InterSystems™
IRIS Data Platform

Using the InterSystems ODBC Driver

Version 2019.2
2019-10-10

Using the InterSystems ODBC Driver
InterSystems IRIS Data Platform   Version 2019.2   2019-10-10
Copyright © 2019 InterSystems Corporation
All rights reserved.

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

List of Figures

About This Book

This book describes how to use the InterSystems ODBC driver, which enables you to connect to InterSystems databases from external applications via ODBC, and allows InterSystems products to access external ODBC data sources.

This book covers the following topics:

- Overview — provides an overview of the InterSystems ODBC driver.

- Defining an ODBC Data Source on Windows — describes how to use InterSystems IRIS as a ODBC data source on Windows.

- Defining an ODBC Data Source on UNIX® — describes how to use InterSystems IRIS as a ODBC data source on UNIX®.

- ODBC Installation and Validation on UNIX® Systems — describes tools test and validate ODBC installations. It also provides instructions for stand-alone installation and for custom iODBC installation.

- Logging and Environment Variables — describes some tools you can use to perform troubleshooting.

For more information, try the following sources:

- The book *Using Java with the InterSystems JDBC Driver* includes information on JDBC connectivity to InterSystems IRIS from external data sources (the JDBC equivalent of what is described in this manual).

- The book *Using the SQL Gateway* provides an overview of how the SQL Gateway works with both ODBC and JDBC.

1

Overview

InterSystems provides ODBC drivers to enable you to access InterSystems databases via an ODBC connection. To use ODBC, install and configure the InterSystems ODBC client driver, then define one or more DSNs to refer to InterSystems databases. Your application can use an InterSystems DSN in the same way it would use any other DSN.

This chapter discusses the following topics:

- Installation

- Supported ODBC Driver Managers

- Support for pyodbc Python ODBC Bridge

- Support for Node.js Relational Access

- An Overview of ODBC

1.1 Installation

To use an InterSystems database as an ODBC data source, you should first ensure that the InterSystems ODBC client driver has been installed. The following options are available:

- The InterSystems standard installation installs ODBC driver components by default (as described in the *Installation Guide*).

- If you perform a custom installation, you can select the SQL client only option to install only the ODBC client driver.

You must also define DSNs (Data Source Names) to provide your ODBC-aware applications with information needed to connect to InterSystems databases. Each InterSystems database can be represented by multiple DSNs, each of which can support multiple connections. See "Defining an ODBC Data Source on Windows" or "Defining an ODBC Data Source on UNIX®" for OS-specific instructions on how to perform these tasks.

Note: On Windows, InterSystems IRIS IDs use the Large Number (BigInt) datatype, so ODBC client applications must have Large Number support. For example, instances of Access 2016 previous to build 16.0.7812 will display row data as #Deleted. This may also happen if Large Number support is not turned on in the Access Settings for the current database.

1.2 Supported ODBC Driver Managers

The InterSystems ODBC drivers are compliant with ODBC 3.5.

InterSystems ODBC supports the following ODBC driver managers:

- On Windows: the Microsoft Windows driver manager provided with the operating system.

- On UNIX®: the iODBC driver manager (for use with the Unicode and 8–bit ODBC APIs) and the unixODBC driver manager (for use with the 8–bit ODBC API). See "ODBC Installation and Validation on UNIX® Systems" for more information.

  For questions about other driver managers, contact the InterSystems WorldWide Response Center (WRC).

For more complete information, including specific supported databases, see the online *InterSystems Supported Platforms* document for this release.

1.3 Support for pyodbc Python ODBC Bridge

pyodbc is an open source Python module which implements the DB API 2.0 specification (PEP 249 -- Python Database API Specification v2.0), leveraging ODBC to access the underlying database. InterSystems supports use of pyodbc as a way to access the database from Python using the relational paradigm. For general information, see the pyodbc GitHub site.

1.3.1 Installation

There are several sites with installation information, both for Windows and for Linux and related operating systems:

- pyodbc GitHub site: pyodbc Python ODBC bridge

- pyodbc Wiki: Wiki Home

- Microsoft pyodbc installation: Python SQL Driver - pyodbc

- General Python documentation: Python

The installation process is simple:

- Install Python 2 or 3 (which supports Unicode) via the Python download:

- From a console with Python in the path:

```
pip install pyodbc
```

1.3.2 MacOS X Installation

MacOS X installation is similar to UNIX platforms (see Python Releases for Mac OS X):

- install homebrew

- install unixODBC

- run pip install:

```
pip install --upgrade --global-option=build_ext
--global-option="-I/usr/local/include"  --global-option="-L/usr/local/lib"
--allow-external pyodbc --allow-unverified pyodbc pyodbc
```

1.3.3 Test Program

The following test program demonstrates using pyodbc to access an InterSystems IRIS database. See "Structure of the ODBC Initialization File" for an example listing the connection keywords supported by the InterSystems ODBC driver.

test.py

```
import pyodbc
import time

input("Hit any key to start")

dsn = 'IRIS Samples'
server = '127.0.0.1'
database = 'USER'
username = '_SYSTEM'
password = 'SYS'
#cnxn = pyodbc.connect('DRIVER={InterSystems ODBC35};SERVER='+server+'; PORT='+port+';
DATABASE='+database+';UID='+username+';PWD='+ password)
cnxn = pyodbc.connect('DSN='+dsn+';')
lowptr=cnxn.getinfo(127)
highptr=cnxn.getinfo(136)
#value = PyLong_FromUnsignedLongLong(lowptr)
#print("%#5.8x"% (value))

print ("Connection high pointer: ")
print (format(highptr, '02x'))
print ("Connection high pointer: ")
print("%#5.8x"% (highptr))
print ("Connection low pointer: ")
print("%#5.8x"% (lowptr))
cursor = cnxn.cursor()
start= time.clock()


#Sample select query
cursor.execute("SELECT * from User.Person")
row = cursor.fetchone()
#while row:
#    print(row)
#    row = cursor.fetchone()

end= time.clock()
print ("Total elapsed time: ")
print (end-start)
input("Hit any key to end")
```

The following changes avoid returning Unicode data specifically and just directly return UTF-8 data.

```
cnxn.setdecoding(pyodbc.SQL_CHAR, encoding='raw')
cnxn.setencoding(str, encoding='raw')
```

This uses the narrow driver, which avoids driver managers using UCS-2 or UCS-4 Unicode and the complications of providing a driver that matches how a particular driver manager was built. For other Unicode options, see Unicode in the pyodbc Wiki.

1.4 Support for Node.js Relational Access

The *node-odbc* open source Node.js module enables ODBC database access for Node.js client applications. According to the node-odbc site (https://github.com/wankdanker/node-odbc), the module is intended to be "an asynchronous/synchronous interface for node.js to unixODBC and its supported drivers" but it also works in Windows with the Windows driver manager. InterSystems IRIS supports node-odbc on both platforms.

1.4.1 Dependencies

- *InterSystems ODBC driver*

  This is installed by default when you install InterSystems IRIS.

- *Node.js and npm*

  Make sure Node.js version 8 or later is installed. npm is typically installed with Node.js.

- *node-odbc*

  The node-odbc package is available using npm, or it can be installed locally from Github. Refer to the Github node-odbc site (https://github.com/wankdanker/node-odbc) for more information.

  The following packages are required to build node-odbc:

  – *node-gyp*

    node-odbc is delivered as source and is built by npm commands using node-gyp. If you use npm to install node-odbc you may also get node-gyp installed. If not, refer to the node-gyp site (https://www.npmjs.com/package/node-gyp) for information about how to install it.

    Depending on the OS or Linux distribution, it may be necessary to install development tools that are required by node-gyp to build the node-odbc module. No attempt is made here to document the tools required or how to install them. Refer to node-gyp and node-odbc installation instructions for more information.

  – *Python and related development tools*

    Python is a requirement for node-gyp. At the time of this writing, node-gyp depends on Python 2.7 but that could change in the future as new versions of node-gyp become available.

- *unixODBC (Linux/UNIX only)*

  The unixODBC driver manager is required to use node-odbc on Linux, and is provided as a standard part of most Linux distributions. If not already installed on your system, see the installation instructions for your distribution. It is also available for download from the unixODBC site (http://www.unixodbc.org/).

1.4.2 Installation and Setup

- Make sure all dependencies are installed:

  – Node.js and npm (https://nodejs.org/en/download/) — Make sure Node.js version 8 or later is installed. npm is also required and typically installed with Node.js. Decide whether to install node modules using npm locally or globally. First step for local installation is to define a project folder, go to that folder and run `npm init` (see the example in the following section).

  – node-gyp (https://www.npmjs.com/package/node-gyp) — This package is required to build node-odbc. It makes sense to install node-gyp globally, but a local installation will work. In either case, node-gyp will also require Python 2.7.

  – node-odbc (https://github.com/wankdanker/node-odbc) — Install on your system using instructions included on the linked page. This should probably be installed locally since it needs to be rebuilt for IRIS ODBC.

- Remove UNICODE support and rebuild node-odbc. Edit ./node_modules/odbc/binding.gyp to comment out `UNICODE` in the `defines` array. Save the modified binding.gyp and then in the project folder execute `npm rebuild`.

- Make sure the appropriate InterSystems ODBC DSNs are defined. On Windows, you can use the Data Source Administrator (see "Defining an ODBC Data Source on Windows"). On non-Windows platforms, define the ODBCINI

environment variable to the location of the desired odbc.ini file (see "Defining an ODBC Data Source on UNIX®").
It is also possible to define this in JavaScript before loading the node-odbc module.

1.4.3 Sample Installation and Setup on Ubuntu

This sample assumes that Node.js and npm have been installed on your system. If you use npm to install node-odbc you
may also get node-gyp installed. The node-gyp module and its dependencies are required before you can build node-odbc.

Set up a project folder with npm init

It is okay to just take the defaults for the npm init options.

```
~$ mkdir my_odbc
~$ cd my_odbc
~/my_odbc$ npm init

This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help json` for definitive documentation on these fields and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and   save it as a dependency in the
package.json file.

Press ^C at any time to quit.
package name: (my_odbc)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to /home/your_home/my_odbc/package.json:

{
"name": "my_odbc",
"version": "1.0.0",
"description": "",
"main": "index.js",
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1"
},
"author": "",
"license": "ISC"
}

Is this OK? (yes)
~/my_odbc$
```

Install node-gyp (if not already installed globally)

```
~/my_odbc$ npm ls node-gypnpm ls node-gyp
my_odbc@1.0.0 /home/your_home/my_odbc
    (empty)

~/my_odbc$ npm install node-gyp --save

npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN my_odbc@1.0.0 No description
npm WARN my_odbc@1.0.0 No repository field.

+ node-gyp@3.8.0
added 97 packages from 67 contributors and audited 183 packages in 6.749s
found 0 vulnerabilities

~/my_odbc$
```

Install node-odbc

```
/my_odbc$ npm install odbc

> odbc@1.4.5 install /home/your_home/my_odbc/node_modules/odbc
> node-gyp configure build

make: Entering directory '/home/your_home/my_odbc/node_modules/odbc/build'
CXX(target) Release/obj.target/odbc_bindings/src/dynodbc.o
SOLINK_MODULE(target) Release/obj.target/odbc_bindings.node
COPY Release/odbc_bindings.node
make: Leaving directory '/home/your_home/my_odbc/node_modules/odbc/build'
npm WARN my_odbc@1.0.0 No description
npm WARN my_odbc@1.0.0 No repository field.

+ odbc@1.4.5
added 4 packages from 10 contributors and audited 187 packages in 6.187s
found 0 vulnerabilities

~/my_odbc$
```

The above may generate a large number of warnings but they can be ignored so long as the package was added successfully.

Remove UNICODE support and rebuild node-odbc

Edit ./node_modules/odbc/binding.gyp to remove UNICODE support:

```
$/my_odbc$ nano ./node_modules/odbc/binding.gyp
```

Make sure the defines section looks like this:

```
'defines' : [
# 'UNICODE'
],
```

Now rebuild node-odbc:

```
~/my_odbc$ npm rebuild
```

Again, this command may generate a number of warnings that can be ignored. Review to make sure that no errors were encountered and the new module was successfully linked.

Set up the sample program

You can use the following Javascript code to test the ODBC connection. This step requires a running InterSystems IRIS Server and a properly defined DSN.

```
// update this line to reference the location of irisodbc.ini on your system
  process.env.ODBCINI = process.env.ODBCINI || '/opt/isc/iris/inat/mgr/irisodbc.ini';

  var db = require("odbc")();

  let cn = 'DSN=';

  if (process.platform == "win32") {
    // Windows
    cn += 'Sampleodbc;';
  } else if (process.platform == "darwin") {
    // Mac OS
    cn += 'Userunixodbc;';
  } else if (process.platform == "linux") {
    cn += 'Userunixodbc;';
  }
  console.log(cn);

  db.open(cn, function (err) {
    if (err) {
      return console.log(err);
    }
    console.log('I am connected')     ;

    db.query('select * from sample.person where id<3',  function cb(err, data) {
      if (err) {
```

```
        console.error(err);
      } else {
        console.log(data);
      }
    });
    db.close(function () { });
  });
```

This code assumes that the Sample.Person class is defined and compiled in the namespace specified by the DSN and that it has data with ID values less than 3.

Run the sample

```
~/my_odbc$ node index.js
~/my_odbc$ node sample.js

DSN=Userunixodbc;
I am connected
[ { ID: 1,
  Age: 3,
  DOB: '2015-09-28',
  FavoriteColors: '',
  Name: 'Ulman,George L.',
  SSN: '293-31-5406',
  Spouse: 0,
  Home_City: 'Newton',
  Home_State: 'MI',
  Home_Street: '6958 Main Avenue',
  Home_Zip: '20649',
  Office_City: 'Xavier',
  Office_State: 'NY',
  Office_Street: '7313 Madison Avenue',
  Office_Zip: '73226' },
{ ID: 2,
  Age: 16,
  DOB: '2002-04-07',
  FavoriteColors: 'Green',
  Name: 'Pascal,Vincent A.',
  SSN: '973-94-3185',
  Spouse: 0,
  Home_City: 'Xavier',
  Home_State: 'ND',
  Home_Street: '3788 Madison Drive',
  Home_Zip: '80569',
  Office_City: 'Washington',
  Office_State: 'SC',
  Office_Street: '1206 Second Place',
  Office_Zip: '37389' } ]

~/my_odbc$
```

1.5 An Overview of ODBC

An ODBC system has the following components:

- The *client application* — An application makes calls according to the Microsoft ODBC API. ODBC calls establish a connection from the client to a data source (see the section on "ODBC Connection Details").

- The *ODBC driver manager* — The driver manager accepts calls from applications using the ODBC API and hands them off to a registered ODBC client driver. The driver manager also performs any necessary tasks so that the client application can communicate with the client driver and, ultimately, the database server.

- The *ODBC client driver* — A database-specific application that accepts calls from a client application through the ODBC driver manager and provides communication to the database server. It also performs any ODBC-related data conversions that the application requests.

- The *database server* — The actual database ultimately receiving the calls from the client application. It can be on the same or a different machine than the client driver from which it is receiving calls.

- An *initialization file* — A set of configuration information for the driver manager; depending on the operating system, it may also contain client driver information. On UNIX®, this is an actual file, frequently called odbc.ini. On Windows, it is a registry entry.

Note: For a particular vendor database, that vendor may offer its own version of the ODBC client driver for that platform. Oracle, for example, supplies its own ODBC driver for use with Oracle databases on Windows. This may be preferred in some cases because the vendor driver may take advantage of its knowledge of how the database works internally to optimize performance or enhance reliability.

1.5.1 ODBC Connection Details

For an application to connect to a database via ODBC, the application must generally provide the following connection details:

- Information about the ODBC client driver to use.

- Information on locating and accessing the database. For example, this may include the server on which the database resides and the port to use when connecting to it. The details needed depend upon the database technology.

- Login credentials to access the database, if the database is protected by a password.

In most cases, this information is stored within a DSN, which has a logical name for use within the client application. The DSN may or may not include login credentials, which can also be stored in the database initialization file, or not stored at all.

The DSNs must be registered with the ODBC driver manager.

In practice, a connection is established as follows:

1. A client application includes ODBC calls that attempt to connect to a particular DSN. A client application is linked to an ODBC driver manager, which accepts the calls.

2. The ODBC driver manager reads the initialization file to obtain the location of the ODBC client driver and load the client driver into memory.

3. Once loaded into memory, the ODBC client driver uses the ODBC initialization file to locate connection information for the DSN, as well as other information. Using this information, the client driver connects to the specified database.

4. Having established the connection, the client driver maintains communications with the database server.

2

Defining an ODBC Data Source on Windows

This chapter describes how to create a DSN for an InterSystems database on Windows, which you can do either via the Control Panel or by creating a file DSN.

2.1 Creating a DSN with the ODBC Data Source Administrator

To create a DSN, you can use the Windows ODBC Data Source Administrator to access the InterSystems ODBC Data Source Setup dialog box:

- In the Windows Control Panel, select `Administrative Tools` and click the `ODBC Data Sources` icon (the actual icon name may vary depending on your version of Windows; see "Selecting the Correct ODBC Data Source Administrator Version" below).

- In the Windows ODBC Data Source Administrator dialog, select the `User DSN` tab and click the `Add...` button.

- Select `Intersystems IRIS ODBC` for the ODBC 2.5 driver or `Intersystems IRIS ODBC35` for the ODBC 3.5 driver, and click the `Finish` button.

The following illustration shows an instance of the InterSystems ODBC Data Source Setup dialog box with all required fields filled in:

Figure 2–1: InterSystems IRIS ODBC Data Source Setup Dialog Box
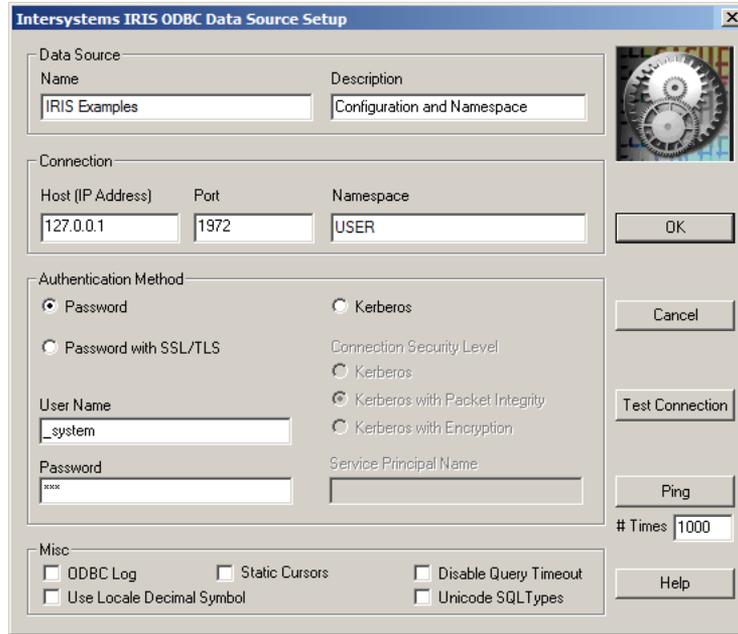
The fields are listed below and are required unless otherwise specified:

Data Source and Connection sections

- `Name` — Specifies the user-defined name of the DSN.

- `Description` — Optional. Provides user-defined information about the DSN.

- `Host IP Address` — Specifies the IP address to be used by the ODBC connection in dotted decimal or dotted quad form, such as "127.0.0.1".

- `Host Port Number` — Specifies the port to be used by the ODBC connection. The default for InterSystems is 51773.

- `Namespace` — Specifies the namespace to use as the ODBC data source.

Authentication Method section

- `Authentication Method` — Select one of the following options, depending on the security used for this database. For detailed information on these options, see "Authentication" in the *Security Administration Guide*.

    - `Password` — authenticate with standard username and password.

    - `Password with SSL/TLS` — authenticate using an SSL/TLS-protected connection (see "Using SSL/TLS" in the *Security Administration Guide*).

    - `Kerberos` — authenticate with Kerberos (see "Configuring for Kerberos Authentication" in the *Security Administration Guide*). For this option, also specify the following settings:

        - `Connection Security Level` — Select `Kerberos`, `Kerberos with Packet Integrity`, or `Kerberos with Encryption`, as appropriate (see the "Client/Server" section of "About Kerberos and the Access Modes" in the *Security Administration Guide*).

        - `Service Principal Name` — Specify the name of the server to be used as a Kerberos principal.

- `User Name` — Optional. Specifies the username to be used by the ODBC connection. By default, this is `_SYSTEM` (not case-sensitive).

- `Password` — Optional. Specifies the password to be used by the ODBC connection. For the default username, the password is `SYS` (must be all upper case).

Misc section (optional settings)

- `ODBC Log` — Optional. If selected, specifies the creation of a log file of ODBC client driver activities for all InterSystems DSNs. This log is for troubleshooting; you should not turn logging on during normal operation as it will dramatically slow down ODBC performance. See "ODBC Logging on Windows" for more information.

- `Static Cursors` — Optional. If selected, enables the InterSystems ODBC client driver's static cursor support. If this flag is off, then the cursor support provided by the ODBC Cursor Library will be used. In general, this flag should be off unless you have a specific reason for not using the ODBC Cursor Library.

- `Disable Query Timeout` — Optional. If selected, causes the ODBC client driver to ignore the value of the ODBC query timeout setting.

  The ODBC query timeout setting specifies how long a client should wait for a specific operation to finish. If an operation does not finish within the specified time, it is automatically cancelled. The ODBC API provides functions to set this timeout value programmatically. Some ODBC applications, however, hard-code this value. If you are using an ODBC application that does not allow you to set the timeout value and the timeout value is too small, you can use the Disable Query Timeout option to disable timeouts.

- `Use Locale Decimal Symbol` — Optional. When selected, specifies the use of the current locale's decimal separator; not checking this sets the decimal separator in the process to a period (`"."`) regardless of the locale. This value can have an affect when the ODBC connection is interoperating with an application that uses the decimal separator as defined for the current locale.

- `Unicode SQL Types` — Optional. This functionality is only relevant if you are working with a multibyte character set, such as in Chinese, Hebrew, Japanese, or Korean locales. If you are only using single-byte character set data, do not select this check box. If selected, this option turns on reporting of a Unicode SQL type (`SQL_WVARCHAR (-9) SQLType`) for string data. This allows some Microsoft applications to allocate the properly sized buffers to hold multibyte data.

  If an application encounters a "SQL data type out of range" error from the Microsoft Driver Manager using **SQLBindParameter**, it can be caused by having selected this check box.

After you have created the DSN, you can use the `Test Connection` button to see if your data source is working correctly.

The `Ping` button attempts to ping the DSN host machine for the number of times specified in the `#Times` field. A popup window will display information on ping success or failure.

Note: **Windows Power Shell Commands**

Windows also offers a set of Power Shell commands for manipulating DSNs from the command line. For details, see the Power Shell documentation for Windows Data Access Components (WDAC).

2.1.1 Selecting the Correct ODBC Data Source Administrator Version

To configure user DSNs on 64-bit Windows, use the Windows Control Panel ODBC Administrator for both 32- and 64-bit programs.

To configure system DSNs for a 32-bit program, run %SystemRoot%\SysWow64\odbcad32.exe.

2.2 Using File DSNs and DSN-less Connections

DSN information is typically stored in the Windows Registry (under [HKLM\SOFTWARE\ODBC]), but you can also specify connection information in a *file DSN* (a text file with extension .dsn).

A file DSN can be created with either the ODBC Data Source Administrator (from the `File DSN` tab) or a standard text editor. For detailed information, see the Microsoft support site (search on `"file DSN"`).

The file DSN can specify the name of an existing DSN to use, for example:

```
[ODBC]
DSN=InterSystems ODBC Sample Code
```

or it can specify a set of key-value pairs that specify the same connection information as a standard registry entry.

A file DSN is invoked by a call to **SQLDriverConnect**.

File DSNs are typically stored in \Program Files\Common Files\ODBC\Data Sources, but you can use the `File DSN` tab in the ODBC Data Source Administrator to define a different default location.

2.2.1 ODBC Connection Strings

SQLDriverConnect takes a connection string argument that can specify connection information in three different ways:

DSN connection

> Specifies the name of a regular DSN in the registry. For example:
>
> `"DSN=ODBC Samples;UID=myUsername;PWD=;"`

FILEDSN connection

> Specifies a file DSN rather than a registry entry. For example:
>
> `"FILEDSN=c:\ODBC_Samples.dsn;UID=myUsername;PWD=;"`

DSN-less connection

> Defines all connection information directly in the connection string. For example:
>
> `"Driver=InterSystems ODBC Driver;Host=127.0.0.1;Port=56772;Database=USER;UID=myUsername;PWD="`

3

Defining an ODBC Data Source on UNIX®

An external application can use InterSystems databases as ODBC data sources. This chapter describes how to create a DSN for an InterSystems database on UNIX®, which you do by editing the ODBC initialization file.

3.1 Structure of the ODBC Initialization File

The ODBC initialization file is used as follows:

- It provides information so that the driver manager can locate and connect to an available DSN, including the path of the ODBC client driver required for that particular connection.

- It defines the DSNs (and optionally includes login credentials for them). The ODBC client drivers use this information.

The following is a sample initialization file for the InterSystems ODBC driver:

```
[ODBC Data Sources]
sampleodbc=sampleodbc

[sampleodbc]
Driver          = /usr/irissys/bin/libirisodbc.so
Description     = InterSystems IRIS ODBC driver
Host            = localhost
Namespace       = USER
UID             = _SYSTEM
Password        = SYS
Port            = 51773
Protocol        = TCP
Query Timeout   = 1
Static Cursors  = 0
Trace           = off
TraceFile       = iodbctrace.log
Authentication Method  = 0
Security Level         = 2
Service Principal Name = iris/localhost.domain.com

[Default]
Driver = /usr/irissys/bin/libirisodbc.so
```

This file includes the following variables:

- ODBC Data Sources — This section lists all DSNs defined in the file the file. Each entry is of the form DSNName=SectionHeading, where *DSNName* is the name specified by the client application and the *SectionHeading* specifies the heading under which DSN information appears in this file.

- *Driver* — Specifies the location of the client driver file to use for this DSN. In this case this is the file libirisodbc.so.

- *Description* — Contains an optional description of the DSN.

- *Host* — Specifies the IP address of the DSN in dotted decimal or dotted quad form, such as "127.0.0.1".

- *Namespace* — Specifies the namespace for the DSN.

- *UID* — Specifies the username for logging into the DSN. By default, this is _SYSTEM (*not* case-sensitive).

- *Password* — Specifies the password for the account specified by the *UID* entry. For default username _SYSTEM, the password is SYS. Unlike the UID, the password *is* case-sensitive.

  Note: Because it is an ODBC standard to allow the storing of usernames and passwords in clear text, the sample initialization file includes the username and password required to access the sample DSN. This is meant merely as an example. A secure ODBC program prompts the user for this information and does not store it, in which case it does not appear in the initialization file at all.

- *Port* — Specifies the port for connecting to the DSN. The default for InterSystems is 51773.

- *Protocol* — Specifies the protocol for connecting to the DSN. For InterSystems, this is always TCP.

- *Query Timeout* — If 1, causes the ODBC client driver to ignore the value of the ODBC query timeout setting.

  The ODBC query timeout setting specifies how long a client should wait for a specific operation to finish. If an operation does not finish within the specified time, it is automatically cancelled. The ODBC API provides functions to set this timeout value programmatically. Some ODBC applications, however, hard-code this value. If you are using an ODBC application that does not allow you to set the timeout value and the timeout value is too small, you can use the Disable Query Timeout option to disable timeouts.

- *Static Cursors* — If 1, enables the InterSystems ODBC client driver's static cursor support. If 0, then the cursor support provided by the ODBC Cursor Library will be used. In general, this flag should be off (that is, set to 0) unless you have a specific reason for not using the ODBC Cursor Library.

- *Trace* — Specifies whether the driver manager performs logging ("on") or not ("off"); by default, logging is off (see "ODBC Logging on UNIX®" for more information).

- *TraceFile* — If logging is enabled by the *Trace* entry, specifies the location of the driver manager log file.

- *Authentication Method* — Specify 0 for password authentication or 1 for Kerberos.

- *Security Level* — Specify this if you use Kerberos for authentication. The allowed values are as follows:

  – 1 = Kerberos

  – 2 = Kerberos with packet integrity

  – 3 = Kerberos with encryption

- *Service Principal Name* — Specify this if you use Kerberos for authentication. This should be the name of the service principal that represents InterSystems.

For more information on Kerberos, see the *Security Administration Guide*.

3.2 Setting up a DSN with odbcinst

A UNIX® ODBC installation includes the program odbcinst. The location is dependent on the install but may be located under /usr/local/bin for example.

There are two template files included with a UNIX® installation located in *install-dir*\dev\odbc\redist\unixodbc. These are:

- odbc.ini_unixODBCtemplate — A sample DSN entry template

- odbcinst.ini_unixODBCtemplate — Intersystems driver template

Edit the template files to suit your configuration. To use them, you can call odbcinst in the following ways:

- To register the driver, specify flags -i -d -f and your odbcinst.ini file. For example:

    ```
    odbcinst -i -d -f   odbcinst.ini_unixODBCtemplate
    ```

- To add a local DSN, specify flags -i -s -h -f and your odbc.ini file. For example:

    ```
    odbcinst -i -s -h -f  odbc.ini_unixODBCtemplate
    ```

- To add a System DSN, specify flags -i -s -l -f and your odbc.ini file. For example:

    ```
    odbcinst -i -s -l -f  odbc.ini_unixODBCtemplate
    ```

From: *install-dir*\dev\odbc\redist\unixodbc\odbcinst.ini_unixODBCtemplate

```
[InterSystems ODBC]
UsageCount=1
Driver=/home/iris/bin/libirisodbc.so
Setup=/home/iris/bin/libirisodbc.so
SQLLevel=1
FileUsage=0
DriverODBCVer=02.10
ConnectFunctions=YYN
APILevel=1
DEBUG=1
CPTimeout=<not pooled>
```

3.3 Setting up SSL Configuration Files

InterSystems IRIS provides two template files for SSL configuration. The files are located in *install-dir*\dev\odbc\redist\ssl. The directory also contains a readme.txt file with further information.

- irisodbc.ini.template — demonstrates how an odbc.ini file entry would be configured for use with an SSL connection.

- odbcssl.ini.template — is an example of an SSL configuration file.

irisodbc.ini.template

This is a sample odbc.ini file with an entry named [SampleSSL] that defines an SSL connection. A working file would typically be named *install-dir*/mgr/irisodbc.ini.

```
[ODBC Data Sources]
SamplesSSL = SampleSSL

[SampleSSL]
Driver = /home/guest/iris/bin/libirisodbc35.so
Description = IRIS ODBC driver
Host = localhost
Namespace = SAMPLES
UID = _SYSTEM
Password = SYS
Port = 1972
Protocol = TCP
Query Timeout = 1
Static Cursors = 0
Trace = off
TraceFile = iodbctrace.log
Service Principal Name = iris/localhost.domain.com

Authentication Method = 2
Security Level = 10
SSL Server Name = SampleSSLConfig
```

In the example above, the last three lines specify the SSL connection. The values must be defined as follows:

- *Authentication Method* must be set to 2.

- *Security Level* must be set to 10.

- *SSL Server Name* must be set to the appropriate named configuration. In this example, SampleSSLConfig is the *SSL Server Name* defined in the following sample file, odbcssl.ini.

odbcssl.ini

This is a sample SSL configuration file. In order for a process to initiate an SSL connection with these values:

- The name of this file (*<path>*/odbcssl.ini) must be specified in environment variable *ISC_SSLconfigurations*.

- The process must be using a DSN that specifies [SampleSSLConfig] as the *SSL Server Name* (as shown in the previous example).

```
[SampleSSLConfig]
CAFile=./CA.cer
CertFile=./Client.cer
KeyFile=./Client.key
Password=MixOfAlphaNumericAndPuncChars!
KeyType=2
Protocols=28
CipherList=ALL:!aNULL:!eNULL:!EXP:!SSLv2
VerifyPeer=1
VerifyDepth=9
```

This example defines the following values:

- *CAFile* — specifies the file containing one or more certificates used to verify the server's certificate.

- *CertFile* — specifies the file containing the client's certificate.

- *KeyFile* — specifies the file containing the client's private key file.

- *Password* — is the client's private key file password, if applicable.

- *KeyType* — specifies the type of private key used by the client.

    - 1 — DSA

    - 2 — RSA (default)

- *Protocols* — specifies which versions of SSL/TLS the client can perform.

    - 1 — SSLv2

    - 2 — SSLv3

    - 4 — TLSv1.0

    - 8 — TLSv1.1

    - 16 — TLSv1.2

    Protocol combinations are specified by adding individual numbers. For example, the default setting is 28 (TLSv1 + TLSv1.1 + TLSv1.2).

- *CipherList* — specifies the list of enabled ciphersuites.

- *VerifyPeer* — specifies the peer certificate verification level.

    - 0 — None (Continue even if certificate verification fails)

    - 1 — Require (Continue only if certificate verification succeeds; default)

- *VerifyDepth* — specifies the maximum number of CA certificates allowed in peer certificate chain.

See "Using SSL/TLS" for detailed information on these values.

3.4 Name and Location of the Initialization File

The initialization file can have any name, but, typically, it is called .odbc.ini when it is located in a user's personal directory, odbc.ini when located in an ODBC-specific directory. The InterSystems sample is called irisodbc.ini and is located in the *install-dir*/mgr directory.

To locate this file, the InterSystems ODBC client driver uses the same search order as iODBC. It looks for the file in the following places, in this order:

1. The file specified by the *ODBCINI* environment variable, if this is defined. When defined, this variable specifies a path and file, such as:

   ```
   ODBCINI=/usr/irissys/irisodbc.ini
   export ODBCINI
   ```

2. The .odbc.ini file in the directory specified by the user's $HOME variable, if *$HOME* is defined and if .odbc.ini exists.

3. If *$HOME* is not defined, the .odbc.ini file in the "home" directory specified in the passwd file.

4. The file specified by the system-wide *SYSODBCINI* environment variable, if this is defined. When defined, this variable specifies a path and file, such as:

   ```
   SYSODBCINI=/usr/irissys/irisodbc.ini
   export SYSODBCINI
   ```

5. The file odbc.ini file located in the default directory for building the iODBC driver manager (/etc/), so that the full path and file name are /etc/odbc.ini.

To use a different odbc.ini file, delete or rename the InterSystems sample initialization file to allow the driver manager to search the *$HOME* or /etc/odbc.ini paths. For example, go to *install-dir*/bin and execute the following command:

```
mv libodbc.so libodbc.so.old
```

and then move your user-defined odbc.ini to etc/odbc, where the driver manager can find it.

4

ODBC Installation and Validation on UNIX® Systems

This chapter provides detailed information about ODBC installation and validation on UNIX® and related operating systems. It discusses the following topics:

- Testing the InterSystems ODBC Configuration — making sure that the InterSystems ODBC driver and the driver manager have been installed and configured correctly..

- Troubleshooting for Shared Object Dependencies — how to validate dependencies on shared objects.

- Performing a Stand-alone Installation — installing the InterSystems ODBC client driver and supported driver manager on UNIX®.

- Custom Installation and Configuration for iODBC — installing and configuring the iODBC driver manager, and configuring PHP for iODBC.

- Key File Names — specific file names of some of important installed components.

Note: The sample ODBC initialization file and test files may include the _SYSTEM/SYS or _system/sys username-password pair in unencrypted form. It is recommended that you remove this data before deployment; it is also recommended that you remove the _SYSTEM account before deployment.

4.1 Testing the InterSystems ODBC Configuration

You should test the ODBC configuration to make sure that the InterSystems ODBC driver and the driver manager have been installed and configured correctly. In addition to tests provided with the unixODBC driver manager, you can run the InterSystems *Select Test program*, which provides specific tests for the InterSystems ODBC driver. The program allows you to specify a DSN and execute a SELECT statement on that connection.

4.1.1 Using the Select Test Program

The InterSystems *select test program* consists of files in the directory *install-dir*/dev/odbc/samples/select

- select.sh — The shell script that runs the test. This script defines the *ODBCINI* environment variable (so that the ODBC initialization file can be found), sets up the search path to find the driver manager, and executes the following SELECT statement:

```
select * from sample.person where ID < 11
```

It then executes the select program using a DSN named sampleodbc. This DSN is defined in the sample ODBC initialization file and points to the InterSystems SAMPLES namespace.

- select — The executable built from select.c. This is a sample ODBC program already linked with the iODBC driver manager.

- select.c — This is the source code for the select program. This source is provided in case you want to make change and compile and link it yourself.

4.1.1.1 Modifying the Shell Script for the SELECT Test

You may need to modify the shell script (select.sh), depending on your configuration:

- The shell script is designed to work with InterSystems login or unauthenticated modes and in Minimal or Normal security installations. It may need modification in other cases.

- By default, the shell script sets up the search paths to find the iODBC driver manager. You would change this if you use the unixODBC driver manager or if you install iODBC in a non-default way.

- The script also assumes that the ODBC initialization file is in the *install-dir*/mgr directory. You should adjust the script as needed to find the ODBC initialization file on your system.

4.1.1.2 Running the SELECT Test

To use the test program:

1. Go to the directory *install-dir*/dev/odbc/samples/select.

2. Execute the test script by typing the following:

   ```
   ./select dsn
   ```

   where *dsn* is the name of the DSN that you want to use in the test.

This test works as follows:

1. The shell script calls the select program.

2. The select program is linked to a driver manager, which reads the ODBC initialization file to get connection information for the given DSN.

3. The driver manager determines the location of the InterSystems ODBC client driver and loads it into memory.

4. The client driver then establishes a TCP/IP connection to the port specified in the ODBC initialization file and is connected to the given namespace using the DSN definition from the ODBC initialization file.

5. Once the connection is established, the client application executes your SELECT statement against the InterSystems database.

4.2 Troubleshooting for Shared Object Dependencies

After installing, you should validate dependencies on other shared objects and correct any problems. The process is as follows:

1. Use the appropriate command to list the dynamic dependencies of the InterSystems ODBC driver.

   For example, on Solaris and other platforms, the command is **ldd**:

```
# ldd install-dir/bin/libirisodbc.so
```

Here *install-dir* is the InterSystems installation directory. If no dependencies are found, you will see a message like the following:

```
libstlport_gcc.so => not found
```

2. If there are no errors, then all dependencies are valid; if there are errors, run the following commands to force the shared object loader to look in the current directory:

```
# sh
# cd install-dir/bin
# LD_LIBRARY_PATH=`pwd`:$LD_LIBRARY_PATH
# export LD_LIBRARY_PATH
```

The **sh** command starts the Bourne shell; the **cd** command changes to the appropriate directory; and the **export** command sets the path to look up shared objects.

Note that on AIX®, you would use *LIBPATH* instead of *LD_LIBRARY_PATH*.

3. Once you have added the current directory to the path, run **ldd** again and check for missing dependencies. If any shared objects cannot be found, add them to the same directory as the ODBC client driver.

4.3 Performing a Stand-alone Installation

By default, a full ODBC installation is performed with a standard InterSystems installation. If you perform a custom installation (as described in the *Installation Guide*), you can select the "SQL client only" option to install only the client access components (ODBC client driver).

In addition, however, a stand-alone installer is provided for InterSystems ODBC. To use this installer:

1. Create the directory where you wish to install the client, such as /usr/irisodbc/.

2. Copy the appropriate zipped tar file into the directory that you just created.

    The ./dist/ODBC/ directory contains zipped tar files with names like the following:

```
ODBC-release-code-platform.tar.Z
```

    where *release-code* is a release-specific code (that varies among InterSystems versions and releases) and *platform* specifies the operating system that the ODBC client runs on.

3. Go to the directory you created and manually unpack the .tar file, as follows:

```
# gunzip ODBC-release-code-platform.tar.Z
# tar xvf ODBC-release-code-platform.tar
```

    This creates bin and dev directories and installs a set of files.

4. Run the **ODBCInstall** program, which will be in the directory that you created. This program creates several sample scripts and configures irisodbc.ini under the mgr directory. For example:

```
# pwd
/usr/irisodbc
# ./ODBCInstall
```

Note: **Identifying the correct platform name**

In some releases, the ./dist/ODBC/ directory contains the following command to display the platform name that identifies the file you need:

```
# ./cplatname identify
```

This command is not present in releases where it is not required.

4.4 Custom Installation and Configuration for iODBC

If you want to build your own iODBC driver manager to operate under custom conditions, you can do so. The iODBC executable and include files are in the directory *install-dir*/dev/odbc/redist/iodbc/. You need to set *LD_LIBRARY_PATH* (*LIBPATH* on AIX®) and the include path in order to use these directories to build your applications.

If you want to customize the iODBC driver manager, you can also do that. Download the source from the iODBC Web site (www.iodbc.org) and follow the instructions.

4.4.1 Configuring PHP with iODBC

You can use InterSystems ODBC functionality in conjunction with PHP (PHP: Hypertext Processor, which is a recursive acronym). PHP is a scripting language that allows developers to create dynamically generated pages. The process is as follows:

1. Get or have root privileges on the machine where you are performing the installation.

2. Install the iODBC driver manager. To do this:

    a. Download the kit.

    b. Perform a standard installation and configuration, as described earlier in this chapter.

    c. Configure the driver manager for use with PHP as described in the iODBC+PHP HOWTO document on the iODBC web site (www.iodbc.org).

    Note that *LD_LIBRARY_PATH* (*LIBPATH* on AIX®) in the iODBC PHP example does not get set, due to security protections in the default PHP configuration. Also, copy libiodbc.so to /usr/lib and run ldconfig to register it without using *LD_LIBRARY_PATH*.

3. Download the PHP source kit from http://www.php.net and un-tar it.

4. Download the Apache HTTP server source kit from http://httpd.apache.org/ and un-tar it.

5. Build PHP and install it.

6. Build the Apache HTTP server, install it, and start it.

7. Test PHP and the Web server using info.php in the Apache root directory, as specified in the Apache configuration file (often httpd.conf). The URL for this is http://127.0.0.1/info.php.

8. Copy the InterSystems-specific initialization file, irisodbc.ini to /etc/odbc.ini because this location functions better with the Apache Web server if the *$HOME* environment variable is not defined.

9. Configure and test the libirisodbc.so client driver file.

10. Copy the sample.php file from the InterSystems ODBC kit to Apache root directory (that is, the directory where info.php is located), and tailor it to your machine for the location of your InterSystems installation directory.

11. You can then run the sample.php program, which uses the SAMPLES namespace, by pointing your browser to
http://127.0.0.1/sample.php

4.5 Key File Names

Depending on your configuration needs, it may be useful to know the specific file names of some of the installed components. In the following lists, *install-dir* is the InterSystems installation directory (the path that $SYSTEM.Util.InstallDirectory() returns on your system).

ODBC driver managers

The *install-dir*/bin/ directory contains the following driver managers:

- libiodbc.so — The iODBC driver manager, which supports both 8-bit and Unicode ODBC APIs.

- libodbc.so — The unixODBC driver manager, for use with the 8-bit ODBC API.

Note: **ODBC on 64-bit UNIX® platforms**

Between releases of the ODBC specification, various data types such as SQLLen and SQLULen changed from being 32-bit values to 64-bit values. While these values have always been 64-bit on iODBC, they have changed from 32-bit to 64-bit on unixODBC. As of unixODBC version 2.2.14, the default build uses 64-bit integer values. InterSystems drivers are available for both 32-bit and 64-bit versions of unixODBC.

InterSystems ODBC client drivers

InterSystems ODBC client drivers are provided for both ODBC 2.5 and ODBC 3.5. The ODBC 3.5 versions will convert 3.5 requests to the older 2.5 automatically, so in most cases either driver can be used. The *install-dir*/bin/ directory contains the following versions (*.so or *.sl):

iODBC-compliant drivers

- libirisodbc — default driver for 8-bit ODBC 2.5

- libirisodbc35 — supports 8-bit ODBC 3.5

- libirisodbciw — supports Unicode ODBC 2.5

- libirisodbciw35 — supports Unicode ODBC 3.5

- libirisodbciw.dylib — supports Unicode ODBC for MAC OS

unixODBC-compliant drivers

- libirisodbcu. — default driver for 8-bit ODBC 2.5

- libirisodbcu35 — supports 8-bit ODBC 3.5

- libirisodbcur64 — supports 8-bit ODBC 2.5 for 64-bit unixODBC

- libirisodbcur6435 — supports 8-bit ODBC 3.5 for 64-bit unixODBC

Other files

The *install-dir*/mgr/irisodbc.ini file is a sample ODBC initialization file.

The files for the test programs are discussed in "Testing the InterSystems ODBC Configuration".

5

Logging and Environment Variables

This chapter describes some tools you can use to perform troubleshooting. It discusses the following topics:

- ODBC Logging on Windows
- ODBC Logging on UNIX®
- ODBC Environment Variables

CAUTION: Enable logging only when you need to perform troubleshooting. You should not enable logging during normal operation, because it will dramatically slow down performance.

5.1 ODBC Logging on Windows

To enable logging for an ODBC data source on Windows, use the *ODBC Data Source Administrator* to change the logging information in the DSN (see "Creating a DSN with the ODBC Data Source Administrator" for detailed usage information). In the ODBC Data Source Administrator, make the following changes:

- To enable logging for the client driver, find the definition of the DSN that you want to log, open it, and check the box labeled `ODBC Log` (or `Log` or variations).

- To enable logging for the driver manager, click the `Tracing` tab and then click the `Start Tracing Now` button. The `Log File Path` field determines the name and location of the trace file.

Note: The default log file name is IRISODBC.log, and the default location is C:\Users\Public\Logs. You can change these values by setting the *IRISODBCTRACEFILE* environment variable (see "ODBC Environment Variables" later in this chapter).

5.2 ODBC Logging on UNIX®

On UNIX®, enable logging for ODBC as follows:

- To enable logging for the client driver, use the *IRISODBCTRACE* environment variable (as described later in "ODBC Environment Variables"). Also configure the ODBC initialization file.

- To enable logging for the driver manager, set the *Trace* entry in the ODBC initialization file (see "Structure of the ODBC Initialization File" in "Using an InterSystems Database as an ODBC Data Source on UNIX®"). In the same file, the *TraceFile* entry specifies the name of the log file to create.

Tip: If you enable logging but the log file is not updated, either you might not have privileges to write to the file or the client application may have loaded the SO before you enabled logging. In the latter case, stop and restart the client application to force it to reload the SO and get the logging flag.

5.3 ODBC Environment Variables

This section describes the environment variables that control the InterSystems ODBC client driver. Typically you use these only for debugging or diagnostics.

IRISODBCDEFTIMEOUT

This variable allows you to specify the duration of a timeout for a default login. Its value is in seconds.

IRISODBCPID

This boolean variable controls the automatic appending of the process ID number to the log file name. Set the value to 1 to enable appending, or 0 to disable. By default, appending is off.

With *IRISODBCPID* enabled, if the base log file is IRISODBC.log and is in your current directory, then the process ID of 21933 generates a full log file name of IRISODBC.log.21933.

Both *IRISODBCPID* and *IRISODBCTRACEFILE* affect the file name. For example, on Windows if you use *IRISODBCTRACEFILE* to set the base file name of the log file (for instance, to C:/home/mylogs/mylog.txt and enable *IRISODBCPID*, then log file names will be of the form C:/home/mylogs/mylog.txt.21965.

IRISODBCTRACE (UNIX® Only)

This boolean variable controls client driver logging. Set the value to 1 to enable client driver logging, or 0 to disable. For more information, see "ODBC Logging on UNIX®" earlier in this chapter.

IRISODBCTRACEFILE

This variable specifies the location and name of the log file. This can be useful for placing the log file in a unique directory or giving it a unique name. The default name for the log file is IRISODBC.log. The default location is as follows:

- For UNIX®, the log is generated in the current directory by default.

- For Windows, the default location for the log file is %PUBLIC%\Logs. This directory is accessible by all users and allows just one location for the log to be created.

IRISODBCTRACETHREADS

This variable controls whether the log also includes threading information. Set the value to 1 to enable inclusion of threading information, or 0 to disable.

It can be useful to enable this additional kind of logging, if you need to debug a threaded application. However, it adds many extra lines to the log for most ODBC applications.