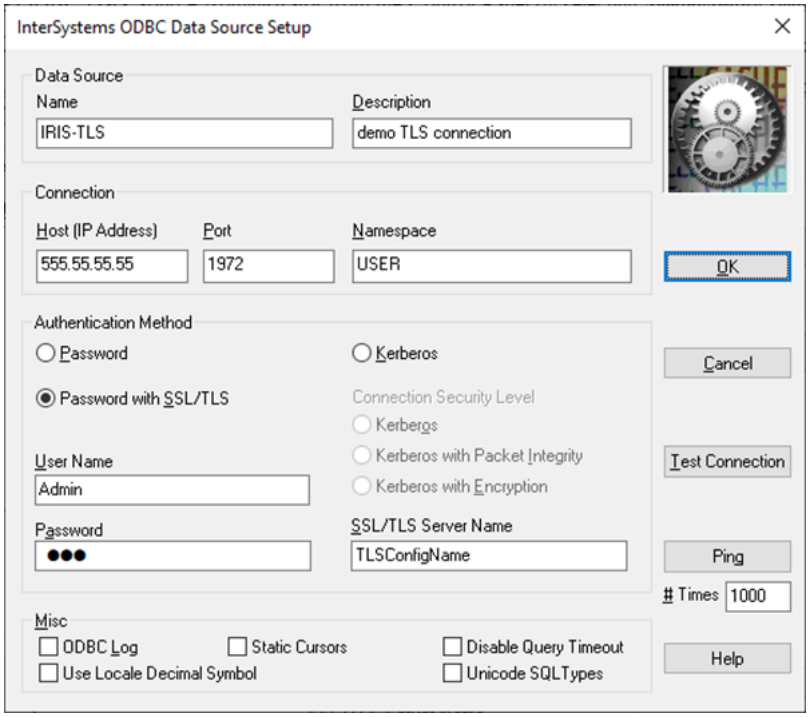
4. Create a DSN entry defining the InterSystems IRIS target as an ODBC data source. This entry contains the connection information for the target, corresponding to the fields of the connection string in the code — host identifier (`Host`), superserver port (`Port`), namespace (`Database`), username (`UID`), and password (`PWD`) . The steps to follow depend on your platform:

   - On Windows:

     a. Open the ODBC Data Source Administrator from the Control Panel by selecting **Administrative Tools** and then **ODBC Data Sources** (32 bit or 64 bit, depending on your system).

     b. On the User DSN or System DSN tab, select **Add**, then on the Create New Data Source panel select **InterSystems IRIS ODBC35**.

     c. Create a name and description for the source, enter the connection information in the appropriate locations, set **Authentication Method** to **Password**, and enter the name of the TLS configuration in the definitions file you created in the previous step as **SSL/TLS Server Name**.



Important: The **Password with SSL/TLS** setting for **Authentication Method** is not required when using a definitions file, such as ssldefs.ini, as described in this procedure; it is included for legacy purposes only. If the definitions file is properly configured, the connection should use SSL/TLS automatically with **Password** as the authentication mechanism. If there are problems with the definitions file, the connection attempt may proceed without using SSL/TLS; if the server allows unencrypted connections, the connection may succeed.

d. Optionally test the connection information using the **Test Connection** button, then click **OK**.

- On Linux:

a. Using the file *extract-directory*/dev/odbc/redist/ssl/irisodbc.ini.template as a template and the connection information, create an ODBC initialization file (typically called odbc.ini) in a location available to the connection code with values as described for the Windows DSN above, as shown here:

```
[ODBC Data Sources]
IRIS-TLS = IRIS-TLS

[IRIS-TLS]
Driver = /home/guest/iris/bin/libirisodbc35.so
Description = demo TLS connection
Host = host-identifier
Port = superserver-port
Namespace = namespace
UID = username
Password = password
Protocol = TCP
Query Timeout = 1
Static Cursors = 0
Trace = off
TraceFile = logfile.log
Service Principal Name = iris/target-domain-name
Authentication Method = 2
Security Level = 10
SSL Server Name = tls-config-in-definitions-file
```

Note: You can find the template file in the same location under the installation directory of an installed instance, which is /usr/irissys in a container.

b. Set the environment variable **ODBCINI** to the full path (including filename) of the ODBC initialization file.

- On UNIX, follow the instructions in Defining an ODBC Data Source on UNIX.

5. In the code provided above, modify the connection string definition to replace the connection information fields with the DSN you defined.

```
SQLTCHAR connect_cmd[255] = _T("DSN=IRIS-TLS;\0");
rc = SQLDriverConnect(hdbc, NULL, (SQLCHAR*) connect_cmd, SQL_NTS,
  szOutConn, 600, cbOutConn, SQL_DRIVER_COMPLETE);
```

You can copy the ODBC connection code **including TLS encryption** from the listing below. For more detailed information about connecting with TLS using ODBC, see Connecting from a Windows Client Using a Settings File.

```
#ifdef _WIN32
#include <windows.h>
#endif
#include <sql.h>
#include <sqlext.h>
#include <stdio.h>
#include <tchar.h>

int main()
{
     RETCODE rc;                          /* Return code for ODBC functions */
     HENV henv = NULL;                    /* Environment handle */
     HDBC hdbc = NULL;                   /* Connection handle */
     SQLTCHAR szoutConn[600]
     SQLSMALLINT *cbOutConn = 0;

     SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &henv);
     SQLSetEnvAttr(henv, SQL_ATTR_ODBC_VERSION, (SQLPOINTER*)SQL_OV_ODBC3, 0);
     SQLAllocHandle(SQL_HANDLE_DBC, henv, &hdbc);

     SQLTCHAR connect_cmd[255] = _T("DSN=IRIS-TLS;\0");
     rc = SQLDriverConnect(hdbc, NULL, (SQLCHAR*) connect_cmd, SQL_NTS,
       szOutConn, 600, cbOutConn, SQL_DRIVER_COMPLETE);

     if (rc == SQL_SUCCESS)
     {
                printf("Successfully connected!!\n");
     }
     else
     {
                printf("Failed to connect to IRIS\n");
                exit(1);
     }

     SQLDisconnect(hdbc);
     SQLFreeHandle(SQL_HANDLE_DBC, hdbc);    /* Free connection handle */
     SQLFreeHandle(SQL_HANDLE_ENV, henv);    /* Free environment handle */

     return 0;
}
```