Using Caché with ODBC

Version 2018.1
2019-09-20

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

Using Caché with ODBC
Caché Version 2018.1 2019-09-20

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

List of Tables

About This Book

This book describes how to use InterSystems ODBC, which enables you to connect to InterSystems databases from external applications (such as a development tool or report writer) 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.
- Using an InterSystems database as an ODBC Data Source on Windows — describes how to use an InterSystems database as an ODBC data source on Windows.
- Using an InterSystems database as an ODBC Data Source on UNIX® — describes how to use an InterSystems database as an 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.
- Using the SQL Gateway with ODBC — describes how to use ODBC and the SQL Gateway to link to an external table or stored procedure, and how to migrate data from an external table.
- Logging and Environment Variables — describes some tools you can use to perform troubleshooting..

For more information, try the following sources:

- The book *Using Caché with JDBC* includes information on JDBC connectivity to Caché from external data sources (the JDBC equivalent of what is described in this manual).
- The chapter on "Using the SQL Gateway" in *Using SQL* 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.

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.
- Optionally, you can use the InterSystems stand-alone ODBC installer (described below).

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 "Using an InterSystems Database as an ODBC Data Source on Windows" or "Using an InterSystems Database as an ODBC Data Source on UNIX®" for OS-specific instructions on how to perform these tasks.

1.2 ODBC Driver Support

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).

  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 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.

1.3.1 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

Using an InterSystems Database as 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.

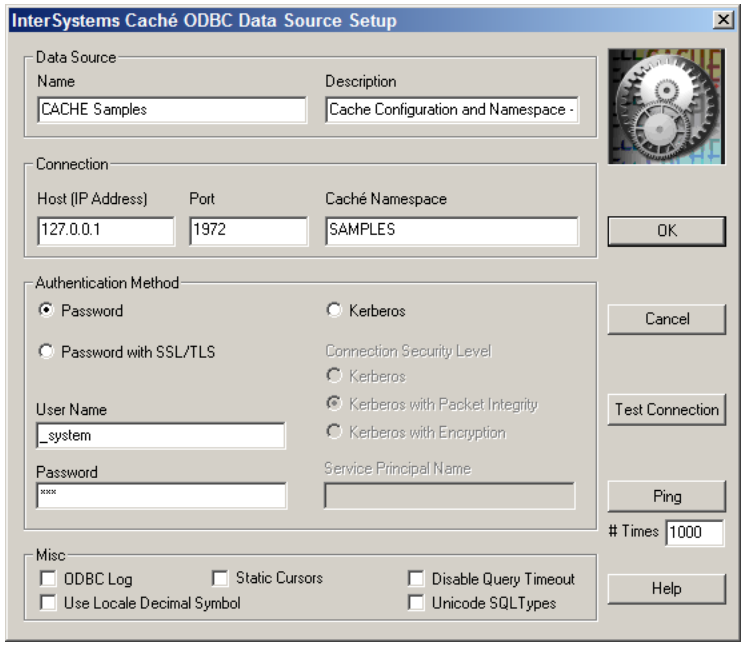
2.1 Creating a DSN with the ODBC Data Source Administrator

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 ODBC` for the ODBC 2.5 driver or `Intersystems 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 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 1972.

Authentication Method section

- `Namespace` — Specifies the namespace to use as the ODBC data source.
- `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 "Enabling logging for ODBC 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.

2.1.1 Selecting the Correct ODBC Data Source Administrator Version

In 64-bit releases of Windows, there are both 32-bit and 64-bit versions of the Data Source Administrator. Select the appropriate version as follows:

- In Windows 8 and higher, the Administrative Tools section of the Control Panel offers two clearly labelled options, `ODBC Data Sources (32-bit)` and `ODBC Data Sources (64-bit)`. Select the appropriate one to create DSNs for 32-bit or 64-bit drivers.
- In 64-bit releases before Windows 8, the Control Panel offers only one option (labeled either `ODBC Data Sources` or `Data Sources (ODBC)`), which will start the 64-bit version. To create a DSN for a 32-bit driver, run the 32-bit Data Source Administrator located at:

%windir%\SysWow64\odbcad32.exe

Note: Both versions of the ODBC Data Source Administrator executable are named odbcad32.exe, but they are in different locations. In Windows releases prior to Windows 8, the 64-bit executable is in %windir%\System32, and the 32-bit executable is in %windir%\SysWOW64 (which may be the opposite of what you expect). Use the one in SysWOW64 to create DSNs based on the 32-bit drivers.

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=Cache ODBC Driver;Host=127.0.0.1;Port=56772;Database=USER;UID=myUsername;PWD="
```

3

Using an InterSystems Database as 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 Configuring 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/cachesys/bin/libcacheodbc.so
Description     = Cache 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
Authentication Method   = 0
Security Level          = 2
Service Principal Name  = localhost.domain.com

[Default]
Driver = /usr/cachesys/bin/libcacheodbc.so
```

This file includes the following variables:

- ODBC Data Sources — Lists all DSNs for 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 libcacheodbc.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" and *is not* case-sensitive.
- *Password* — Specifies the password for the account specified by the *UID* entry. For the SYSTEM username, the password is "SYS" and *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 1972.
- *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 "Enabling logging for ODBC 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 installs 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 \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: \dev\odbc\redist\unixodbc\odbcinst.ini_unixODBCtemplate

```
[InterSystems ODBC]
UsageCount=1
Driver=/home/cache/bin/libcacheodbc.so
Setup=/home/cache/bin/libcacheodbc.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 provides two template files for SSL configuration. The files are located in <Dir>\dev\odbc\redist\ssl. The directory also contains a readme.txt file with further information.

- cacheodbc.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.

cacheodbc.ini.template

This is a sample odbc.ini file with an entry named [SampleSSL] that defines an SSL connection. A working file would be named <installdir>/mgr/cacheodbc.ini.

```
[ODBC Data Sources]
SamplesSSL = SampleSSL

[SampleSSL]
Driver = /home/guest/cache/bin/libcacheodbc35.so
Description = Cache 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 = 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 cacheodbc.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/cachesys/cacheodbc.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/cachesys/cacheodbc.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 <cachesys>/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.

Note: There is also a specific test program for the SQL Gateway. For details, see "Using the UNIX® Gateway Test Program" in the SQL Gateway chapter.

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/libcacheodbc.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/cacheodbc/.
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 cacheodbc.ini under the mgr directory. For example:

```
# pwd
/usr/cacheodbc
# ./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, cacheodbc.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 libcacheodbc.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

- libcacheodbc — default driver for 8-bit ODBC 2.5
- libcacheodbc35 — supports 8-bit ODBC 3.5
- libcacheodbciw — supports Unicode ODBC 2.5
- libcacheodbciw35 — supports Unicode ODBC 3.5
- libcacheodbciw.dylib — supports Unicode ODBC for MAC OS

unixODBC-compliant drivers

- libcacheodbcu. — default driver for 8-bit ODBC 2.5
- libcacheodbcu35 — supports 8-bit ODBC 3.5
- libcacheodbcur64 — supports 8-bit ODBC 2.5 for 64-bit unixODBC
- libcacheodbcur6435 — supports 8-bit ODBC 3.5 for 64-bit unixODBC

InterSystems SQL Gateway drivers

The *install-dir*/bin/ directory contains the following versions of the shared object used by the InterSystems SQL Gateway. This enables you to connect from Caché to other ODBC client drivers. These files are not installed if you perform a stand-alone installation.

linked against iODBC

- cgate.so — supports 8-bit ODBC.
- cgateiw.so — supports Unicode ODBC.

linked against unixODBC

- cgateu.so — supports 8-bit ODBC.
- cgateur64.so — supports 8-bit ODBC for 64-bit unixODBC

Other files

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

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

5

Using the SQL Gateway with ODBC

The SQL Gateway allows Caché to access external databases via both JDBC and ODBC. For a detailed description of the SQL Gateway, see the chapter on Using the SQL Gateway in *Using Caché SQL*.

This chapter discusses the following topics:

- Creating ODBC SQL Gateway Connections for External Sources — describes how to create an ODBC logical connection definition for the SQL Gateway.
- Using the ODBC SQL Gateway Programmatically — discusses how to access an ODBC-compliant database programmatically. This option provides more control over the connection than the setup provided by the standard SQL Gateway wizards.

Note: **Setting the Shared Library Path on UNIX® Systems**

When using third-party shared libraries on a UNIX® system, `LD_LIBRARY_PATH` must be defined by setting the Caché *LibPath* parameter (see "LibPath" in the *Caché Parameter File Reference*). This is a security measure to prevent unprivileged users from changing the path.

5.1 Creating ODBC SQL Gateway Connections for External Sources

Caché maintains a list of SQL Gateway connection definitions, which are logical names for connections to external data sources. Each connection definition consists of a logical name (for use within Caché), information on connecting to the data source, and a username and password to use when establishing the connection. These connections are stored in the table %Library.sys_SQLConnection. You can export data from this table and import it into another Caché instance.

5.1.1 Creating an ODBC SQL Gateway Connection

To define a gateway connection for an ODBC-compliant data source, perform the following steps:

1. Define an ODBC data source name (DSN) for the external database. See the documentation for the external database for information on how to do this.
2. In the Management Portal, go to the System Administration > Configuration > Connectivity > SQL Gateway Connections page.
3. Click **Create New Connection**.

4. On the **Gateway Connection** page, enter or choose values for the following fields:
   - For **Type**, choose **ODBC**.
   - **Connection Name** — Specify an identifier for the connection, for use within Caché.
   - **Select an existing DSN** — Choose the DSN that you previously created. You must use a DSN, since the ODBC SQL Gateway does not support connections without a DSN.
   - **User** — Specify the name for the account to serve as the default for establishing connections, if needed.
   - **Password** — Specify the password associated with the default account.

   For example, a typical connection might use the following values:

Setting	Value
Type	`ODBC`
Connection Name	`ConnectionODBC1`
Select an existing DSN	`MyAccessPlayground`
User	`DBOwner`
Password	`DBPassword`

   Also see "Implementation-specific Options" later in this chapter.
5. Optionally test if the values are valid. To do so, click the **Test Connection** button. The screen will display a message indicating whether the values you have entered in the previous step allow for a valid connection.
6. To create the named connection, click **Save**.
7. Click **Close**.

5.1.2 Creating an ODBC Connection to Caché via the SQL Gateway

Caché provides ODBC drivers and thus can be used as an ODBC data source. That is, a Caché instance can connect to itself or to another Caché instance via ODBC and the SQL Gateway. Specifically, the connection is from a namespace in one Caché to a namespace in the other Caché. To connect in this way, you need the same information that you need for any other external database: the connection details for the database driver that you want to use. This section provides the basic information.

5.1.2.1 Connecting to Caché as an ODBC Data Source

To configure a Caché instance (*Caché_A*) to use another Caché instance (*Caché_B*) as an ODBC data source, do the following:

1. On the machine that is running *Caché_A*, create a DSN that represents the namespace in *Caché_B* that you want to use. (See "Using an InterSystems database as an ODBC Data Source on Windows" or "Using an InterSystems database as an ODBC Data Source on UNIX®" for OS-specific instructions on how to create a DSN.)

   Tip: If *Caché_B* is installed on this machine, a suitable DSN might already be available, because when you install Caché, the installer automatically creates DSNs.

2. Within *Caché_A*, use the SQL Gateway to create an ODBC connection that uses that DSN. Provide the following details:
   - For **Type**, choose **ODBC**.

- **Connection Name** — Specify an identifier for the connection, for use within *Caché_A*.
- **Select an existing DSN** — Choose the DSN that you previously created for *Caché_B*.

For example, a typical connection might use the following values:

Setting	Value
Type	ODBC
Connection Name	Cache2Samples
Select an existing DSN	Cache2Samples

Tip: You do not need to specify **User** and **Password** because that information is part of the DSN itself.

3. Click **Save**.
4. Click **Close**.

5.1.3 Implementation-specific Options

Before you define an SQL gateway connection, you should make sure that you understand the requirements of the external database and of the database driver, because these requirements affect how you define the connection. The following options do not apply to all driver implementations.

Legacy Outer Join

The **Enable legacy outer join syntax (Sybase)** option controls whether the SQL gateway connection will enable you use to use legacy outer joins. Legacy outer joins use SQL syntax that predates the SQL-92 standard. To find out whether the external database supports such joins, consult the documentation for that database.

Needs Long Data Length

The **Needs long data length** option controls how the SQL gateway connection will bind data. The value of this option should agree with the SQL_NEED_LONG_DATA_LEN setting of the database driver. To find the value of this setting, use the ODBC **SQLGetInfo** function. If SQL_NEED_LONG_DATA_LEN equals Y, then select the **Needs long data length** option; otherwise clear it.

Supports Unicode Streams

The **Supports Unicode streams** option controls whether the SQL gateway connection supports Unicode data in streams, which are fields of type LONGVARCHAR or LONGVARBINARY.

- Clear this check box for Sybase. If you are using a Sybase database, all fields you access via the SQL gateway should include only UTF-8 data.
- Select this check box for other databases.

Do Not Use Delimited Identifiers by Default

The **Do not use delimited identifiers by default** option controls the format of identifiers in the generated routines.

Select this check box if you are using a database that does not support delimited SQL identifiers. This currently includes the following databases:

- Sybase
- Informix

- MS SQL Server

Clear the check box if you are using any other database. All SQL identifiers will be delimited.

Use COALESCE

The **Use COALESCE** option controls how a query is handled when it includes a parameter (?), and it has an effect only when a query parameter equals null.

- If you do not select **Use COALESCE** and if a query parameter equals null, the query returns only records that have null for the corresponding value. For example, consider a query of the following form:

  ```
  SELECT ID, Name from LinkedTables.Table WHERE Name %STARTSWITH ?
  ```

  If the provided parameter is null, the query would return only rows with null-valued names.

- If you select **Use COALESCE**, the query wraps each parameter within a `COALESCE` function call, which controls how null values are handled.

  Then, if a query parameter equals null, the query essentially treats the parameter as a wildcard. In the previous example, if the provided parameter is null, this query returns all rows, which is consistent with the behavior of typical ODBC clients.

Whether you select this option depends on your preferences and on whether the external database supports the `COALESCE` function.

To find out whether the external database supports the `COALESCE` function, consult the documentation for that database.

Conversion in Composite Row IDs

The **Conversion in composite Row IDs** option controls how non-character values are treated when forming a composite ID. Choose an option that is supported by your database:

- **Do not convert non-character values** — This option performs no conversion. This option is suitable only if your database supports concatenating non-character values to character values.
- **Use CAST** — This option uses `CAST` to convert non-character values to character values.
- **Use {fn convert ...}** — This option uses `{fn convert ...}` to convert non-character values to character values.

In all cases, the IDs are concatenated with `||` between the IDs (or transformed IDs).

Consult the documentation for the external database to find out which option or options it supports.

5.1.4 Using the UNIX® ODBC SQL Gateway Test Program

Within a full UNIX® Caché installation, you can use a special program to test gateway access from Caché.

Note: The test program uses the default 8-bit iODBC–compliant drivers (libcacheodbc.so and cgate.so). See "Key File Names" in the chapter on "ODBC Installation and Validation on UNIX® Systems" for a complete list of the Caché ODBC client drivers and Caché SQL Gateway drivers available for supported UNIX® platforms.

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

- gatewaytest.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 then accesses a DSN

named `samples`, and executes a routine. This DSN is defined in the sample ODBC initialization file and points to the Caché SAMPLES namespace.

You may need to modify the shell script, depending on your configuration. See the section "Modifying the Shell Script for the SELECT Test" for details.

- SQLGatewayTest.ro — A routine that makes the callout to the Caché SAMPLES namespace using iODBC and the Caché ODBC client driver libcacheodbc.so.

To use the test program:

1. Go to *install-dir*/dev/odbc/samples/
2. Execute the test script by typing the following:

```
./sqlgateway/gatewaytest.sh
```

The gatewaytest.sh script does the following:

1. It starts a Caché session and runs the routine **SQLGatewayTest** in the SAMPLES namespace.
2. This application routine then loads the default Caché SQL Gateway driver, cgate.so, which is linked against the iODBC driver manager.
3. The driver manager loads the client driver using information from the ODBC initialization file.
4. The client driver then establishes a TCP/IP connection to port 1972 and is connected to the Caché SAMPLES namespace using the DSN definition from the ODBC initialization file.
5. The routine executes the following query:

```
SELECT * FROM SAMPLE.PERSON
```

6. The routine then fetches the first ten rows of the result set.

The difference between this example and the simple select test is that in gatewaytest.sh, the Caché process making the initial call is the client application. Typically, a Gateway call from Caché calls the DSN of another vendor's database.

5.2 Using the ODBC SQL Gateway Programmatically

If you require options that are not provided by the standard SQL Gateway wizards, you can use the %SQLGatewayConnection class to access an ODBC-compliant database programmatically. You can either execute a dynamic query (obtaining a result set) or you can perform low-level ODBC programming.

To use this section, you should have some experience with ODBC — this book does not provide details on the ODBC functions. You should also have a basic familiarity with ObjectScript and the InterSystems IDE.

If you encounter any problems, you can monitor the gateway by enabling logging for ODBC (as described in "Logging and Environment Variables").

5.2.1 Creating and Using an External Data Set

To create and use a data set that queries an external database, do the following:

1. Create an instance of %SQLGatewayConnection via the **%New** method.
2. Call the **Connect** method of that instance, passing arguments that specify the ODBC data source name, as well as the username and password that are needed to log into that source, if necessary.

The **Connect** method has the following signature:

```
method Connect(dsn, usr, pwd, timeout) returns %Status
```

Here *dsn* is the DSN for the data source, *usr* is a user who can log into that data source, *pwd* is the corresponding password, and *timeout* specifies how long to wait for a connection.

For more information on connecting, see "Managing the Connection" in the section on Performing ODBC Programming.

3. Create an instance of %ResultSet via the **%New** method, providing the string argument `"%DynamicQueryGW:SQLGW"`.

   Note: Notice that this is slightly different from the argument that you use with a typical dynamic query (`"%DynamicQuery:SQL"`).

4. Invoke the **Prepare** method of the data set. The first argument should be a string that consists of a SQL query, the second argument should be omitted, and the third argument should be the instance of %SQLGatewayConnection.

   This method returns a status, which should be checked.

5. Call the **Execute** method of the data set, optionally providing any arguments in the order expected by the query. This method returns a status, which should be checked.

To use the data set, you generally examine it one row at a time. You use methods of %ResultSet to retrieve information such as the value in a given column.

To advance to the next row, you use the **Next** method; typically you iterate through all the rows until you reach the end, when **Next** returns 0. The **Next** method also returns 0 if an error occurs.

An example follows:

```
ClassMethod SelectAndWrite() as %Status
{
    Set conn=##class(%SQLGatewayConnection).%New()
    Set sc=conn.Connect("AccessPlayground","","")
    If $$$ISERR(sc) do $System.Status.DisplayError(sc) quit

    Set res=##class(%ResultSet).%New("%DynamicQueryGW:SQLGW")
    Set sc=res.Prepare("SELECT * FROM PEOPLE",,conn)
    If $$$ISERR(sc) do $System.Status.DisplayError(sc) quit

    Set sc=res.Execute()
    If $$$ISERR(sc) do $System.Status.DisplayError(sc) quit

    While res.Next()
    { Write !,res.GetData(1)," ",res.GetData(2)," ",res.GetData(3)
    }
    Set sc=conn.Disconnect()
    Quit sc
}
```

For more information on %ResultSet, see the chapter "Dynamic SQL" in *Using Caché SQL*. Also see the class documentation for %ResultSet.

5.2.2 Performing ODBC Programming

If %ResultSet does not provide enough control, you can perform ODBC programming. The %SQLGatewayConnection class provides a set of methods that correspond to ODBC functions, as well as other utility functions. With this class, you can connect to and use an ODBC-compliant database and then perform low-level ODBC programming. The overall procedure is as follows:

1. Create an instance of %SQLGatewayConnection via the **%New** method.
2. Call the **Connect** method of that instance, passing arguments that specify the ODBC data source name, as well as the username and password that are needed to log into that source, if necessary.

   For more information on connecting, see "Managing the Connection."

3. Call the **AllocateStatement** method and receive (by reference) a statement handle.
4. Call other methods of the gateway instance, using that statement handle as an argument. Most of these methods call ODBC functions.

This section discusses the following:

- Null Values and Empty Strings — differences in how Caché and SQL represent null values and empty strings.
- Checking the Status — how to check the status of your activities
- Managing the Connection — properties and methods for managing the connection to the external data source.
- Basic Methods — the basic methods that this class provides as an interface for ODBC functions.
- Getting Information about the Shared Library — how to get information about the currently loaded shared library (for the Caché ODBC SQL Gateway).
- Unloading the Shared Library — how to unload this shared library
- Other Methods — how to use some other methods of the %SQLGatewayConnection class.
- Example — a simple example that executes a query.

Note: This section assumes that you are familiar with ODBC programming.

5.2.2.1 Null Values and Empty Strings

When you use the methods described in this chapter, remember that Caché and SQL have the following important differences:

- In SQL, `""` represents an empty string.
- In Caché, `""` equals null.
- In Caché, `$char(0)` equals an empty string.

5.2.2.2 Checking the Status

Most of the methods of %SQLGatewayConnection return a status, which you should check. Status information is also available via the following properties and methods:

sqlcode property (%Integer)

Contains the SQL code return by the last call (if any).

GatewayStatus property (%Integer)

Indicates the status of the last call. This is one of the following:

- 0 - success
- -1 - SQL error
- -1000 - critical error

GetLastSQLCode() method

```
method GetLastSQLCode() returns %Integer
```

Returns an SQL code for the last call if this call does not return an SQL code (for example, if you used **SQLGetData**).

GatewayStatusGet() method

```
method GatewayStatusGet() returns %Integer
```

Returns an error code for the last call. It does not initialize the error code and can be called multiple times. See the previous notes for the GatewayStatus property.

5.2.2.3 Managing the Connection

The %SQLGatewayConnection class provides properties and methods that you can use to manage the connection to the external data source.

DSN property (%String)

Data source name of the ODBC-compliant data source to which you want to connect.

User property (%String)

Username to log into the data source.

Password property (%String)

Associated password.

ConnectionHandle property (%Binary)

The current connection handle to the ODBC-compliant data source.

Connect() method

```
method Connect(dsn, usr, pwd, timeout) returns %Status
```

Establishes a connection to a DSN. If username and password are both empty, this method calls the ODBC function **SQLDriverConnect**. If that call is unsuccessful or username/password are specified, the method calls the ODBC function **SQLConnect**.

If the timeout parameter is not 0, **SQLSetConnectAttr** is first called to set `SQL_ATTR_LOGIN_TIMEOUT`.

GetConnection() method

```
method GetConnection(conn, timeout) returns %Status
```

Establishes a connection. This method uses an entry from the Caché configuration to determine the DSN, username, and password.

SetConnectOption() method

```
method SetConnectOption(opt, val) returns %Status
```

Invokes the ODBC function **SQLSetConnectAttr**. Only integer values are supported. Integer values for the *opt* argument may be taken from the sql.h and sqlext.h header files.

Disconnect() method

```
method Disconnect() returns %Status
```

Closes the connection.

5.2.2.4 Basic Methods

The following table lists the supported ODBC functions and indicates which methods access those functions. For details on the method arguments, actions, and return values, see the class reference for %SQLGatewayConnection.

Table 5–1: Calling ODBC Functions from %SQLGatewayConnection

ODBC Function	Method That Calls This Function
SQLAllocHandle	**AllocateStatement()**
SQLBindParameter	**BindParameter()**
SQLCloseCursor	**CloseCursor()**
SQLColAttribute	**DescribeCols()**
SQLColumns	**Columns()**
SQLColumnsW	**ColumnsW()**
SQLDescribeCols	**DescribeCols()**
SQLDescribeParam	**DescribeParam()**
SQLDiagRec	**GetErrorList()**
SQLEndTran	**Transact()**
SQLExecute	**Execute()**
SQLFetch	**Fetch()**
SQLFreeHandle	**DropStatement()**
SQLFreeStmt	**UnbindParameters()**
SQLGetData	**GetData()**
SQLGetDataW	**GetDataW()**
SQLGetInfo	**GetInfo()**
SQLMoreResults	**MoreResults()**
SQLNumParams	**DescribeParameters()**
SQLParamData	**ParamData()**
SQLPrepare	**Prepare()**
SQLPrepareW	**PrepareW()**
SQLPrimaryKeys	**PrimaryKeys()**
SQLPrimaryKeys	**PrimaryKeysW()**
SQLProcedureColumns	**ProcedureColumns()**
SQLProcedureColumnsW	**ProcedureColumnsW()**
SQLProcedures	**Procedures()**
SQLPutData	**PutData()**
SQLPutDataW	**PutDataW()**
SQLRowCount	**RowCount()**

ODBC Function	Method That Calls This Function
SQLSetConnectAttr	**SetConnectOption()**
SQLSetStmtAttr	**SetStmtOption()**
SQLSpecialColumns	**SpecialColumns()**
SQLSpecialColumnsW	**SpecialColumnsW()**
SQLTables	**Tables()**
SQLTablesW	**TablesW()**

5.2.2.5 Getting Information about the Shared Library

The %SQLGatewayConnection class provides properties and methods that you can call to get information about the shared library used by the ODBC SQL Gateway.

Note: The phrase *shared library* refers in general to the file or library that comprises the ODBC SQL Gateway. On Windows platforms, this is a file with the extension .dll, but the filename is different on other platforms (see "Key File Names" for a complete list of Caché SQL Gateway shared objects available for supported UNIX® platforms). The properties and methods described here apply in all cases.

DLLHandle property (%Binary)

Handle for the shared library, as currently in use. This is set when you connect.

DLLName property (%String)

Name of the shared library currently in use. This is set when you connect.

GetGTWVersion() method

```
method GetGTWVersion() returns %Integer
```

Returns the current version of the shared library.

GetUV() method

```
method GetUV(ByRef infoval) returns %Status
```

Returns (by reference) whether the shared library was built as Unicode. Note that this method always returns a status of $$$OK.

5.2.2.6 Unloading the Shared Library

The %SQLGatewayConnection class provides a method that you can use to unload the shared library for the ODBC SQL Gateway.

UnloadDLL() method

```
method UnloadDLL() returns %Status
```

Unloads the shared library from the process memory.

5.2.2.7 Other Methods

The %SQLGatewayConnection class provides other utility methods:

FetchRows()

```
method FetchRows(hstmt, Output rlist As %List, nrows As %Integer) returns %Status
```

Returns (by reference) a specified number of rows for the given connection handle. Here *hstmt* is the connection handle, returned (by reference) from **AllocateStatement()**. Also, *rlist* is the returned list of rows; this is a Caché **$list**. Each item in the list contains a row. If there is no data (SQL_CODE = 100), fetching is assumed to be successful but the return list is empty.

CAUTION: This method is primarily useful for testing, and it truncates character fields up to 120 characters so that more fields would fit in a row. Use **GetData()** instead when you need non-truncated data.

GetOneRow()

```
method GetOneRow(hstmt, ByRef row) returns %Status
```

Returns (by reference) the next row for the given connection handle. Here *hstmt* is the connection handle, returned (by reference) from **AllocateStatement()**. Also, *row* is the returned row, a Caché **$list**. Each item in the list contains a field. If there is no data (SQL_CODE = 100), fetching is assumed to be successful but the return list is empty.

CAUTION: This method is primarily useful for testing, and it truncates character fields up to 120 characters so that more fields would fit in a row. Use **GetData()** instead when you need non-truncated data.

GetParameter()

```
method GetParameter(hstmt, pnbr, ByRef value) returns %Status
```

Returns (by reference) the current value of the indicated parameter. Here *hstmt* is the connection handle returned (by reference) from **AllocateStatement()** and *pnbr* is the ordinal number of the parameter.

SetParameter()

```
method SetParameter(hstmt, pvalue, pnbr) returns %Status
```

Sets the value of a previously bound parameter. Here *hstmt* is the connection handle returned (by reference) from **AllocateStatement()**, *pvalue* is the value to use, and *pnbr* is the ordinal number of the parameter. The parameters are stored in **$list** format. If the allocated buffer is not sufficient, a new buffer will be allocated.

5.2.2.8 Example

The following shows a simple example that executes a query:

```
ClassMethod ExecuteQuery(mTable As %String)
{
 set mDSN="DSNtest"
 set mUsrName="SYSDBA"
 set mUsrPwd="masterkey"

 set mx=##class(%SQLGatewayConnection).%New()
 set status=mx.Connect(mDSN,mUsrName,mUsrPwd)
 if $$$ISERR(status) do $System.Status.DisplayError(status) quit $$$ERROR()
 set hstmt=""
 set status=mx.AllocateStatement(.hstmt)
 if $$$ISERR(status) do $System.Status.DisplayError(status) quit $$$ERROR()
 set status=mx.Prepare(hstmt,"SELECT * FROM "_mTable)
 if $$$ISERR(status) do $System.Status.DisplayError(status) quit $$$ERROR()
 set status=mx.Execute(hstmt)
 if $$$ISERR(status) do $System.Status.DisplayError(status) quit $$$ERROR()
 quit mx.Disconnect()
}
```

6

Logging and Environment Variables

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

- Enabling logging for ODBC on Windows
- Enabling logging for ODBC on UNIX®
- InterSystems 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.

When using the SQL Gateway (as discussed in "Using the SQL Gateway with ODBC"), be sure to consult the documentation for the remote database to which you are connecting.

6.1 Enabling Logging for ODBC on Windows

On Windows, to enable logging for an ODBC data source, you generally use the **ODBC Data Source Administrator** screen (within the Windows Control Panel). To access this screen, open the Windows Control Panel, open the **Administrative Tools** subpanel, and then double-click **Data Sources (ODBC)**. Or open the Windows Control Panel and then double-click **ODBC Data Sources**.

Then do the following:

- To enable logging for the client driver, find the definition of the DSN that you want to log. Different kinds of DSN are on different tabs. Click the appropriate tab. Look for a check box labeled **ODBC Log** (or **Log** or variations) and select it.
- 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 location of the trace file.

The details may vary depending on your version of Windows as well as the client driver that you use for this DSN.

Note: The default location of the CacheODBC.log file varies depending on the version of Windows. For Windows Vista and higher, the log will be created in the Public folder under %PUBLIC%\Logs (default path C:\Users\Public\Logs). This folder is accessible by all users and allows just one location for the log to be created. For earlier versions, the log is under %WINDIR% (the C:\Windows or C:\WinNT folder, depending on your version of Windows).

You can change the name and location of the log file by setting the *CACHEODBCTRACEFILE* environment variable (see "InterSystems ODBC Environment Variables" later in this chapter).

6.2 Enabling Logging for ODBC on UNIX®

On UNIX®, enable logging for ODBC as follows:

- To enable logging for the client driver, use the *CACHEODBCTRACE* environment variable (as described later in "InterSystems 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. In the same file, the *TraceFile* entry specifies the name of the log file to create. For information on the initialization file, see "Configuring the ODBC Initialization File."

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.

6.3 InterSystems 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.

CACHEODBCDEFTIMEOUT

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

CACHEODBCPID

This boolean variable enables the automatic appending of the process ID number to the log file name. A value of 1 enables appending and a value of 0 disables it. By default, appending is off.

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

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

CACHEODBCTRACE (UNIX® Only)

This boolean variable enables client driver logging. The default name for this file is CacheODBC.log. For more information on logging, see "Enabling Logging for ODBC on UNIX®" earlier in this chapter.

CACHEODBCTRACEFILE

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 location of the log file is as follows:

- For UNIX®, the log is generated in the current directory by default.
- For Windows platforms previous to Vista, the default location for the log file is %SYSTEMROOT%. For Vista, the default location for the log file is %PUBLIC%\Logs\CacheODBC.log. This directory is accessible by all users and allows just one location for the log to be created.

CACHEODBCTRACETHREADS

This variable controls whether the log also includes threading information. If the variable is 1, threading information is included; if it is 0, threading information is not included.

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.

Note: **Trace Files on Windows 2003**

There are special requirements for setting up the trace file on Windows 2003, specifically for the situation where ODBC is being run by the Web server process. In addition to ensuring that the ODBC client has permission to write to the appropriate logging directory, you need to perform the following procedure:

1. Specify `CACHEODBCTRACEFILE` as `C:\ODBC_Logs\CacheODBC.log`.
2. When specifying the log file information, you also have the option of defining the *CACHEODBCPID* environment variable to include PID information. To do this, create another new variable with a name of `CACHEODBCPID` and a value of 1.
3. Create the directory C:\ODBC_Logs and grant universal write access to this directory.
4. Activate ODBC logging by selecting the **ODBC Log** check box in the DSN setup screen.