CA IDMS - 19.0
Using CA IDMS Server

Date: 15-Jan-2018
This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the "Documentation") is for your informational purposes only and is subject to change or withdrawal by CA at any time. This Documentation is proprietary information of CA and may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA.

If you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Copyright © 2017 CA. All rights reserved. All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.
# Table of Contents

## Using the Client on Windows ................................................................. 9
- ODBC Driver Connect Dialogs ............................................................... 9
- Connecting to a Predefined Data Source ............................................. 9
- Connecting Dynamically to a Data Source Not Previously Defined ........ 10
  - Connection Options ........................................................................ 12
- Configuring JDBC Applications to Use CA IDMS Server ..................... 13
- Using the JDBC Server on Windows ..................................................... 13

## Using the Client on z/OS .................................................................... 14
- Configuring Applications to Use CA IDMS Server ............................... 15
- Controlling the JDBC Server ............................................................... 15
- Monitoring the JDBC Server ............................................................... 17

## Using the Java Client on Any Platform ................................................. 18
- Installing CA IDMS Server on other Platforms .................................... 18
- Using the JDBC Driver ....................................................................... 19
- Using the JDBC Server ..................................................................... 20

## ODBC Programmer Reference ............................................................. 22
- Debugging User Sessions .................................................................... 22
- Error Messages .................................................................................. 22
- ODBC Conformance Levels ................................................................ 23
  - API Conformance Levels ................................................................ 23
    - Core API .................................................................................... 23
    - Level 1 API .............................................................................. 24
    - Level 2 API .............................................................................. 24
  - SQL Conformance Levels ................................................................ 25
- SQL Database Type Mapping Between ODBC and CA IDMS ............... 26
  - CA IDMS to ODBC Data Type Mapping ........................................... 26
  - ODBC to CA IDMS Data Type Mapping .......................................... 27
  - Driver-Specific Data Types .............................................................. 28
Sample SSL Scripts .................................................................................................................................. 67

Windows Registry Information ................................................................. 68
Registry Information ........................................................................................................ 68
HKEY_LOCAL_MACHINE Software ODBC ODBCINST.INI .................................................. 69
HKEY_LOCAL_MACHINE Software ODBC ODBC.INI ......................................................... 70
  Data Source Advanced Options ........................................................................ 71
HKEY_LOCAL_MACHINE Software CA CA IDMS Server ............................................ 75
  Servers ........................................................................................................ 76
  Server server_name ...................................................................................... 76
  Server Advanced Options ........................................................................ 77
  Options ...................................................................................................... 78
  Proxy ........................................................................................................ 83
HKEY_CURRENT_USER Software ODBC ODBC.INI ...................................................... 86
HKEY_CURRENT_USER Software CA CA IDMS Server ........................................... 86

Configuration File Information ................................................................. 86
Configuration Information ..................................................................................... 86
  Environment Variables ............................................................................. 87
  Sections .................................................................................................. 87
    Datasource .......................................................................................... 87
    Server server_name ............................................................................. 90
    Options ............................................................................................... 91
    Proxy .................................................................................................. 94

Properties File Information ................................................................. 96
Setting CA IDMS Server Options as Properties ............................................. 96
  JDBC Driver Options ............................................................................. 97
  JDBC Server Options ............................................................................ 97
  Global Options ....................................................................................... 97
Using CA IDMS Server

This section provides the following information on how to use IDMS Server:
Using the Client on Windows

This section covers the elements of data source connection, and how to use the DriverConnect dialogs to connect to data sources. These dialogs are implemented in the ODBC driver.

ODBC Driver Connect Dialogs

Many ODBC applications use the DriverConnect dialogs to connect to data sources. If your application uses them, the CA IDMS DriverConnect dialogs let you connect to an existing data source, or, in some cases, to connect dynamically to a data source that has not been previously defined.

Although the JDBC driver uses the same types of information, it does not display any dialogs, leaving the collection of such information to the JDBC application. For more information on connecting to a data source using JDBC, see JDBC Programmer Reference.

For more information, see the following topics:
- Connecting to a Predefined Data Source (see page 9)
- Connecting Dynamically to a Data Source Not Previously Defined (see page 10)
- Configuring JDBC Applications to Use CA IDMS Server (see page 13)
- Using the JDBC Server on Windows (see page 13)

Connecting to a Predefined Data Source

Many applications use the Select Data Source dialog to connect to a data source that has been previously defined using the ODBC Administrator dialog. In the Select Data Source dialog, select the desired data source from a list of defined sources and click OK.

The CA IDMS DriverConnect dialog appears, with the name of the data source identified in the Data Source field. This field cannot be changed.
Connecting to a Predefined Data Source

Enter your user ID and password, and, optionally, an account, if your site requires it, in the fields on the CA IDMS DriverConnect dialog. Click OK to connect to the specified data source.

Connecting Dynamically to a Data Source Not Previously Defined

Contents

- Connection Options (see page 12)

Some applications let you connect to a data source dynamically without first adding or defining the data source. If your application supports this, the CA IDMS DriverConnect dialog appears.
Supply the data source connection information to be in effect for the duration of the session. This information is similar to some of the data source definition information specified with the ODBC Administrator dialog. Detailed information for each of the fields is listed below:

- **Dictionary**
  Specifies the DBNAME or segment name of the dictionary containing the definitions of the tables you want to access. This name must be defined in the DBNAME table on the CA IDMS system identified by the server name.

- **Node Name**
  Specifies the Node Name of the system containing the tables you want to access. This is the SYSTEMID specified in the system generation parameters.

- **User ID**
  Specifies a valid user ID for the CA IDMS system.

- **Password**
  Specifies a password field if your system requires it.

- **Task Code**
  Specifies an alternate Task Code to be used for statistics and limit checking. The value you enter must be defined to the CA IDMS system using the TASK system generation statement. If no value is entered, the default Task Code of CASERVER is used.
Connection Options

Specify the following options in the Connection Options section:

- **Account**
  Specifies your account, if your site requires it.

- **Communications Protocol**
  Specifies the Communications interface to be used for the connection. The options are:
  - IDMS -- To establish Wire Protocol connection directly to CA IDMS
  - CCI -- To establish a connection to CA IDMS via CCITCP and CCIENF

- **Host Name**
  Specifies the name or TCP/IP address of either:
  - The CA IDMS CV (where the IDMSJSRV listener is running)
  - The CAICCI host server, overriding the default CAICCI Server name for this connection only.
  The value specified must depend upon the Communications Protocol you select.

- **Port**
  Specifies the TCP/IP port of either:
  - The IDMSJSRV listener running on your CA IDMS CV
  - The CAICCI host server, overriding the default CAICCI Server port for this connection only. Enter 0 to use the default value set by CAICCI, typically 1202.
  The value specified must depend upon the Communications Protocol you select.

- **Wait Timeout**
  Specifies the number of seconds to wait for a reply from the server. This setting overrides the Reply Wait Timeout specified for this Server only. When this limit is exceeded, a communications error is returned and the connection can no longer be used. If multithreading is enabled, the application can continue processing other connections. Options are:
  - Enter 0 to indicate the use of the default value set by CAICCI
  - Enter -1 to indicate an indefinite wait (the largest positive integer)
  - Enter a specific time, in seconds

⚠️ Note: This data source exists only for the duration of the connection.
Configuring JDBC Applications to Use CA IDMS Server

JDBC-enabled applications running on Windows must be able to find the CA IDMS Server executable files, which include both Java classes and native DLLs. Both the startup executable for the Java VM and the native SQL client interface DLLs are installed in the WINDOWS\SYSTEM32 directory, and are always effectively in the PATH. The CLASSPATH must point to idmsjdbc.jar, which is installed in the \Program Files\CA\IDMS Server\Java\lib directory. The sample batch files set this. It is also useful to include a classes subdirectory, such as the \Program Files\CA\IDMS Server\Java\classes, for the caidms.properties file and any updated class files provided by Technical Support.

When running standalone Java applications, the SSL keystore file must be specified to the Java VM. When running Java applications in application servers such as Websphere or Weblogic, see the vendor's documentation on how to specify the keystore file.

Applications can connect to a database using the JDBC DriverManager class with a URL or using a JNDI server with an IdmsDataSource object.

For an applet to use the JDBC driver, the classes must be accessible to web pages accessed from the web server, and the subdirectory containing idmsjdbc.jar should be defined to the web server. For the Microsoft IIS, define a virtual directory pointing to this directory.

Using the JDBC Server on Windows

The JDBC server is installed automatically when the JDBC driver is installed and when using applets must be installed on the same machine as the web server.

The URL used by the applet or other client application identifies the address of the JDBC server. An ODBC data source included in the URL must be a system data source to be recognized by the JDBC server.

⚠️ Note: For a description of the URL recognized by the JDBC driver, see the section "JDBC Programmer Reference."

The Windows version of the JDBC server service wrapper, jsrv.exe, is installed in the Java\bin\Win32 subdirectory. This version invokes the JVM using the Java command (as provided by the JRE or Java Development Kit (JDK) from Sun Microsystems). Configuration settings are maintained in the registry, and can be updated using the CA IDMS ODBC Administrator.

You can start and stop the JDBC server from the CA IDMS Server submenu, which you access from your Start menu.

The Windows version of the JDBC server is controlled like the z/OS and Linux versions. A batch file, jsrv.bat, is installed in the Java\bin subdirectory. This batch file sets the classpath to the idmsjdbc.jar file, and should be run from the Java directory, using one of the following commands:
CA IDMS - 19.0

- **jsrv start**
  Starts the JDBC server as a background process

- **jsrv stop**
  Stops the JDBC server

- **jsrv suspend**
  Suspends the JDBC server

- **jsrv resume**
  Resumes the JDBC server

- **jsrv status**
  Checks the JDBC server status

- **jsrv debug**
  Starts the JDBC server as a foreground process

The NT Service version of the JDBC server is no longer supported because it requires the Microsoft Java VM, which is not compatible with newer versions of Java.

**Note:** For more information, see Controlling the JDBC Server in the section "Using the Client on z/OS (see page 15)." For more detailed command information, see Using the JDBC Driver in the section "Using the Java Client (see page 19)."

# Using the Client on z/OS

This section describes how to use CA IDMS Server in the Unix System Services environment on z/OS. CA IDMS Server supports JDBC-enabled applications running in the USS environment and client applications running on other platforms.

The JDBC driver always runs on the same platform as the client application. Applications running on z/OS use the JDBC driver on z/OS. The Type 2 JDBC driver uses the native SQL client interface to access the CA IDMS system through CAICCI/ENF. The Type 4 JDBC driver uses TCP/IP to communicate directly with CA IDMS r16 SP2 or later.

Remote client applications use a local (from the application's point of view) copy of the JDBC driver, which uses TCP/IP to communicate with the JDBC server on z/OS. The JDBC server acts as a proxy server, calling the native client interface on behalf of the Type 3 JDBC driver. Remote applications can also use the Type 4 JDBC driver to connect directly to CA IDMS r16 SP2 or later.

---

**Important!** Applications running on z/OS do not need the JDBC server to communicate with a CA IDMS system.
Configuring Applications to Use CA IDMS Server

JDBC-enabled applications running on z/OS must be able to find the CA IDMS Server executable files, which include both Java classes and native DLLs. The PATH, LIBPATH, and CLASSPATH environment variables provide this information.

JDBC-enabled applets and applications running on other platforms need only the JDBC driver. The native DLLs are not used on the remote system. The JDBC Driver, idmsjdbc.jar, can be downloaded from the web server with the applet, or can be installed in a directory named in the CLASSPATH environment variable on the remote system.

⚠️ **Note:** For more information, see the section "Using the Java Client."

You can specify settings used by the JDBC driver and JDBC server in the caidms.properties file, which has the same format on all platforms. This includes many settings in the Options section, and all settings in the Proxy section. Options used by the native libraries must be specified in the configuration file. See the section "Properties File Information" for more information.

For an applet to use the JDBC driver, the classes must be accessible to web pages accessed from the web server. These classes are installed in a standard Java archive file, idmsjdbc.jar. The subdirectory containing this file should be defined to the web server. For the IBM HTTP Server, an entry similar to the following can be added to the httpd.conf file:

```plaintext
pass /idmsdir /idmsdir/lib
```

⚠️ **Note:** For more information about setting the required environment variables and defining data sources, see the section "Configuring the Client on z/OS." For information about the URL format, DriverPropertyInfo objects, and DataSource objects used by the JDBC driver, see the section "JDBC Programmer Reference."

Controlling the JDBC Server

Use the JDBC Service wrapper to control the JDBC server with batch jobs or shell commands. Four batch jobs are included in the sampjcl sub-directory of the CA IDMS Server HFS (referred to as /idmsdir):

- **jsrvstrt**
  Starts the JDBC server.

- **jsrvstop**
  Stops the JDBC server.
These files can be customized and copied to an MVS PDS to facilitate job submission from MVS.

These jobs use BPXBATCH to run the corresponding shell scripts. The JDBC server process inherits the CPU time limit from the job that runs BPXBATCH to start it (member JSRVSTRT in the /sampjc1 sub-directory). Set the CLASS and TIME parameters appropriately for the length of time that you plan to keep the JDBC server running.

Each example job has been coded so that the PARM parameter of the EXEC PGM=BPXBATCH statement specifies "nohup" and "& sleep 1." This is an IBM recommendation for running shell commands in batch. For more information, see the z/OS UNIX System Services User's Section.

You can control the JDBC server with shell commands in the following format:

```
jsrv <command> [<jsrv options>] [-jvm] [<java options>]
```

The following commands are used to control the JDBC server:

- `jsrv start`
  Starts the JDBC server as a background process.

- `jsrv stop`
  Stops the JDBC server.

- `jsrv suspend`
  Suspends the JDBC server.

- `jsrv resume`
  Resumes the JDBC server.

- `jsrv status`
  Checks the JDBC server status.

- `jsrv debug`
  Starts the JDBC server as a foreground process.

- `jsrv usage`
  Displays usage information.

These commands are designed to run from the home directory of the CA IDMS Server installation and therefore must be prefixed with "bin/", such as bin/jsrv start. The commands invoke a shell script that sets the required environment variables (described earlier in the section "Configuring the Client on z/OS") and runs the JDBC server service wrapper. The service wrapper starts the Java VM and passes control to the JDBC server entry point. You can also enter the commands in the following form: jsrv.start, jsrv.stop, and so on.

You can override the run-time options from the configuration file by specifying them on the command line.
All options that follow "-jvm" are passed unchanged to the Java VM to allow specification of Java tuning and debugging options.

When started in normal mode, the JDBC server forks a new process and detaches from the terminal. All tracing and debugging is written to the log file specified in the configuration file. When started in debug mode, the JDBC server runs in the foreground and stays attached to the terminal. Pressing Enter shuts down the JDBC server. Tracing output can be displayed on the terminal, redirected to the standard output, or written to the trace file. Messages to the system log can also be echoed on the standard output.

**Note:** For detailed information on the command line options, see Using the JDBC Server in the section "Using the Java Client." For more information on customizing the sample JCL, see the section "Installing the Client on z/OS."

### Monitoring the JDBC Server

The JDBC server sends status messages to the system log or operator console. These messages have a standard format to facilitate monitoring with CA Common Services and other system management products. These messages are identified by message number, which conforms to the standard z/OS message format, $PPPNNNNS$ as follows:

- **PPP**
  - Specifies a product-specific prefix, such as "UJS."

- **NNNN**
  - Specifies a message number, such as "0000-9999."

- **S**
  - Specifies the severity level. The following are valid values:
    - E (Error)
    - W (Warning)
    - I (Information)
    - D (Debugging)

The destination and level of messages written are controlled by settings in the configuration file.

Messages sent include the following:

- UJS00001I - Server started
- UJS0002I - Server stopped
- UJS0003D - Server stopping
Using the Java Client on Any Platform

Versions of CA IDMS Server that use native code are available for z/OS and Windows. The JDBC driver and JDBC server can also be used on other platforms that support Java 1.6 (or later) and TCP/IP.

This section provides information on how to install and use the JDBC driver and server on any Java platform, including Windows, z/OS, and Linux, without installing the native client interface. The procedures are especially suited to the use of the Type 3 or Type 4 JDBC drivers with J2EE application servers.

Installing CA IDMS Server on other Platforms

The CA IDMS Server JDBC driver can be installed on any platform by copying archive files from the CA IDMS Server CD, extracting the needed class or jar files, and setting the CLASSPATH environment variable to point to them. The CA IDMS Server CD contains the following archive files in the \java directory:

- idmsjdbc.tar
  Compiled class files, archived in jar files.

- samples.tar
  Sample Java source files, shell scripts, and input files.
Use the tar utility, or an equivalent such as pax, to extract the needed files on UNIX or Linux. On most platforms and Linux/386, the files can be copied directly from the \java directory on the CA IDMS Server CD. The tar files are supplied as a convenience, the individual jar files included in this directory can be copied directly, if desired.

For more information, see the following topics:
- Using the JDBC Driver (see page 19)
- Using the JDBC Server (see page 20)

**Using the JDBC Driver**

Applications, application servers, and servlets running on platforms other than Windows or z/OS can use the JDBC driver to communicate with a CA IDMS system. CA IDMS Server need not be installed or configured on these platforms. No native methods are used. The JDBC driver uses TCP/IP to communicate directly with the JDBC server running on Windows or z/OS, or directly to CA IDMS r16 SP2 or later. The JDBC server does not need to run on the application platform.

Configuration settings are specified in the caidms.properties file, because the native configuration file is not available. Trace information can be written to a container managed DataSource log or to a file specified in the properties file.

**To use the JDBC driver on other platforms**

1. Extract or copy the JDBC driver, idmsjdbc.jar, to the client machine. For example, on UNIX, assuming you have copied the archive idmsjdbc.tar to the /classes directory:

   ```
   cd /classes
   tar -xovf idmsjdbc.tar idmsjdbc.jar
   ```

2. Update the CLASSPATH environment variable to point to the JDBC driver directory and Java archive file. For example, on Windows:

   ```
   set CLASSPATH=c:\classes;c:\classes\idmsjdbc.jar;%CLASSPATH%
   ```

   On UNIX:

   ```
   set CLASSPATH=/classes:/classes/idmsjdbc.jar:$CLASSPATH
   ```

   **Note:** J2EE application servers have various ways of defining JDBC drivers and specifying how to access the driver jar file. Consult the documentation provided with the application server for details.

3. Specify the system where the JDBC server or CA IDMS r16 SP2 or later is running as part of the URL or the data source object used to connect to the database. For example:

   ```
   jdbc:idms://hostname/datasource
   ```
Using the JDBC Server

The JDBC server can be used as a command line application to support web servers running on platforms other than Windows and z/OS. The JDBC server application is provided as a Java archive file, and is actually the same file used by the JDBC server service on z/OS. Because the native code has not been ported to all platforms, certain limitations apply:

- The service wrapper is not supported. Start and stop the JDBC server by running the JVM, specifying the main class file. It can be run as a background process.

- The configuration file is not supported. Instead, specify options in the properties file or on the command line.

- The native log file is not supported, but trace information can be written to stdout or a file specified in the properties file. Log messages are sent to stderr instead of the syslog daemon.

- The native SQL client is not supported. Connections are routed to CA IDMS using a remote JDBC server running on Windows, z/OS, or directly to CA IDMS r16 SP2 or later which is treated as a remote server.

To use the JDBC server as a command line application

1. Extract the JDBC server Java archive file, idmsjsrv.jar, on the client machine. For example, on UNIX, assuming you have copied the archive to the /classes directory:
   ```
   cd /classes
   tar -xovf idmsjdbc.tar idmsjsrv.jar
   ```
   Update the CLASSPATH environment variable to point to the JDBC server directory and Java archive file. For example:
   ```
   set CLASSPATH=/classes:/classes/idmsjsrv.jar:$CLASSPATH
   ```

2. Start the JDBC server with a command similar to:
   ```
   java ca.idms.proxy.ProxyMain start -h host 1>out 2>err &
   ```
   The parameters are as follows:
   ```
   - host
     Specifies the DNS name or TCP/IP address of the Windows or z/OS machine where the native JDBC server is running, or for CA IDMS r16 SP2 or later, the DNS name or TCP/IP address associated with the TCPIP line of the IDMS system.
   - out
     Specifies the name of the trace file, and err specifies the name of the log file.
   ```

4. Stop the JDBC server with:
java ca.idms.proxy.ProxyMain stop

Options equivalent to those specified in the configuration file on z/OS or using the ODBC Administrator on Windows are specified in the properties file or on the command line:

<table>
<thead>
<tr>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-?</td>
<td>Prints this information</td>
</tr>
<tr>
<td>-h host</td>
<td>Host listener name or IP address</td>
</tr>
<tr>
<td>-p port</td>
<td>Host listener IP port</td>
</tr>
<tr>
<td>-q count</td>
<td>Host listener queue length</td>
</tr>
<tr>
<td>-r host</td>
<td>Remote host name or IP address</td>
</tr>
<tr>
<td>-s port</td>
<td>Remote IP port</td>
</tr>
<tr>
<td>-c</td>
<td>Enables control by remote client</td>
</tr>
<tr>
<td>-e encoding</td>
<td>Overrides platform encoding</td>
</tr>
<tr>
<td>-u</td>
<td>Specifies Unicode fallback encoding</td>
</tr>
<tr>
<td>-w seconds</td>
<td>Client wait timeout interval</td>
</tr>
<tr>
<td>-t seconds</td>
<td>Server reply timeout interval</td>
</tr>
<tr>
<td>-b seconds</td>
<td>Socket blocking timeout interval</td>
</tr>
<tr>
<td>-v [level]</td>
<td>Syslog message level (level = 10 if not specified)</td>
</tr>
<tr>
<td>-l level</td>
<td>Trace log message level</td>
</tr>
<tr>
<td>-d option [option]</td>
<td>Enables debugging with the following trace options, where option can be: trace -- debug tracing native -- native trace snap -- object display buffer -- native buffer display object -- native object display</td>
</tr>
<tr>
<td>-k</td>
<td>Enables SSL client support</td>
</tr>
<tr>
<td>-a</td>
<td>Requires SSL client certificate</td>
</tr>
<tr>
<td>-y</td>
<td>Enables SSL to remote JDBC Server or CA IDMS r16 SP2 or later</td>
</tr>
<tr>
<td>-i class [class]</td>
<td>Includes class in trace</td>
</tr>
<tr>
<td>-x class [class]</td>
<td>Excludes class from trace</td>
</tr>
</tbody>
</table>

**Note:** For detailed information on these options, see the appendices "Configuration File Information" and "Properties File Information."
ODBC Programmer Reference

The ODBC interface allows a Windows application to access different databases using SQL, without specifically targeting any particular database. A module called an ODBC driver is used to link an application to a specific database.

The ODBC interface was developed by Microsoft and is aligned closely with the international-standard ISO Call-Level Interface.

Debugging User Sessions

CA IDMS Server writes messages to the log file specified on the Log and Trace Options tab of the CA IDMS Server Option ODBC Administrator dialog. These messages relay the status of the PC-to-mainframe database connection. Common messages relate to a user’s authorization to sign on to the database, CCI timeouts, and unsuccessful connections because the CV is down.

Error Messages

Error messages returned by the ODBC driver have one of the following formats, depending on the component in which the error is detected:

[CA][IDMS ODBC Driver]Message text...

or

[CA][IDMS ODBC Driver][IDMS]Message text...

The ODBC driver generates the first type of message when it detects an error condition. The second type of message is generated as a result of an error detected within the ODBC data source, which includes CAICCI, the CV, and the network components.

- ODBC Conformance Levels (see page 23)
- SQL Database Type Mapping Between ODBC and CA IDMS (see page 26)
- SQLDriverConnect Connection String Format (see page 28)
- Driver-Specific Connection and Statement Attributes (see page 30)
- Supported Isolation and Lock Levels (see page 30)
- Bulk Insert Support (see page 30)
- Retrieving Network Set Information (see page 31)
- Procedures (see page 31)
- Describe Input (see page 35)
- Positioned Updates (see page 37)
ODBC Conformance Levels

Contents
- API Conformance Levels (see page 23)
  - Core API (see page 23)
  - Level 1 API (see page 24)
  - Level 2 API (see page 24)
- SQL Conformance Levels (see page 25)

CA IDMS Server conforms to the ODBC 3.5 standard. It also provides the functions defined in the ODBC 2.5 specification to continue support for older applications.

Unless otherwise noted, all descriptions of ODBC in this document refer to ODBC 3.5. Microsoft ODBC documentation specifies ODBC conformance in two areas: ODBC API conformance and ODBC SQL conformance. A driver must support all functionality in a conformance level in order to claim conformance to that level, but is not restricted from supporting some of the functionality of higher levels. ODBC defines functions that allow an application to determine the functionality supported by a driver in detail, including the API and SQL conformance levels, specific API function, data type, and scalar function support.

API Conformance Levels

The ODBC 3.5 API includes three conformance levels:

- **Core API**
  The Core API provides the minimum services to support dynamic SQL, including connection establishment and termination, SQL statement execution, retrieval of results, and transaction control. The features in the Core level correspond to those defined in the ISO CLI specification and to the non-optional features defined in the X/Open CLI specification.

- **Level 1**
  Supports Core functionality plus an extended set of features.

- **Level 2**
  Supports Core API and Level 1 functionality, as well as an extended set of features.

The conformance of an ODBC driver is based on its ability to support functions, descriptor fields, and attributes of statement and connection objects. For more information, see the ODBC Programmer’s Section, available from Microsoft.

Core API

The CA IDMS Server ODBC driver supports all Core 3.5 API functions, descriptor fields and statement attributes. It supports all connection attributes with the exception of the following:

SQL_ATTR_TRANSLATE_LIB
Level 1 API

The CA IDMS Server ODBC driver supports the following Level 1 API functions, descriptor fields and attributes.

Functions:
- SQLMoreResults
- SQLPrimaryKeys
- SQLProcedureColumns
- SQLProcedures

Descriptor Fields:
- SQL_DESC_BASE_TABLE_NAME
- SQL_DESC_ROWVER
- SQL_DESC_SCHEMA_NAME
- SQL_DESC_TABLE_NAME

Connection Attributes:
- SQL_ATTR_AUTOCOMMIT
- SQL_ATTR_TXN_ISOLATION

Statement Attributes:
- SQL_ATTR_MAX_LENGTH
- SQL_ATTR_MAX_ROWS
- SQL_ATTR_ROW_OPERATION_PTR

Level 2 API

The CA IDMS Server ODBC driver supports the following Level 2 API functions, descriptor fields and attributes.

Functions:
- SQLDescribeParam
- SQLForeignKeys

Descriptor Fields:
- SQL_DESC_LABEL
SQL Conformance Levels

ODBC 3.5 defines a minimum SQL grammar, which is a subset of the entry level of the ISO/IEC 9075 (or ANSI X3.135-1992) standard, commonly referred to as SQL-92. ODBC drivers must support at least this minimum grammar. A driver and its underlying DBMS may also implement additional features to comply with conformance levels of the SQL-92 standard itself: entry, intermediate or full. Applications can query a driver’s capabilities using the SQLGetInfo function.

CA IDMS Server conforms to the SQL-92 entry level, but also supports some higher-level features as well. For more information, refer to Appendix A of the CA IDMS SQL Reference Section.

In some instances, CA IDMS SQL syntax differs from the SQL-92 standard, as shown in the following table.

<table>
<thead>
<tr>
<th>SQL Statement</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLE clauses:</td>
<td>DEFAULT -- CA IDMS supports WITH DEFAULT, and allows default values of NULL, 0, or blank. UNIQUE -- CA IDMS does not support specification of uniqueness constraints at column or table level. A unique index can be defined to provide the same effect. PRIMARY KEY -- CA IDMS does not support specification of a primary key at column or table level. A unique index can be defined to provide the same effect. REFERENCES -- CA IDMS does not support specification of referential constraints on the CREATE TABLE statement, at column or table level. CREATE CONSTRAINT statement can be used to define referential constraints. CHECK -- CA IDMS does not support specification of CHECK constraints at column level. CHECK constraints can be specified at table level.</td>
</tr>
<tr>
<td>DROP TABLE</td>
<td>RESTRICT -- CA IDMS supports CASCADE, but does not support the RESTRICT keyword. The absence of CASCADE implies RESTRICT.</td>
</tr>
<tr>
<td>GRANT UPDATE (column-list) REFERENCES (column-list)</td>
<td>CA IDMS does not support column level security. CA IDMS driver removes the column list and grants UPDATE to all columns of the table.</td>
</tr>
<tr>
<td>REVOKE</td>
<td>CASCADE/RESTRICT -- CA IDMS does not support the CASCADE and RESTRICT options on REVOKE.</td>
</tr>
</tbody>
</table>
CA IDMS supports:

ISO/IEA/ANSI standard outer join syntax beginning with r17; prior to this outer joins were supported only with the CA IDMS proprietary PRESERVE parameter. If an outer join is coded within an escape sequence, the ODBC driver converts the escape sequence to spaces and pass the statement unchanged to CA IDMS. Scalar functions in escape sequences are supported in the same manner. SQL statements submitted in batch jobs are not supported.

CA IDMS supports data types that map to all ODBC data types.

**SQL Database Type Mapping Between ODBC and CA IDMS**

Contents

- CA IDMS to ODBC Data Type Mapping (see page 26)
- ODBC to CA IDMS Data Type Mapping (see page 27)
- Driver-Specific Data Types (see page 28)

The following tables describe how ODBC data types map to CA IDMS database data types. The tables organize the data types by SQL conformance level. You can also use the SQLGetTypeInfo ODBC function to return detailed information on the mapping of ODBC and CA IDMS data types.

### CA IDMS to ODBC Data Type Mapping

The following chart shows how CA IDMS data types map to ODBC data types:

<table>
<thead>
<tr>
<th>CA IDMS Data Type</th>
<th>ODBC Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BINARY</td>
<td>SQL_BINARY</td>
</tr>
<tr>
<td>CHAR</td>
<td>SQL_CHAR</td>
</tr>
<tr>
<td>CHARACTER VARYING (VARCHAR synonym)</td>
<td>SQL_VARCHAR</td>
</tr>
<tr>
<td>DATE</td>
<td>SQL_TYPE_DATE</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>SQL_DECIMAL</td>
</tr>
<tr>
<td>DOUBLE PRECISION*</td>
<td>SQL_DOUBLE</td>
</tr>
<tr>
<td>FLOAT*</td>
<td>SQL_FLOAT</td>
</tr>
<tr>
<td>GRAPHIC (DBCS Disabled)</td>
<td>SQL_BINARY</td>
</tr>
<tr>
<td>GRAPHIC (DBCS Enabled)</td>
<td>CAID_GRAPHIC</td>
</tr>
<tr>
<td>INTEGER</td>
<td>SQL_INTEGER</td>
</tr>
<tr>
<td>LONGLONG</td>
<td>SQL_BIGINTEGER</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>SQL_NUMERIC</td>
</tr>
<tr>
<td>REAL*</td>
<td>SQL_REAL</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SQL_SMALLINT</td>
</tr>
</tbody>
</table>
## ODBC to CA IDMS Data Type Mapping

The following chart shows how ODBC data types map to CA IDMS data types:

<table>
<thead>
<tr>
<th>ODBC Data Type</th>
<th>CA IDMS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAID_GRAPHIC - DBCS Enabled</td>
<td>GRAPHIC</td>
</tr>
<tr>
<td>CAID_VARGRAPHIC - DBCS Enabled</td>
<td>VARGRAPHIC</td>
</tr>
<tr>
<td>SQL_BINARY</td>
<td>BINARY</td>
</tr>
<tr>
<td>SQL_LONGVARBINARY</td>
<td>BINARY</td>
</tr>
<tr>
<td>SQL_CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>SQL_TYPE_DATE</td>
<td>DATE</td>
</tr>
<tr>
<td>SQL_DECIMAL</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>SQL_DOUBLE</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>SQL_FLOAT*</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>SQL_GUID</td>
<td>CHAR</td>
</tr>
<tr>
<td>SQL_REAL*</td>
<td>REAL</td>
</tr>
<tr>
<td>SQL_INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>SQL_BIGINT</td>
<td>LONGINT</td>
</tr>
<tr>
<td>SQL_NUMERIC</td>
<td>NUMERIC</td>
</tr>
<tr>
<td>SQL_BIT</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>SQL_SMALLINT</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>SQL_TINYINT</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>SQL_TYPE_TIME</td>
<td>TIME</td>
</tr>
<tr>
<td>SQL_TYPE_TIMESTAMP</td>
<td>TIMESTAMP</td>
</tr>
<tr>
<td>SQL_LONGVARCHAR</td>
<td>VARCHAR</td>
</tr>
</tbody>
</table>
Driver-Specific Data Types

When DBCS processing is enabled, the CA IDMS GRAPHIC and VARGRAPHIC data types are mapped to driver-specific ODBC SQL data types, as allowed by the ODBC 3.5 specification. These types are defined as CAID_GRAPHIC and CAID_VARGRAPHIC in the CAIDOOPT.H header file which is installed in the CA IDMS Server directory. These data types are returned by SQLColumns, SQLDescribeCol, and SQLColAttributes, and they should be used with SQLBindParameter to define input parameters for GRAPHIC and VARGRAPHIC columns.

Since most applications are not specifically designed to handle DBCS data as defined by CA IDMS, these types are treated in the same manner as SQL_CHAR and SQL_VARCHAR. The default C type for both is SQL_C_CHAR, and the precision is specified in bytes.

Note: The length on CA IDMS is specified in DBCS characters, which is half the precision specified using the ODBC driver.

When DBCS is not enabled, GRAPHIC and VARGRAPHIC are both mapped to SQL_BINARY, with a default C type of SQL_C_BINARY and precision equal to the length in bytes.

SQLDriverConnect Connection String Format

Contents
- Supported Attribute Keywords and Attribute Values (see page 29)

CA IDMS Server supports additional keywords for the SQLDriverConnect connection string.

The connection string takes one of the following forms:

DSN=\data_source_name\;[\attribute=value[;attribute=value]...]  
DRIVER={CA-IDMS};[\attribute=value[;attribute=value]...]
Supported Attribute Keywords and Attribute Values

The following table provides a summary of the connection string attribute keywords and attribute values supported on the SQLDriverConnect function. This table includes both the keywords defined as part of the Microsoft ODBC specification and those defined as extensions for CA IDMS Server. These keywords correspond to the fields in the DriverConnect dialogs as well as to the information used to define data sources and servers in the ODBC Administrator.

<table>
<thead>
<tr>
<th>Keyword Defined By</th>
<th>Attribute Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSN</td>
<td>Microsoft</td>
</tr>
<tr>
<td>DRIVER</td>
<td>Data source name</td>
</tr>
<tr>
<td></td>
<td>(cannot use with DSN)</td>
</tr>
<tr>
<td>NODE</td>
<td>Dictionary name (use with DRIVER only)</td>
</tr>
<tr>
<td>TASK</td>
<td>Node name (use with DRIVER only)</td>
</tr>
<tr>
<td>UID</td>
<td>Alternate task code (use with DRIVER only)</td>
</tr>
<tr>
<td>PWD</td>
<td>User ID</td>
</tr>
<tr>
<td>ACCT</td>
<td>Password</td>
</tr>
<tr>
<td>CCINAME</td>
<td>Account information, if used</td>
</tr>
<tr>
<td>CCIPORT</td>
<td>CAICCI host server name or IP address (optional, use with DRIVER only). Presence of this option dictates the use of the 'CCI' communications protocol. Either an IPv4 or an IPv6 address can be specified.</td>
</tr>
<tr>
<td>WAIT</td>
<td>CAICCI reply wait timeout (optional, use with DRIVER only)</td>
</tr>
<tr>
<td>HOST</td>
<td>DNS host name or IP address of the target IDMS CV (optional, use with DRIVER only). Presence of this option dictates the use of the 'IDMS' communications protocol. Either an IPv4 or an IPv6 address can be specified.</td>
</tr>
<tr>
<td>PORT</td>
<td>TCP/IP port number of the IDMSJSRV listener (optional, use with DRIVER only). This option is required when the HOST option is specified.</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer indicator (optional, use with DRIVER only)</td>
</tr>
<tr>
<td>PROGRA</td>
<td>Program name (optional, use with DRIVER only)</td>
</tr>
</tbody>
</table>

The following is an example of a connection string for CA IDMS Server:

DSN=CA IDMS database;UID=JELKA01;PWD=XYZZY;ACCT=R45-87

**Note:** For more information, see the following:
Microsoft *ODBC Programmer's Reference* for more information on calling the SQLDriverConnect function.

- Online help and the section "Using the Client on Windows" for more information on the DriverConnect dialog.
- The section "Configuring the Client on Windows" for more information on attribute values.

**Driver-Specific Connection and Statement Attributes**

The ODBC options that can be specified for a data source using the ODBC Administrator can also be specified during program execution using SQLSetConnectAttr and SQLSetStmtAttr.

You can use SQLSetConnectAttr to set the External Identity when connected to CA IDMS with the IDMS wire protocol. The External Identity is sent to CA IDMS when the next transaction starts if it has changed.

These options and their values are defined in IDMSATTR.H, installed in the CA IDMS Server directory.

**Supported Isolation and Lock Levels**

Transaction isolation is set with the SQLSetConnectOption ODBC API function. The default transaction isolation can be set using the ODBC Administrator. The ability to set the default transaction isolation is an IDMS extension. The ODBC driver supports the following two transaction isolation levels:

- **SQL_READ_COMMITTER**
  (Default) Corresponds to the SET TRANSACTION CURSOR STABILITY CA IDMS SQL Statement.

- **SQL_READ_UNCOMMITTED**
  Corresponds to the SET TRANSACTION TRANSIENT READ CA IDMS SQL Statement.

**Bulk Insert Support**

CA IDMS Server supports the ODBC 3.5 Core and Level 1 API functions listed in the API Conformance Levels section earlier in this section. The functions SQLSetStmtAttr and SQLMoreResults can be used to facilitate Bulk Inserts. To ensure that the ODBC driver takes advantage of the CA IDMS INSERT...BULK feature, use parameter markers ("?"") in the VALUES clause of the INSERT statement. Do not use a combination of parameter markers and constant values.
Retrieving Network Set Information

You can use the SQLExecuteDirect function with the following syntax to return information on network sets used to join network records accessed as SQL tables.

\$SETS owner table table

The parameters are as follows:

- **owner**
  Specifies the name of the SQL schema containing the names of the dictionary and network schema where the records are defined. This value applies to all tables and appears to the ODBC application as the TABLE_OWNER returned by SQLTables.

- **table**
  Specifies the name of a record in the network schema. Enter from zero to two `table` arguments. Each `table` argument must be unique and must be defined in the same network schema. This value appears to the ODBC application as the TABLE_NAME returned by SQLTables.

The `owner` and `table` name arguments are case-sensitive. The following list identifies the contents of the result set, which depends on what you specify for the `table` arguments:

- If you specify no `table` arguments, the result set contains a list of all sets in the network schema referenced by `owner`
- If you specify one `table` argument, the result set contains a list of all sets in the network schema referenced by `owner` in which `table` is either the owner or a member
- If you specify two `table` arguments, the result set contains a list of all sets in the network schema referenced by `owner` between the two tables, where either is the owner or member

The result columns are described in the following table. All columns are defined as VARCHAR(18):

- **SET_NAME**
  Network set name
- **SCHEMA_NAME**
  SQL schema name (ODBC owner)
- **OWNER_NAME**
  Network owner record name (ODBC table)
- **MEMBER_NAME**
  Network owner record name (ODBC table)
CA IDMS supports procedures and table procedures. CA IDMS procedures are used like procedures supported by other data bases. A table procedure is a CA IDMS extension that is used like a table.

Using Procedures

An application uses the SQL CALL statement to invoke a procedure.

Get Procedure Metadata

The application can get information on procedures using the following functions:

- **SQLGetInfo**
  Returns information on how CA IDMS supports procedures:
  - SQL_ACCESSIBLE_PROCEDURES - The result is "Y" for for CA IDMS r15.0 or later.
  - SQL_BATCH_SUPPORT - The result includes the BS_SELECT_PROC bit when connected to an r17, or later IDMS CV, to indicate that procedures can include statements that return result sets. Note that the result also includes the BS_ROW_COUNT_PROC bit when connected to an r15, or later IDMS CV, to indicate that procedures can include statements that return row counts.
  - SQL_BATCH_ROW_COUNT - The result must be 0 to indicate that row counts are not returned to the calling application for procedure invocations.

- **SQLProcedures**
  Gets a list of the procedures defined in the SQL catalog.
  The PROCEDURE_TYPE column in the result set is always SQL_PTPROCEDURE, which indicates that the procedure does not return a value.
  The result set includes 3 additional columns, which are partially defined by ODBC, but "reserved for future use":

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_INPUT_PRAMS</td>
<td>4</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>NUM_OUTPUT_PRAMS</td>
<td>5</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>NUM_RESULT_SETS</td>
<td>6</td>
<td>SMALLINT</td>
</tr>
</tbody>
</table>
The NUM_RESULT_SETS column indicates the maximum number of result sets that can be returned from a procedure.

- **SQLProcedureColumns**
  Gets the parameter descriptions for one or more procedures from the SQL catalog. The COLUMN_TYPE for all parameters is SQL_PARAM_INPUT_OUTPUT.

### Prepare the Statement

The application uses the SQLPrepare function to specify the SQL CALL statement to the ODBC driver. The driver converts the standard ODBC escape sequence to CA IDMS format.

### Get Parameter Descriptions

The application can use the SQLNumParams and SQLDescribeParam or SQL GetDescriptor functions to determine the type and size of each input parameter for a prepared CALL statement.

### Bind Parameters

The application uses the SQLBindParameter or SQLSetDescriptor functions to specify the type and buffer for each parameter. Even though all parameters are actually SQL_PARAM_INPUT_OUTPUT, the driver allows an application to bind the parameters as SQL_PARAM_INPUT or SQL_PARAM_OUTPUT. If a parameter is bound as SQL_PARAM_INPUT or SQL_PARAM_INPUT_OUTPUT, the buffer must contain the input value or NULL when the statement is executed. If a parameter is bound as SQL_PARAM_OUTPUT the driver assumes a NULL value.

### Execute the Statement

The application uses SQLExecute to execute the CALL statement. The driver opens a cursor for the internal result set used to return output parameters and issues a FETCH to return the first row. Note that this is the only row in the result set for procedures. The driver returns output parameter values into the buffers specified by SQLBindParameter. If the procedure returned one or more result sets, the driver opens a received cursor on the first one.

### Get the Results

- **SQLNumResultCols**
  This function can be used to determine if a procedure returned one or more result sets. After the statement that calls a procedure is executed the function returns the number of columns in the first result set (not the number of output parameters). If there are no returned result sets it returns 0. After SQLGetMoreResults is called it returns the number of columns in the current result set.

- **SQLMoreResults**
  Closes the current returned result set and opens the next result set returned by the procedure, if any.
Using Named Parameters

ODBC applications can use the parameter names defined in the CA IDMS CREATE PROCEDURE statement when binding parameter values. This technique simplifies parameter binding for procedures with large numbers of parameters, particularly if default values can be used.

The named parameter feature for ODBC is used differently than it is used for JDBC because of ODBC use of structures known as descriptors. An application must first bind a data buffer to a parameter marker (‘?’) in the SQL string using an ordinal index. This process creates an implementation parameter descriptor (IPD). The parameter name can then be set in the IPD to provide the correlation between the IPD and the parameter in the CA IDMS procedure. Because of this correlation, named parameters do not have to be bound in any particular order, and a parameter can be skipped completely if an acceptable default value has been defined in the procedure.

An application uses the following functions for named parameters:

- **SQLPrepare**
  Prepares the SQL CALL statement with one or more parameter markers.

- **SQLBindParameter**
  Binds a data buffer and data type information associated with a parameter marker.

- **SQLGetStmtAttr**
  Obtains the handle for the IPD associated with a bound parameter.

- **SQLSetDescField**
  Sets the SQL_DESC_NAME field in an IPD with the appropriate parameter name from the CREATE PROCEDURE statement.

- **SQLExecute**
  Calls the procedure and returns output values.

Note that attempting mix unnamed ordinal parameters with named parameters causes an error. If automatic IPD population is used by the application, the IPDs is already in the order of the parameters specified in the CREATE PROCEDURE statement, and named parameters are ignored. Automatic IPD population is enabled by calling SQLSetConnectAttr to set SQL_ATTR_ENABLE_AUTO_IPD to SQL_TRUE. This feature is commonly used by ad-hoc query tools for determining parameter metadata.

For more information, see the Microsoft's ODBC Programmer's Section.
Using Table Procedures

An application normally uses a table procedure as if it were a table or view, by executing SQL SELECT, INSERT, UPDATE, and DELETE statements. The effect of these statements depends on how the table procedure is implemented. If an application uses a CALL statement to invoke a table procedure, only the first row of the result set is returned. You can alter this behavior if required for compatibility with an existing application.

Catalog Data

Both types of procedures are represented as rows in the SYSTEM.TABLE catalog table. The default SYSCA.ACCESSIBLE_PROCS view returns only procedures defined with the CREATE PROCEDURE syntax (table type = 'R'). You can customize the view to return table procedures by modifying the filter clause to include type 'P' as well as type 'R'.

Describe Input

Contents

- Using Describe Input (see page 35)
  - Prepare the Statement (see page 35)
  - Get Input Parameter Descriptions (see page 36)
  - Bind Input Parameters (see page 36)
  - Execute the Statement (see page 36)
  - Get Output Column Descriptions (see page 36)

The ODBC driver supports DESCRIBE INPUT for prepared statements when connected to a CA IDMS r14.0 or later system. The driver supports delayed parameter binding, which allows input parameter types to be changed when a prepared statement is re-executed without requiring that the statement be prepared again.

Using Describe Input

This feature is exposed to the application using the ODBC SQLDescribeParam function. The following is a summary of how a user written application program uses this feature with CA IDMS Server and describes how the ODBC driver processes the related ODBC APIs.

Prepare the Statement

An application uses SQLPrepare to specify the SQL statement to the ODBC driver. The driver caches the syntax and scans it to determine the type of command, counts parameter markers, and translates any ODBC escape sequences. To enhance performance, it does not PREPARE the statement on the server immediately.
Get Input Parameter Descriptions

The application can use the SQLNumParams function to determine the number of input parameters. This function does not require the driver to PREPARE the statement on the server.

The application can use the SQLDescribeParam function to determine the type and size of each input parameter. If the server is r12.0, the driver returns the "default parameter" description, usually VARCHAR. In this case, the driver does not need to PREPARE the statement on the server. If the server is r14.0 or later, the driver PREPAREs the statement on the server with the DESCRIBE INPUT option and returns the parameter description. This allows the application to get an accurate description of the parameter. However, it may result in an additional converse with the server for a query statement because the OPEN cannot be piggybacked onto the PREPARE until the statement is actually executed.

Bind Input Parameters

The application uses SQLBindParameter to specify the type of each parameter and the buffer that contains its value when the statement is executed. An application would usually specify the same type attributes that were returned by SQLDescribeParam. If a parameter's attributes are changed in a way that is incompatible with the previous definition and the server is r12.0, the driver must PREPARE the statement again before executing it. This is transparent to the application, but may cause an additional converse with the server. The driver does not need to PREPARE the statement again if the server is r14.0 or later.

If the application calls the ODBC SQLDescribeParam function subsequent to SQLBindParameter, then the results returned reflect the column definitions as they exist in the CA IDMS catalog, rather than that of a prior SQLBindParameter call.

Execute the Statement

The application uses SQLExecute to cause the driver to EXECUTE the statement or OPEN the cursor for a query. If the statement has not been PREPAREd yet, or a parameter has been changed for r12.0, the driver must PREPARE the statement first. If the statement is a query, the driver may piggyback an OPEN on this PREPARE converse.

For r14.0 or later, the application can change the contents of bound parameter buffers and re-execute the prepared statement repeatedly without requiring the driver to PREPARE the statement again.

CA IDMS discards all prepared statements when the transaction is committed. When auto commit is enabled, statements must be prepared each time they are executed. The driver does this transparently by caching the SQL syntax. An application can avoid this by disabling auto commit or setting the commit behavior to SQL_CB_PRESERVE, as described for the ODBC Positioned Updates feature.

Get Output Column Descriptions

The application uses the SQLNumResultCols function to determine the number of result set columns. It uses the SQLDescribeCol function to determine the type and size of each column in the result set.

An application usually calls these functions after executing the statement with SQLExecute or SQLExecuteDirect.
If the application calls them after SQLPrepare but before SQLExecute, the driver PREPAREs the statement on the server. If the server is r12.0, the driver uses the default parameter type for any input parameters; otherwise, the driver uses the DESCRIBE INPUT option. This may result in an additional converse with the server.

Positioned Updates

Contents

- Using Positioned Updates (see page 37)
  - Disable AutoCommit if Necessary (see page 38)
  - Set Cursor Concurrency (see page 38)
  - Specify a Cursor Name (see page 38)
  - Execute the Query (see page 39)
  - Execute the Update (see page 39)
- Developing a Custom Conversion DLL (see page 39)
  - API Reference (see page 39)
  - DBCSAloc (see page 39)
  - DBCSInit (see page 40)
  - SetDBCSOption (see page 41)
  - GetDBCSLength (see page 41)
  - DBCStoPC (see page 42)
  - DBCStoMF (see page 43)
  - DBCSEnd (see page 44)
  - How CA IDMS Server Uses the API (see page 45)

The ODBC driver supports positioned UPDATE and DELETE commands when connected to a CA IDMS r14.0 or later system. This supports a more efficient native implementation than the one implemented in the ODBC Cursor Library supplied by Microsoft, which simulates positioned updates.

When using positioned updates, the BULK FETCH and piggybacked OPEN, CLOSE, and COMMIT optimizations are not used because they would interfere with cursor currency.

Using Positioned Updates

The following is a summary of how a user written application program uses this feature with CA IDMS Server. This differs from the description in the ODBC documentation, which actually describes how to use the ODBC cursor library to simulate positioned updates. To ensure that the ODBC driver manager invokes the CA IDMS Server ODBC driver directly, the application program should not enable the ODBC Cursor Library when using this feature. Note that it is disabled by default.
Disable AutoCommit if Necessary

The default behavior for ODBC is to execute a COMMIT after each statement. The default behavior for CA IDMS is to close any open cursors when a transaction is committed, which would allow only a single positioned update for a cursor. To avoid this, the application must disable the automatic commit or change the CA IDMS commit behavior.

- **SQLGetInfo**
  The application uses this function to get SQL_CURSOR_COMMIT_BEHAVIOR. If the value is not SQL_CB_PRESERVE, the application should disable AutoCommit.

- **SQLSetConnectAttr**
  The application uses this function to disable the SQL_AUTOCOMMIT option.

If the application does not disable the AutoCommit option at runtime, use the CA IDMS ODBC Administrator to set the COMMIT BEHAVIOR option to PRESERVE CURSORS. This causes the ODBC driver to issue a COMMIT CONTINUE when committing a transaction, which preserves cursor currency.

Set Cursor Concurrency

The application allocates a statement handle for the query statement and must set cursor concurrency to allow updates. The default cursor concurrency is SQL_CONCUR_READ_ONLY, which causes the CA IDMS ODBC driver to optimize the calls to the database for retrieval, using BULK FETCH and piggybacked CLOSE options.

- **SQLSetStmtAttr**
  The application uses this function to set the SQL_ATTR_CONCURRENCY attribute to SQL_CONCUR_LOCK. This causes the driver to generate calls to the CV that allow the cursor to support positioned updates. If the statement has already been prepared, it is prepared again to use the specified concurrency. If it has already been executed, an error is returned.

Specify a Cursor Name

The application program must specify a cursor name or get a name generated by the driver.

- **SQLSetCursorName**
  The application uses this to specify an explicit cursor name. As an extension to the ODBC specification, this function sets the cursor concurrency to SQL_CONCUR_LOCK if needed. If the statement has already been prepared, it is prepared again to use the specified cursor name. If it has already been executed, an error is returned.

- **SQLGetCursorName**
  If the application has specified an explicit cursor name, this returns the application specified name. If the application has not specified a cursor name, this returns a name generated in accordance with the ODBC specification if cursor concurrency has been set to SQL_CONCUR_LOCK, and returns an error HY015 otherwise. This is according to the ODBC 2.x specification; an ODBC 3.x driver always returns a cursor name. This function can be called after the statement has been executed.
Execute the Query

The application executes an updatable query expression, using the SQLPrepare and SQLExecute or SQLExecuteDirect functions. An updatable query has the form:

```
SELECT query-expression FOR UPDATE [OF column-list]
```

CA IDMS requires the FOR UPDATE clause for a positioned update; it is optional for a positioned delete. The application program must include this clause. The ODBC driver does not scan the syntax for it and does not generate an error if the clause is present when the cursor concurrency is set to SQL_CONCUR_READ_ONLY. See the CA IDMS Database SQL Option Reference Section for detailed documentation on updateable query expressions.

The application program uses the SQLFetch function to position the cursor on the desired row. The CA IDMS Server ODBC driver disables the BULK FETCH and piggybacked CLOSE optimizations to ensure that the cursor is positioned on the row that is returned to the application.

Execute the Update

The application program allocates a second statement handle that it uses to execute the positioned UPDATE or DELETE statement, which specifies the cursor name, as shown in the following:

```
UPDATE table-name SET value-list WHERE CURRENT OF cursor-name
```

```
DELETE FROM table-name WHERE CURRENT OF cursor-name
```

CA IDMS returns an error if no cursor name has been implicitly or explicitly specified.

Developing a Custom Conversion DLL

A custom conversion DLL replaces the DLL used by CA IDMS Server to handle DBCS. This DLL is dynamically loaded when it is first used, and called for each character field sent to or received from the CA IDMS system. This includes SQL syntax, input parameters, output data, and some internal control blocks.

A custom conversion DLL can be written in any language that supports the Microsoft Windows DLL calling conventions. It must be thread safe.

API Reference

The following section describes the API that the conversion DLL must implement, and how CA IDMS Server uses each function in the API.

A custom conversion DLL must implement each function described here. The function prototypes and constants are defined in cadbcs.h, installed in the CA IDMS Server directory. This header file includes additional functions used by other CA products. Because CA IDMS Server does not use them, they are not documented here.

DBCSAlloc

- Syntax
UINT DBCSAlloc(HANDLE * hDBCS)

**Description**
Allocates the environment needed to do character conversion. This is the first call made to the conversion DLL, which must return a handle to the environment. CA IDMS Server uses this handle for all subsequent calls.

**Arguments**

hDBCS
Buffer for environment handle.

**Returns**

- **DBCS_SUCCESS** -- Function completed successfully
- **DBCS_NO_MEMORY** -- Unable to allocate memory
- **DBCS_INVALID_HANDLE** -- hDBCS is Null

### DBCSInit

**Syntax**

UINT DBCSInit(HANDLE hDBCS, UNIT fType, LPSTR lpPath)

**Description**
Initializes the conversion environment. For real DBCS processing, this specifies particular DBCS conversion tables. The custom conversion DLL can perform any initialization not completed in DBCSAlloc, or it can just return.

**Arguments**

- **hDBCS**
  Environment handle
- **fType**
  Conversion type, 1 for a custom DLL
- **lpPath**
  Path to translation tables

**Returns**

- **DBCS_SUCCESS** -- Completed successfully
- **DBCS_NO_MEMORY** -- Unable to allocate memory
- **DBCS_INVALID_HANDLE** -- hDBCS is Null
- **DBCS_TRANS_NOT_SUPPORTED**
- **DBCS_FILE_NOT_FOUND**
SetDBCSOption

**Syntax**

```c
UINT SetDBCSOption(HANDLE hDBCS, BYTE nOption, BOOL bFlag)
```

**Description**
Sets conversion options.

**Arguments**

- **hDBCS**
  Environment handle.
- **nOption**
  Option type:
  - `DBCS_KATAKANA`
  - `DBCS_NULL_TERMINATED`
  - `DBCS_PAD_SPACES`
- **bFlag**
  True to enable, False to disable

**Returns**

- `DBCS_SUCCESS` -- Completed successfully
- `DBCS_NO_MEMORY` -- Unable to allocate memory
- `DBCS_INVALID_HANDLE` -- hDBCS is Null

GetDBCSLength

**Syntax**

```c
UINT GetDBCSLength(HANDLE hDBCS, LPSTR sBuffer, LPSTR nBufferLen, UINT fType, UNIT * nLength)
```

**Description**
Computes the converted data length.

**Arguments**

- **hDBCS**
  Environment handle.
- **sBuffer**
  Input buffer
DBCStoPC

**Syntax**

```c
UINT DBCStoPC(HANDLE hDBCS, LPSTR sInBuffer, UINT nInBufferLen, LPSTR sOutBuffer, UINT nOutBufferLen, UINT fType, UINT *nLength)
```

**Description**

Converts the input buffer from EBCDIC to ASCII. The caller must allocate the output buffer and provide an output field for the converted length. Because CA IDMS Server always sets the DBCS_NULL_TERMINATED option to False, the DLL should not null terminate the converted data.

**Arguments**

- **hDBCS**
  Environment handle
- **sBuffer**
  Input buffer
- **nBufferLen**
  Input buffer length
- **nInBufferLen**
  Input buffer length
- **sOutBuffer**
  Output buffer
- **nOutBufferLen**
  Output buffer length
- **fType**
  Input data format:
  - DBCS_MF (EBCDIC)
  - DBCS_PC (ASCII)
- **nLength**
  Buffer for converted length

**Returns**

- DBCS_SUCCESS -- Completed successfully
- DBCS_NO_MEMORY -- Unable to allocate memory
- DBCS_INVALID_HANDLE -- hDBCS is Null
- DBCS_ERR_PARM -- Invalid parameter passed
- **nOutBufferLen**
  Output buffer length

- **fType**
  SQL data type:
  - DBCS_CHAR (includes VARCHAR)
  - DBCS_GRAPHIC (includes VARGRAPHIC)

- **nLength**
  Buffer for converted length

- **Returns**
  - DBCS_SUCCESS -- Completed successfully
  - DBCS_NO_MEMORY -- Unable to allocate memory
  - DBCS_INVALID_HANDLE -- hDBCS is Null
  - DBCS_ERR_PARM -- Invalid parameter passed
  - DBCS_TRUNCATION -- Converted data was truncated

### DBCStoMF

- **Syntax**
  ```c
  UINT DBCStoMF(HANDLE hDBCS, LPSTR sInBuffer, UINT nInBufferLen, LPSTR sOutBuffer, UINT nOutBufferLen, UINT fType, UINT * nLength)
  ```

- **Description**
  Converts the input buffer from ASCII to EBCDIC. The caller must allocate the output buffer and provide an output field for the converted length. The DBCS_PAD_SPACES option indicates whether the data is fixed or variable length. When True, the DLL should pad the converted data with spaces (in EBCDIC).

- **Arguments**
  - **hDBCS**
    Environment handle.
  - **sBuffer**
    Input buffer
  - **nBufferLen**
    Input buffer length
  - **nInBufferLen**
    Input buffer length
sOutBuffer
Output buffer

nOutBufferLen
Output buffer length

fType
SQL data type:
- DBCS_CHAR (includes VARCHAR)
- DBCS_GRAPHIC (includes VARGRAPHIC)

nLength
Buffer for converted length

Returns
- DBCS_SUCCESS -- Completed successfully
- DBCS_NO_MEMORY -- Unable to allocate memory
- DBCS_INVALID_HANDLE -- hDBCS is Null
- DBCS_ERR_PARM -- Invalid parameter passed
- DBCS_TRUNCATION -- Converted data was truncated

DBCSEnd

Syntax
UINT DBCSEnd(HANDLE hDBCS)

Description
Terminates the DBCS environment. CA IDMS Server calls this function before unloading the DLL, which should free all resources for the DBCS environment specified by the handle.

Arguments
- hDBCS
  Environment handle.

Returns
- DBCS_SUCCESS -- Completed successfully
- DBCS_INVALID_HANDLE -- hDBCS is Null
- DBCS_FREE_ERROR -- Unable to free memory
How CA IDMS Server Uses the API

CA IDMS Server calls the custom conversion DLL functions as follows:

<table>
<thead>
<tr>
<th>DLL Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBCSAlloc</td>
<td>Called before any other processing is done.</td>
</tr>
<tr>
<td>DBCSInit</td>
<td>Called after DBCSAlloc and before any other processing. CA IDMS Server passes the DBCS type, arbitrarily set to 1, and path specified on the CA IDMS International tab as parameters. These can be ignored.</td>
</tr>
<tr>
<td>SetDBCSOption</td>
<td>Called before DBCStoMF and DBCStoPC with the DBCS_KATAKANA option. False when Katakana is not enabled, and can be ignored. Called before DBCStoPC with the DBCS_NULL_TERMINATE option. Always False, as the ODBC driver sets the null terminator on all character data. Called before DBCStoPC with the DBCS_PAD_SPACES option. This option is 1 (TRUE) when the SQL data type is CHAR, 0 (FALSE) when it is VARCHAR.</td>
</tr>
<tr>
<td>GetDBCStLength</td>
<td>Called before DBCStoMF when the ASCII string is SQL syntax.</td>
</tr>
<tr>
<td>DBCStoPC</td>
<td>Called for each field converted from EBCDIC to ASCII.</td>
</tr>
<tr>
<td>DBCStoMF</td>
<td>Called for each field converted from ASCII to EBCDIC.</td>
</tr>
<tr>
<td>DBCSEnend</td>
<td>Called before unloading the DLL.</td>
</tr>
</tbody>
</table>

JDBC Programmer Reference

The JDBC interface allows Java applications to access different databases without specifically targeting any particular database. A set of classes called a JDBC driver is used to link an application to a specific database. The JDBC interface was developed by Sun Microsystems based on ODBC 2.5, and like ODBC, is consistent with the X/Open Call Level Interface (CLI).

This section provides information useful to developers of Java applications intended to access CA IDMS databases. A general familiarity with Java and JDBC is assumed.

The javadoc generated from the JDBC driver source code contains additional information on the CA IDMS implementation of JDBC. This HTML format documentation is installed in the CA IDMS Server directory and can be accessed from the CA IDMS Server menu.

JDBC Conformance
CA IDMS Server conforms to the JDBC 4.0 specification, which is included in Java 1.6 or later. Unless otherwise noted, all descriptions of JDBC in this document refer to JDBC 1.6.

SQL Conformance

To be JDBC compliant, a JDBC driver must support ANSI SQL-92 Entry Level. This is consistent with ODBC 3.0. With a few minor exceptions, CA IDMS conforms to the ANSI SQL-92 entry level standard. Both the ODBC and JDBC drivers pass most SQL statements to the CV essentially unchanged, other than converting escape sequences into CA IDMS equivalents.

Note: For more information on SQL conformance, see the section "ODBC Programmer Reference."

Database Type Mapping between JDBC and CA IDMS

The following tables describe how JDBC data types map to CA IDMS database data types. Java applications can use the DatabaseMetaData.getTypeInfo method to return detailed information on the mapping of JDBC and CA IDMS data types.

CA IDMS to JDBC Data Type Mapping

The following chart shows how CA IDMS types map to JDBC data types when data is returned in a result set:

<table>
<thead>
<tr>
<th>CA IDMS Data Type</th>
<th>JDBC Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>LONGINT</td>
<td>BIGINT</td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>FLOAT</td>
<td>REAL (Precision &lt; 25)</td>
</tr>
<tr>
<td>FLOAT</td>
<td>FLOAT (Precision &gt; 24)</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>DOUBLE</td>
</tr>
</tbody>
</table>
## JDBC to CA IDMS Data Type Mapping

The following chart shows how JDBC data types map to CA IDMS types when a parameter value is set.

<table>
<thead>
<tr>
<th>JDBC Data Type</th>
<th>CA IDMS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>TINYINT</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTEGER</td>
</tr>
<tr>
<td>BIGINT</td>
<td>LONGINT</td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
</tr>
<tr>
<td>FLOAT</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE PRECISION</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>DECIMAL</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>NUMERIC</td>
</tr>
<tr>
<td>CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>LONGVARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>BINARY</td>
<td>BINARY</td>
</tr>
<tr>
<td>VARBINARY</td>
<td>BINARY</td>
</tr>
<tr>
<td>LONGVARBINARY</td>
<td>BINARY</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
</tr>
<tr>
<td>TIME</td>
<td>TIME</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>TIMESTAMP</td>
</tr>
</tbody>
</table>
DriverManager

Contents

- IDMS URL Format (see page 48)
- DriverPropertyInfo (see page 49)

This section describes the information needed to connect to a CA IDMS database using the JDBC DriverManager, including the URL formats and DriverProperties recognized by the JDBC driver.

IDMS URL Format

A URL is used to locate a resource on the Internet. A URL always begins with a protocol followed by a colon, such as http: or ftp:, and the rest of the string is defined by the protocol. In keeping with the Internet orientation of Java and JDBC, URLs are used to identify databases. The JDBC specification defines conventions for the format of JDBC URLs. Each JDBC driver defines the actual format of the URLs that it recognizes. The general format of a JDBC URL is:

```
protocol: subprotocol: subname
```

- `protocol` is always `jdbc`. `subprotocol` and `subname` are defined by the JDBC driver.

The CA IDMS Server JDBC driver recognizes three URLs with subprotocol `idms`. The location of the native SQL client interface and the data source or dictionary name are specified by the `subname`.

- `jdbc:idms:database`
  Specifies the format used when the JDBC driver runs on the same machine as the native SQL client interface, that is, as a Type 2 driver. The JDBC driver calls the native interface directly.

- `jdbc:idms://hostname:port/database`
  Specifies the format used when the JDBC driver runs on a different machine than the native SQL client interface, that is, as a Type 3 or Type 4 driver. The Type 3 JDBC driver communicates with the JDBC server, which calls the native interface directly. `hostname` is the DNS name or IP address of the web server machine on which the JDBC server is running, and `port` is the IP port that was specified as the JDBC server listener. CA IDMS r16 SP2 or later supports direct connections from the Type 4 JDBC driver to the CV. `hostname` is the DNS name or IP address of the machine where the CV is running, and `port` is the IP port that was specified for the listener PTERM.

  Specifies the format used when the JDBC driver runs on a different machine than the native SQL client. In this case, the Secure Sockets Layer protocol is used for all communications between the driver and the CV (the driver acting as type 4), or between the driver and a JDBC Server (the driver acting as a type 3).

Important! The database can be an ODBC data source name or the dictionary name of the catalog containing the table definitions. When database is an ODBC data source name, the actual dictionary and physical connection information are resolved by the native SQL client interface, and must be defined on the system where the native code runs. When database is a dictionary name, the physical connection information is specified by DriverPropertyInfo objects.
When using JDBC driver types 3 and 4 with an IPv6 destination, it is still possible to code the hostname parameter in the URL with either a DNS name or an IP address. The DNS name is specified in the same way as it would be for an IPv4 destination. The IP address for an IPv6 destination must be enclosed in square brackets as shown in the following example.

```
jdbc:idms://[fec0::a:9:67:115:66]:3730/appldict
```

**DriverPropertyInfo**

JDBC DriverPropertyInfo objects are analogous to the connection attributes used by the ODBC SQLDriverConnect and SQLBrowseConnect functions. For the JDBC driver, they are used to specify user ID, password, and optional accounting information. They can also be used to specify physical connection information, allowing an application to connect to a CA IDMS database without requiring the definition of an ODBC style data source. CA IDMS Server supports the following driver properties:

- **account**
  Specifies accounting information. An optional feature that may be used by the CA IDMS system. A user exit must be installed on the DC system to process the information. See the section "Passing Accounting Information to CA IDMS" for more information.

- **ccihost**
  Specifies the DNS name or IP address of the CAICCI host server, for use by the native SQL client interface. Ignored unless node is specified. Typically, the default is used.

- **cciport**
  Overrides the default IP port of the CAICCI host server. Ignored unless ccihost is specified. Typically, the default is used.

- **csuspend**
  Specifies that when set to true, the JDBC driver suspends the SQL session and pseudo converse immediately after connecting to the database. The default for a non-pooled connection is false.

- **defschem**
  Specifies the name of the default SQL Schema. This is an optional 1-to-18 character field. When specified, this field is used as the schema qualifier for all SQL table references that do not contain an explicit schema qualifier. The default is blank (unspecified).

- **ewait**
  Sets the external wait interval for the task. This property effectively becomes the socket timeout and overrides the value specified for the task definition. It is used only by the Type 4 JDBC driver.

- **node**
  Specifies the DC NODE name, which identifies the CV containing the database. Using this property allows a connection to be established without defining an ODBC style data source. Use of this property implies that the subname contains a DICTNAME, and the driver does not search the registry or configuration file.

- **password**
  Specifies the password associated with the user ID. Required to connect to a secured CV.
• **program**
  Sets the program name for stand-alone applications when used with the DriverManager.
  `getConnection(String url, Properties info)`.

• **rsint**
  Sets the resource interval for the task. This property overrides the value specified for the task
definition. It is used only by the Type 4 JDBC driver.

• **sbuflen**
  Sets the default data buffer size for IDMSJSRV. This property overrides the value specified in the
  listener PTERM PARM string. It is used only by the Type 4 JDBC driver.

• **ssl**
  When set to true, specifies that the JDBC driver obtains a secure socket for all communication to
  an IDMS CV or a JDBC proxy server.

• **strace**
  Sets the IDMSJSRV trace flags as defined in CSACFLG1 and CSACFLG2 as directed by CA IDMS
  Technical Support.

• **task**
  Overrides the default DC TASK code that invokes the internal CA IDMS Server interfaces.

• **user**
  Specifies that a user ID is required to sign onto CA IDMS.

• **via**
  Specifies the NODE name of an intermediate CV that is used to route requests to the target
  system. Used when a physical connection cannot be established directly to the CV containing the
  SQL database. Ignored unless node is specified.

### DataSource Connection Parameters

**Contents**

- idmsDataSource (see page 51)
- IdmsConnectionPoolDataSource (see page 53)
- JDBC Connection Options Summary (see page 54)
- WebSphere Application Server DataStoreHelper (see page 55)
- Setting the External Identity (see page 55)
  - IdmsConnection.setIdentity(String identity) (see page 55)

This section describes the information used to connect to a CA IDMS database using a JDBC
DataSource, including the CA IDMS Server implementation classes and their properties.
IdmsDataSource

The IdmsDataSource class implements the JDBC DataSource interface. It is used with an application server provided Java Naming and Directory Interface (JNDI) naming service to establish a connection to a CA IDMS database.

IdmsDataSource properties conform to the Java Beans naming conventions and are implicitly defined by public "setter" and "getter" methods. For example, the "description" property, which is required for all DataSource implementations, is set using the setDescription(String) method. The application server may use the java.lang.reflect methods to discover DataSource properties and provide an interface to set them, or may simply require that they are defined in some configuration file.

IdmsDataSource properties are used to specify the connection parameters. These properties are the equivalent of the DriverPropertyInfo attributes described in the previous section and can be used to completely define the parameters needed to connect to a database. Like a URL, an IdmsDataSource object can also reference an "ODBC" style data source name, where the connection parameters are defined in the registry on Windows, the configuration file on z/OS or Linux, or in the Java properties file.

- **accountInfo**
  Specifies optional accounting information. See the DriverPropertyInfo attribute descriptions.

- **connectSuspend**
  Specifies that when set to true, the JDBC driver suspends the SQL session and pseudo converses immediately after connecting to the database. The default for a non-pooled connection is false.

- **databaseName**
  Identifies the database on the CA IDMS CV. When nodeName is specified, the driver interprets this property as the DICTNAME of the catalog in which the tables are defined and assumes that the IdmsDataSource contains all information needed to connect to the CV. When nodeName is not specified, the driver interprets this property as the name of an ODBC style data source containing connection information. The networkProtocol property can be used to override this behavior.

- **dataSourceName**
  Specifies a logical data source name. Container provided DataSource implementations use this to name the ConnectionPoolDataSource object. The IdmsDataSource implementation does not use this internally.

- **defaultSchema**
  Specifies the name of the default SQL Schema. This is an optional 1-to-18 character field. When specified, this field is used as the schema qualifier for all SQL table references that do not contain an explicit schema qualifier. The default is blank (unspecified).

- **description**
  Specifies a data source description. This property is required of all DataSource implementations.

- **externalWait**
  Overrides the external wait interval for the server task invoked by the Type 4 driver. This effectively becomes the socket timeout.
- identityAudited
   Enables end-to-end auditing of the external user identity provided by an identity manager such as CA SiteMinder.

- networkProtocol
   Specifies how the JDBC driver communicates with the CV:
   - CCI can be specified on Windows or z/OS when the native client interface is installed. It causes the driver to function as a Type 2 driver, using CAICCI/PC on Windows or CAICCI/ENF on z/OS to communicate with the CV.
   - TCP can be specified on any platform. It causes the driver to function as a Type 3 driver when connected to the JDBC Server or as a Type 4 driver when connected directly to CA IDMS.
   - IDMS can be specified on any platform. It provides a hint to the driver that it connects directly to CA IDMS as a Type 4 driver. When IDMS is specified, the driver ignores the nodeName and viaName properties and always interprets the databaseName as the DICTNAME.

- nodeName
   Specifies the NODE name that identifies the CV containing the database. This property should be used to define a Type 2 connection without defining an ODBC style data source. Use of this property implies that the databaseName is a DICTNAME.

- password
   Specifies the password for the default user ID. The application can override this when the connection is established. For security, the getPassword method does not return the value.

- portNumber
   Specifies the TCP/IP port number of the CCITCP address space, JDBC Server, or CA IDMS CV, depending on the value of networkProtocol.

- programName
   Sets an external application name to be used as the CA IDMS program name for all connections created by this data source. The first eight characters of this name are written to the journal.

- resourceInterval
   Overrides the resource interval for the server task invoked by the Type 4 driver.

- roleName
   Supported for compatibility with other DataSource implementations. The IdmsDataSource implementation does not use this internally.

- serverLength
   Overrides the default data buffer size specified in the listener PTERM PARM string for the Type 4 driver server interface, IDMSJSRV.

- serverName
   Specifies the DNS name or TCP/IP address of the server. The driver uses the value of networkProtocol to interpret this property:
   - When networkProtocol is CCI, this property refers to the mainframe where the CCITCP address space is running when the driver is running on Windows and is ignored when the JDBC driver is running on z/OS.
• When network protocol is TCP, this property refers to the machine where the JDBC server is running, which can be on any platform.

• When network protocol is IDMS, this property refers to the machine where the CA IDMS CV is running.

• serverTrace
  Sets the IDMSJSRV trace flags as defined in CSACFLG1 and CSACFLG2 as directed by CA IDMS Technical Support. Used only with the Type 4 driver.

• ssl
  Specifies that when set to true, the JDBC driver requests a secure socket for all communication to a CA IDMS CV or a JDBC proxy server.

• taskCode
  Overrides the TASK code. See the DriverPropertyInfo attribute descriptions.

• user
  Specifies a default user ID to sign on to CA IDMS. The application can override this when the connection is established.

• viaNodeName
  Specifies the NODE name of an intermediate CV used to route the connection to the destination CV for a Type 2 driver connection. Valid only if nodeName is specified. See the DriverPropertyInfo attribute descriptions.

### IdmsConnectionPoolDataSource

The IdmsConnectionPoolDataSource class implements the JDBC ConnectionPoolDataSource interface. It is used with an application server that provides container managed connection pooling to establish a pooled connection to a CA IDMS database.

An application server typically provides visible DataSource implementation that references a ConnectionPoolDataSource and is exposed to the application as a standard DataSource. Connection pooling is completely transparent to the application.

IdmsConnectionPoolProperties are used by the application server connection pool manager to administer the pool of connections for a particular data source.

• connectSuspend
  Specifies the default for a pooled connection is true.

• initialPoolSize
  Specifies the number of connections that the pool manager should initially allocate.

• maxIdleTime
  Specifies the interval in seconds that a pooled connection can be idle before it is closed. When set, it increases the resourceInterval value for a Type 4 connection to the value specified plus the value of propertyCycle property. This aligns the task resource interval with the application server idle time so that the CA IDMS system does not terminate an idle pooled connection due to inactivity.
- **maxPoolSize**
  Specifies the maximum number of connections that the pool manager should allocate.

- **maxStatements**
  Specifies the maximum statement pool size. Note that CA IDMS/DB provides an internal statement caching feature in releases 16.0 and later.

- **minPoolSize**
  Specifies the minimum number of available connections that the pool manager should maintain. This property (or the corresponding property within an application server’s connection pool definition) should always be set to 0. This avoids potential timeout-related problems between the JDBC connections and their associated tasks and sessions on the CA IDMS/DC system.

- **propertyCycle**
  Indicates the interval in seconds the pool manager should wait before enforcing these policies.

**Note:** For more information on deploying and using DataSource and ConnectionPoolDataSource objects to connect to a database, see JDBC 4.0 Specification, available at www.java.sun.com. Detailed information for these interfaces is included in the JDK "javadoc," available in the same place, and detailed information on the CA IDMS Server implementations methods is included in the installed "javadoc" (and also on the CD in /doc /javadoc.zip).

### JDBC Connection Options Summary

As described in the previous sections, there are numerous ways to define CA IDMS Server connection information. Commonly used options are:

- **When using the JDBC driver with a Servlet, EJB, or other application running in an application server**, all information can be defined in an IdmsDataSource, or more likely, an IdmsConnectionPoolDataSource or IdmsXADatasource object, and accessed using the container’s JNDI implementation. It is not necessary to define an ODBC style data source.

- **When using the Type 2 JDBC driver in a standalone Java application on Windows or z/OS**, a JNDI implementation is generally not available, and it is usually most convenient to reference an ODBC style data source in a CA IDMS URL, accessed using the DriverManager. The ODBC style data source is defined in the registry on Windows and the configuration file on the mainframe.

- **When using the Type 3 JDBC driver in an applet or standalone application that connects through the JDBC server**, it is usually most convenient to reference an ODBC style data source defined on the machine where the JDBC server invokes the native client interface.
WebSphere Application Server DataStoreHelper

Clients using CA IDMS Server as a JDBC provider within WebSphere Application Server version 7.0 and later can use the supplied com.ca.idms.was.IdmsJdbcDataStoreHelper class instead of the IBM-supplied com.ibm.websphere.rsadapter.GenericDataStoreHelper. This class is contained within the idmsjdbc.jar file and provides CA IDMS specific data source settings.

Setting the External Identity

Standalone JDBC applications can sectionly set an external user identity to be recorded in the journal similar to the audit trail provided for applications managed by CA SiteMinder.

IdmsConnection.setIdentity(String identity)

This method is a CA IDMS extension to JDBC that can be invoked to set the external identity at any time after the connection is established. This specified identity is recorded in the journal at the start of the next transaction and remains in effect until changed or set to null. This method cannot be used with pooled connections.

Distributed Transactions

Contents
- Using Distributed Transactions with JDBC (see page 55)
- Using SQL Transaction and Session Commands (see page 56)
  - COMMIT and ROLLBACK (see page 56)
  - SET SESSION (see page 56)
  - SET TRANSACTION (see page 57)

The CA IDMS Server JDBC driver supports distributed transactions when connected to CA IDMS r16 SP2 or later.

When an application enlists in a global transaction, CA IDMS creates a new internal SQL session in addition to the SQL sessions created for the local transaction and to execute DatabaseMetaData methods.

Using Distributed Transactions with JDBC

The JDBC driver supports the Java Transaction API (JTA), which is a mapping of the Open Group XA Specification and works with J2EE Compliant Transaction Managers. The JDBC driver implements the XADataSource, XAConnection, and XAResource interfaces defined by the JDBC 4.0 Specification.
A Java application uses these interfaces to create, enlist in, and commit or rollback a distributed (or global) transaction. Alternatively, a Java application can be deployed in a J2EE application server using declarative syntax that defines transaction attributes used by application server to manage the distributed transaction. In either case, the JTA compliant Transaction Manager invokes methods provided by the JDBC driver. Application servers provide tools to define Data Source objects for use with distributed transactions.


Messages returned by CA IDMS, such as those that are returned in the SQLCA, are returned in the exception that is thrown when an error occurs.

Using SQL Transaction and Session Commands

JDBC provides an explicit API to control sessions and transactions. Using the equivalent SQL statements directly is not recommended and has implementation defined results.

**COMMIT and ROLLBACK**

COMMIT and ROLLBACK are ANSI standard statements used to control transaction boundaries:

- COMMIT [CONTINUE|RELEASE]
- ROLLBACK [RELEASE]
- RELEASE

If an application executes these statements while the connection is associated with the local transaction, the driver sends them to CA IDMS and attempts to determine the state of the transaction and SQL session. Applications should use the equivalent JDBC Connection commit and rollback methods instead of executing these commands.

If an application executes these statements while the connection is associated with a global transaction, the JDBC driver returns an error. The application or transaction manager must use the commit and rollback methods defined by the Java Transaction Architecture (JTA) to complete the transaction.

**SET SESSION**

The SET SESSION statement is a CA IDMS SQL extension used to set options and default transaction attributes for the SQL session. JDBC applications can set the following SQL session options:

- SET SESSION CHECK SYNTAX SQL89|FIPS|EXTENDED
- SET SESSION CURRENT SCHEMA schema-name|NULL
- SET SESSION SQL CACHING ON|OFF|DEFAULT
CA IDMS treats these options as user session options when they are executed in an XAConnection and applies them to the SQL session for the local transaction as well as the SQL sessions for any global transactions. It also propagates the options to any subordinate sessions used by procedures and table procedures.

JDBC applications should not set the following options directly:

- SET SESSION CURSOR STABILITY|TRANSIENT READ
- SET SESSION READ ONLY|READ WRITE

JDBC applications should use the Connection setReadOnly and setTransactionIsolation methods to set transaction options instead so that the driver can maintain transaction attributes internally.

According to the JDBC specification, the behavior of these methods is implementation defined when the connection is associated with a global transaction. In this case, the CA IDMS JDBC driver applies the transaction options only to the current transaction branch.

SET TRANSACTION

The SET TRANSACTION statement is a CA IDMS SQL extension used to set the current transaction attributes:

- SET TRANSACTION CURSOR STABILITY|TRANSIENT READ
- SET TRANSACTION READ ONLY|READ WRITE

The transaction attributes are reset to the default, which can be specified by the SET SESSION statement. JDBC applications should use the Connection setReadOnly and setTransactionIsolations methods instead of executing these commands.

Batch Updates

Contents

- Using Batch Updates (see page 57)
  - Statement.executeBatch(String sql) (see page 58)
  - PreparedStatement.executeBatch() (see page 58)
  - CallableStatement.executeBatch() (see page 58)
  - BatchUpdateException (see page 58)

The JDBC driver supports batched update commands. Applications can specify a number of SQL DML or DDL commands for execution in a single request. It provides compatibility required by J2EE and supports the CA IDMS bulk insert feature for improved performance.

Using Batch Updates

An application uses the following methods to perform batch updates:
**addBatch**
Adds an SQL statement or set of parameter values to the batch.

**clearBatch**
Deletes SQL statements or parameters from the batch.

**executeBatch**
Executes the SQL statements in the batch.

Complete documentation for JDBC is available from Sun, IBM, and other sources. The JDBC 4.0 Specification contains an example of how an application would use this feature. The following sections describe CA IDMS specific considerations for this feature.

**Statement.executeBatch(String sql)**
The CA IDMS client/server interface generally supports execution of a single SQL statement per communication request (although certain transaction and session commands can be piggybacked on the main request for performance). Because CA IDMS does not currently support batch input natively, the JDBC driver caches batched statements and executes them individually.

**PreparedStatement.executeBatch()**
The JDBC driver uses the CA IDMS bulk input feature to execute INSERT statements. This allows an INSERT statement to be executed with multiple sets of parameter values in a single request. The number of sets of parameter values is limited by the maximum fetch buffer size. Because CA IDMS does not currently support bulk input for UPDATE and DELETE commands, the JDBC driver caches the parameter values and executes these commands individually for each set of values.

**CallableStatement.executeBatch()**
According to the JDBC specification, using OUT or INOUT parameters with procedures should cause an exception to be thrown. Because CA IDMS treats all procedure parameters as INOUT, an exception is thrown only if the application has specified a parameter as OUT or INOUT using the registerOutParameter method.

**BatchUpdateException**
When one or more errors occur processing a batch update command, the JDBC driver throws a single BatchUpdateException. An SQLException is generated for each error that occurs during the processing of the batch and chained to the BatchUpdateException in the order that the statements were added to the batch. Each SQLException identifies the statement with a message in the form: "Batch element #: original message text"

**Procedures 1**

**Contents**
- Using Named Parameters (see page 59)
- Result Sets (see page 60)
The JDBC driver supports specifying procedure parameters by name instead of ordinal. When connected to CA IDMS r17 or later it supports procedures that return result sets.

**Using Named Parameters**

A CallableStatement object, used for calling SQL procedures, can support binding of parameters using the parameter names defined in CA IDMS by the CREATE PROCEDURE statement. This technique is an alternative to identifying each parameter by an ordinal index corresponding to a parameter marker ("?") in the SQL CALL statement. Named parameters are useful for procedures which have large numbers of parameters, particularly if default values can be used. Parameters can be bound in any order; a parameter can also be skipped completely if an acceptable default value has been defined in the procedure.

To use named parameters, an SQL CALL statement is prepared which contains markers for necessary parameters, for example:

```java
CallableStatement cstmt = conn.prepareCall("{ CALL MYPROC(? , ? , ?) }");
```

Each parameter used as IN or INOUT is then bound to a value using the setXXX method that is appropriate for the parameter's data type. Each parameter used as OUT must be registered using the registerOutputParameter method and specifying the expected data type. The parameter names for both types of methods must have been defined in the CREATE PROCEDURE statement in CA IDMS.

```java
cstmt.setString("INPARM", "First");
cstmt.setString("IOPARM", "Second");
cstmt.registerOutParameter("OUTPARM", java.sql.types.STRING);
```

After the statement has been executed, INOUT and OUT parameters can be retrieved using the parameter names:

```java
String io = cstmt.getString("IOPARM");
String out = cstmt.getString("OUTPARM");
```

Named parameter binding cannot be mixed with ordinal binding on the same CallableStatement object; an SQLException is thrown if this is attempted.
For more information on named parameters, see the Java documentation for the Java Platform API Specification or the JDBC 4.0 Specification.

Result Sets

CA IDMS r17, or later, supports procedures that return result sets.

When connected to a CA IDMS r17 system, or later, the JDBC driver supports multiple open result sets, and the DatabaseMetaData supportsMultipleResultSets and supportMultipleOpenResults methods both return true.

The result set returned by the DatabaseMetaData.getProcedures method contains 3 additional columns. JDBC indicates that these are "reserved for future use", consistent with the equivalent result set defined by ODBC.

<table>
<thead>
<tr>
<th>Name</th>
<th>Number</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUM_INPUT_PARMS</td>
<td>4</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>NUM_OUTPUT_PARMS</td>
<td>5</td>
<td>SMALLINT</td>
</tr>
<tr>
<td>NUM_RESULT_SETS</td>
<td>6</td>
<td>SMALLINT</td>
</tr>
</tbody>
</table>

The NUM_RESULT_SETS column indicates the maximum number of result sets that can be returned from a procedure. The value is NULL for CA IDMS r16 or earlier.

The Statement.getMoreResultSet(int) method supports all three values of the argument:

- KEEP_CURRENT_RESULT
- CLOSE_CURRENT_RESULT
Scrollable Result Sets

Contents
- JDBC Result Sets and Row Sets (see page 61)
- CA IDMS Result Sets (see page 63)

JDBC Result Sets and Row Sets

The JDBC java.sql.ResultSet interface defines an object used to manipulate an SQL cursor. It provides methods to position the cursor, access columns within the current row, and update values in the table.

The type attribute indicates how the current row is positioned for the result set, whether it is scrollable or not, and the visibility of changes made by other transactions or cursors. There are three types:

- TYPE_FORWARD_ONLY -- the cursor can only move forward.

- TYPE_SCROLL_INSENSITIVE -- the cursor can move forward, backward, or to a specific row. The values in the result set are fixed when the cursor is opened or the rows are first retrieved, depending on the database implementation, and do not generally reflect changes made by other transactions.

- TYPE_SCROLL_SENSITIVE -- the cursor can move forward, backward, or to a specific row. The values in result set generally do reflect changes made by other transactions.

The concurrency attribute indicates if the result set is updateable. An updateable result set provides methods that can be used to change values in the table, and is an alternative to using SQL positioned update statements. There are two concurrency options:
CONCUR_READ_ONLY -- the current row cannot be updated directly.

CONCUR_UPDATABLE -- the current row can be updated using JDBC methods instead of SQL statements.

These attributes are independent, which means there are six possible combinations.

A JDBC driver provides an implementation of the ResultSet interface. At a minimum a driver must support a forward only, read only result set. Any additional capabilities are optional.

The JDBC javax.sql.ResultSet interface extends the ResultSet interface with methods that support the JavaBeans component model. The javax.sql.rowset package includes a set of specialized row set interfaces that provide additional capabilities. These include the javax.sql.rowset.JdbcRowSet, wraps a JDBC ResultSet and maintains a connection to the database, and the javax.sql.rowset.CachedRowSet, which caches column values and can be disconnected from the database.

Row sets are designed to be implemented on top of the JDBC methods, and JDBC drivers are not required to implement them. A row set implementation can support type and concurrency options beyond those supported by the JDBC drivers result set implementation. Starting with Java 1.5, the Java Run Time Environment (JRE) includes a Reference Implementation (RI) of the javax.sql.rowset package.

The JDBC API documentation (javadoc) contains detailed descriptions of the classes and methods that support this feature. The JDBC Specification also contains examples of how an application would use a scrollable or updateable result set.
CA IDMS Result Sets

The CA IDMS Server r17 JDBC driver supports TYPE_FORWARD_ONLY and TYPE_SCROLL_INSENSITIVE result sets. It supports the concurrency option CONCUR_READ_ONLY.

Although the CA IDMS JDBC driver does not directly support TYPE_SCROLL_SENSITIVE and CONCUR_UPDATABLE result sets, these options are available when used with an appropriate javax.sql.RowSet implementation.

- **TYPE_SCROLL_INSENSITIVE**
  The driver implements TYPE_SCROLL_INSENSITIVE result sets with a memory based client-side cache. Values are added to the result set as the rows are fetched, and are not are not changed to reflect changes by other transactions or other statements within the same transaction when the application positions the cursor on a cached row.
  The Reference Implementation of the javax.sql.JdbcRowSet interface, com.sun.rowset.JdbcRowSetImpl, is included in the Java 1.6 run time library. When used with an IdmsResultSet object it supports a row set that is TYPE_SCROLL_INSENSITIVE and CONCUR_READ_ONLY.

- **TYPE_SCROLL_SENSITIVE and CONCUR_UPDATABLE**
  The Reference Implementation of the javax.sql.CachedRowSet interface, com.sun.rowset.CachedRowSetImpl, is included in the Java 1.6 run time library. When used with an IdmsResultSet object it can support a row set that is TYPE_SCROLL_SENSITIVE and CONCUR_UPDATABLE (or any combination of type and concurrency).
  There are some restrictions on the use of this feature.
  The query statement used for a CONCUR_UPDATABLE row set must satisfy the CA IDMS criteria for an updateable cursor:
  - Only one table can be specified in the FROM clause.
  - The query cannot contain derived or aggregate columns.
  - The query cannot contain UNION, ORDER BY, or GROUP BY.
  - The query must select all NOT NULL columns that have no default.
  - The query statement used for a TYPE_SCROLL_SENSITIVE result set must satisfy a subset of these criteria:
    - The query cannot contain derived or aggregate columns.
    - The query cannot contain UNION, ORDER BY, or GROUP BY.

Note that the driver does not detect that the query cannot be used for the requested type or concurrency. If the RowSet implementation detects this, it demotes the result set to a supported type or concurrency.

The query statement should not include a FOR UPDATE clause. This does not prevent deadlocks and can result in a less efficient access strategy when the clause does not specify specific columns. A CA IDMS cursor can move forward only. An application can fetch multiple rows in a single request to improve performance. When an application uses this BULK FETCH feature, CA IDMS considers the last row in the buffer to be current of cursor, and only this row has a read lock. CA IDMS has no way to specify that the current row should have an update lock. The FOR UPDATE clause only affects the access path generated by the optimizer, not the locking strategy.

The application cannot use a scrollable or updateable row set for positioned updates.
Since the driver uses a memory-based cache, applications should not use a TYPE_SCROLL_INSENSITIVE result set or row set for large result sets. TYPE_SCROLL_SENSITIVE row sets can be used for fairly large result sets but may perform more slowly in applications that access a high percentage of the fetched rows.

When the fetch direction hint for a statement or result set is set to FETCH_REVERSE, the driver considers the result set holdability to be CLOSE_CURSORS_AT_COMMIT, no matter what has been set for other statements. The driver attempts to use COMMIT instead of COMMIT CONTINUE in order to reduce resource use and contention in CA IDMS.

See the CA IDMS javadoc, installed in the HTNL Bookshelf for detailed information on the CA IDMS JDBC method implementations.

Positioned Updates 1

The JDBC driver supports positioned updates and deletes in dynamic SQL, when connected to a CA IDMS r14.0 or later system. For prior releases, the ResultSet setCursorName and getCursorName methods are implemented only to conform to the JDBC specification, and are not used internally.

To use positioned updates and deletes, you must specify the FOR UPDATE clause in the SQL query statement as follows:

```
SELECT ... FROM ... WHERE ... FOR UPDATE [OF column-name...]
```

If only a subset of the columns in the result set needs to be updated, it is advisable to use the "OF column-name..." clause. Otherwise, CA IDMS/DB uses an area sweep to read the table, even when the table is indexed.

To optimize performance, the JDBC driver usually attempts to fetch more than one row at a time. Because row currency is at the last row, issuing a positioned update or delete would not have the expected effect. Specifying the FOR UPDATE clause or setting a cursor name using setCursorName (String) directs the driver to fetch one row at a time.

In general (to improve performance), we recommend that you turn auto-commit OFF when using Positioned Updates. CA IDMS discards all prepared statements when the transaction is committed. When auto-commit is enabled, you need to prepare statements each time they are executed. An application can avoid this overhead by either:

- Disabling auto-commit, or
- Setting the cursor behavior to SQL_CB_PRESERVE (ODBC).

HibernateDialect

Hibernate is an open-source software product available from Red Hat that provides an Object-Relational Mapping (ORM) and persistence solution for Java developers. With Hibernate, developers are able to work with data as Java objects rather than as the rows and columns of a relational database.
database. Developers generally do not have to code JDBC calls or SQL statements and do not have to be concerned with the syntax and behavior of a particular DBMS. A special Java class known as a dialect, unique to a particular DBMS, specifies the data types, functions and features supported by the DBMS.

CA IDMS Server provides a dialect, IDMSDialect.class, which is included in the idmsjdbc.jar file. To use it, add the jar file to the classpath definition and set the following properties in the hibernate.cfg.xml file:

<table>
<thead>
<tr>
<th>Hibernate Property</th>
<th>Setting for IDMS Dialect</th>
</tr>
</thead>
<tbody>
<tr>
<td>dialect</td>
<td>com.ca.idms.hibernate.IDMSDialect</td>
</tr>
<tr>
<td>connection.driver_class</td>
<td>ca.idms.jdbc.IdmsJdbcDriver</td>
</tr>
</tbody>
</table>

Sample Programs

Contents

- IdmsJcf (see page 65)
- IdmsExample (see page 66)
- IdmsJdbcDataStoreHelper (see page 66)

Two simple SQL query utilities are included as sample programs distributed with CA IDMS Server. Neither requires installation. You can copy the class files to the client machine along with the JDBC driver.

The sample programs are installed in the src, lib, and classes sub-directories of the installation directory on Windows and z/OS. They are also provided in the /java/samples.tar file within the CA IDMS Server installation.

IdmsJcf

This can be thought of as a simple Java version of OCF, providing a Graphical User Interface (GUI) query facility. It can be run as an application or an applet on any machine supporting the Swing classes. Both source code and compiled class files are installed, as well as a sample HTML page to invoke it as an applet. On the Windows platform, a shortcut is added to the CA IDMS Server menu to run it as an application.

To run to the CA IDMS JCF applet demo in a web browser, the JDBC server must be running on the web server. Because JdbcTest is the default data source, consider defining a data source called JdbcTest.

This sample is installed in the CA IDMS Server installation directory:

- /idmsdir/src/ca/idms/jcf/IdmsJcf.java
  Source code, entry point and UI

- /idmsdir/src/ca/idms/jcf/JdbcTable.java
  Source code, JDBC calls
- /idmsdir/src/idmsjcf.html
  Sample web page to invoke as applet

- /idmsdir/lib/idmsjcf.jar
  Compiled IdmsJcf classes

⚠️ For UNIX and Linux systems, where there is no automated installation process, these files can be found on the CA IDMS Server CD in these locations:

- \java\samples.tar
- \Server\Windows32\program files\CA\CA IDMS Server\Java\idmsjcf.html

**IdmsExample**

This can be thought of as a simple Java version of BCF. It reads a series of SQL commands from a text file and writes the results to the standard output. Since it has no GUI, it can be run from any command line interface, including a 3270 terminal on z/OS. Both source code and a compiled class file are installed, along with a shell script to invoke it, and a sample SQL input file. The script and sample input file contain documentation on the command line options.

This sample is installed in the CA IDMS Server installation directory:

- /idmsdir/src/example/IdmsExample.java
  Source code

- /idmsdir/example.sql
  Sample SQL input file

- /idmsdir/bin/example
  Shell script to run IdmsExample.class

- /idmsdir/classes/example/IdmsExample.class
  Compiled sample program

**IdmsJdbcDataStoreHelper**

The helper class for WebSphere Application Server is also supplied as a sample, installed in the CA IDMS Server installation directory:

- /idmsdir/src/com/ca/idms/was/IdmsJdbcDataStoreHelper.java
Sample SSL Scripts

Several sample scripts have been provided to assist you in testing the SSL feature when using a type 3 JDBC connection through the CA-IDMS Java Server running on Unix Systems Services (USS). These scripts are samples only and may need to be tailored to your specific installation. The scripts, their descriptions, and locations are listed following:

**USS** (within directory "/idmsdir/sampssl"):

- GenServerKey - Generates the Server Key
- ListAllSSLCerts - Lists all Certificates in the Keystore
- ExportServerSSLCert - Exports the Server Key
- SSLStart - Starts the Java Server
- SSLStatus - Checks the Java Server status
- SSLStop - Stops the Java Server

**Windows** (within directory "/idmsdir/sampssl"):

- GenClientKey.bat - Creates the Client keystore
- ImportSSLCert.bat - Imports the Server Certificate
- ListSSLCert.bat - Lists the Server Certificates
- Jcf_SSL_Testing.bat - Starts the JCF demo app.

The following procedure can be used to create and populate your keystores, and to start both the Java Server and the JCF Demo facility using the appropriate parameters. The JCF Demo application is used to test the SSL feature.

**Within OMVS:**

1. Ensure that both the HOME and JAVA_HOME environment variables have been properly set for your environment.
2. Copy the contents of the "sampssl" sub-directory into the CA IDMS Server main directory. All script invocations should occur from the CA IDMS Server main directory.
3. Run the GenServerKey script.
4. Run the ListAllSSLCerts script.
5. Run the ExportServerSSLCert script.
6. Edit the caidms.cfg file and set SSL=1 within the Proxy section.
7. Run the SSLStart script.

8. Run the SSLStatus script.

9. FTP the file created in Step 1e (named "idsslsrv.cer") to the "sampssl" sub-directory on Windows. This file must be transferred in binary mode.

10. Within Windows:

11. Update your PATH environment variable to include the JAVA/BIN directory for your JRE or JSDK installation.

12. Open a Command Prompt window and issue a Change Directory (CD) command to go to the "sampssl" sub-directory for your CA IDMS Server installation.

13. Run the GenClientKey.bat script

14. Run the ImportSSLCert.bat script

15. Run the ListSSLCert.bat script

16. Run the Jcf_SSL_Testing.bat script

17. Establish a JDBC Type 3 SSL connection to your IDMS data source, making sure to specify "ssl" within the connection URL. For example:

```
jdbc:idms:ssl://host-name:port/data-source-name
```

Once you are done with your testing, stop the Java Server running under USS. To do so, run the SSLStop script within OMVS.

## Windows Registry Information

The registry is a database used by Windows to store system and application information.

### Registry Information

**Contents**

- HKEY_LOCAL_MACHINE Software ODBC ODBCINST.INI (see page 69)
- HKEY_LOCAL_MACHINE Software ODBC ODBC.INI (see page 70)
  - Data Source Advanced Options (see page 71)
    - Values (see page 71)
- HKEY_LOCAL_MACHINE Software CA CA IDMS Server (see page 75)
  - Servers (see page 76)
  - Server server_name (see page 76)
    - Values (see page 76)
  - Server Advanced Options (see page 77)
This section describes the information stored in the registry and used by CA IDMS Server. This information is provided to help you identify problems that may arise with CA IDMS Server. The registry information is maintained using the ODBC Administrator, available from the Control Panel. Unlike ini files, it cannot be edited directly, but it can be edited using the registry editor provided by Microsoft. Only advanced users should attempt to edit the registry directly, since an error can disable not only CA IDMS Server, but also Windows itself.

The registry is structured as a hierarchical database, with keys, sub-keys, and values. Two of the top level keys are used by the ODBC Driver Manager and the CA IDMS Server drivers. HKEY_LOCAL_MACHINE contains information on hardware and software common to all users of the machine. HKEY_CURRENT_USER contains preferences and application settings for the current user. A sub-key is analogous to a directory path and is specified in a similar fashion. The following are the sub-keys used by ODBC and CA IDMS:

- HKEY_LOCAL_MACHINE\Software\ODBC\ODBCINST.INI
- HKEY_LOCAL_MACHINE\Software\ODBC\ODBC.INI
- HKEY_LOCAL_MACHINE\Software\CA\CA IDMS Server
- HKEY_CURRENT_USER\Software\ODBC\ODBC.INI
- HKEY_CURRENT_USER\Software\CA\CA IDMS Server

Under each of these keys are sub-keys corresponding to the section names used in ini files. At the lowest level are value names, corresponding to the key names used in ini files, and the values themselves. The remainder of this section describes the information in these sub-keys.

### HKEY_LOCAL_MACHINE Software ODBC ODBCINST.INI

This section contains information on the ODBC drivers installed on the machine. The CA IDMS Server installer program adds the information for the ODBC driver using the Microsoft ODBC installer DLL when the product is installed. The following is a summary of these values. Refer to the Microsoft ODBC reference for more detailed information

<table>
<thead>
<tr>
<th>Subkey</th>
<th>Value Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ODBC Core</td>
<td>UsageCount</td>
<td>Driver manager usage count</td>
</tr>
<tr>
<td>ODBC Drivers</td>
<td>CA IDMS</td>
<td>Each installed ODBC driver has an entry. Value name is the driver name. Value data is installed.</td>
</tr>
<tr>
<td>CA IDMS</td>
<td></td>
<td>Each installed driver has a sub-key, whose name is the name of the driver.</td>
</tr>
</tbody>
</table>
Subkey | Value Name | Description
--- | --- | ---
APILevel | Driver ODBC API conformance level.
ConnectFunc | Connect functions supported by driver.
Driver | Driver DLL name and path.
DriverODBCV | Version of ODBC supported by driver.
FileExtns | Not used for CA IDMS Server.
FileUsage | Not used for CA IDMS Server.
Setup | Driver setup DLL name and path.
SQLLevel | Driver SQL conformance level.
UsageCount | Driver usage count
Default | Driver | Name of ODBC driver for the default data source.

HKEY_LOCAL_MACHINE Software ODBC ODBC.INI

This section contains information on system data sources, which are available to all users of the system, as well as system services. The ODBC.INI key contains the following sub-keys and values:

Subkey | Value Name | Description
--- | --- | ---
ODBC Data Sources | DSN | Each data source has an entry. Value Name is the data source name. Value Data is the driver name. For CA IDMS Server, this is CA IDMS.
DSN | Each data source has a sub-key whose name is the data source name.
Driver | Driver DLL name and path, copied from the ODBCINST.INI key.
Dictionary | DBNAME or segment name of the CA IDMS dictionary defined in the DBNAME table on the target CV. Value comes from the CA IDMS Server ODBC Administrator dialog Dictionary field. The default is the first eight characters of data source name.
Server | Server name that specifies how to connect to the CA IDMS system.
Default Schema | Optional default schema name.
Options | Advanced options, defined below.
Default | Default data source can contain the same values as other data source definitions.
Data Source Advanced Options

The following advanced options reside under the Data Source Name (DSN) sub-key. Note that integer values be stored as a registry type of REG_SZ when set under a data source name sub-key in the registry. Commonly used options are set in the CA IDMS Server ODBC Administrator Data Source tab Advanced Options dialog. Rarely used options can be set by editing the registry.

<table>
<thead>
<tr>
<th>Value Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccessibleTables</td>
<td>0</td>
</tr>
<tr>
<td>AccountPrompt</td>
<td>0</td>
</tr>
<tr>
<td>CacheSQLTables</td>
<td>0</td>
</tr>
<tr>
<td>CatalogTable</td>
<td>view_name</td>
</tr>
<tr>
<td>CallSelect</td>
<td>0</td>
</tr>
<tr>
<td>CloseCommit</td>
<td>0</td>
</tr>
<tr>
<td>CommitBehavior</td>
<td>0</td>
</tr>
<tr>
<td>ConnectSuspend</td>
<td>0</td>
</tr>
<tr>
<td>DefaultParmType</td>
<td>integer_value</td>
</tr>
<tr>
<td>DefaultSchema</td>
<td>schema_name</td>
</tr>
<tr>
<td>DescribeExtended</td>
<td>0</td>
</tr>
<tr>
<td>EnableEnsure</td>
<td>0</td>
</tr>
<tr>
<td>FetchDouble</td>
<td>0</td>
</tr>
<tr>
<td>FetchRows</td>
<td>integer_value</td>
</tr>
<tr>
<td>FetchSize</td>
<td>integer_value</td>
</tr>
<tr>
<td>FetchSuspend</td>
<td>0</td>
</tr>
<tr>
<td>FetchSuspendClose</td>
<td>0</td>
</tr>
<tr>
<td>IgnoreDTC</td>
<td>0</td>
</tr>
<tr>
<td>InvalidDecimal</td>
<td>integer_value</td>
</tr>
<tr>
<td>LoginTimeout</td>
<td>integer_value</td>
</tr>
<tr>
<td>PoolSuspendActive</td>
<td>0</td>
</tr>
<tr>
<td>PreservePrepared</td>
<td>0</td>
</tr>
<tr>
<td>QueryTimeout</td>
<td>integer_value</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>0</td>
</tr>
<tr>
<td>SuspendStrategy</td>
<td>0</td>
</tr>
<tr>
<td>TxnIsolation</td>
<td>1</td>
</tr>
<tr>
<td>WaitTimeOut</td>
<td>integer_value</td>
</tr>
</tbody>
</table>

Values
AccessibleTables=0|1
When set to 1, the ODBC and JDBC drivers use the SYSCA.ACCESSIBLE_TABLES view, or another view defined by you, for the SQLTables function and getTables method. A setting of 0 disables this option. This value is set from the Use Accessible Tables View Name field.

AccountPrompt=0|1
Directs the ODBC driver to prompt for information if the ACCT keyword is not supplied in the connection string passed to SQLDriverConnect. For more information, see the section "Passing Accounting Information to CA IDMS."

CacheSQLTables=0|1
When set to 1, the ODBC driver caches the table list returned from an SQLTables call. A value of 0 disables this option. This value is set from the Cache SQL Tables option.

CallSelect=0|1
A value of 1 specifies that the ODBC and JDBC drivers should treat all SQL CALL statements as SELECT statements. This means that all parameters are returned in a result set. All procedures are essentially treated as TABLE PROCEDUREs, which can be useful to solve compatibility problems with some applications. The default is 0, which allows the use of the CallableStatement methods with an SQL CALL statement. This feature is deprecated and should not be used with new applications.

CatalogTable=view_name
Specifies the name of the view that the ODBC and JDBC drivers use for the SQLTables function and getTables method, if other than the default view name. This value is set from the Use Accessible Tables View Name field.

CloseCommit=0|1 (ODBC Driver)
When enabled, CA IDMS Server sends a COMMIT following a CLOSE when auto-commit is off. The default value is 1, enabled. This option is also considered enabled when auto-commit is on. The COMMIT (or COMMIT CONTINUE) is usually piggybacked onto the FETCH or CLOSE request when no other cursors are open and no updates are pending. This can also be specified in a specific Data Source section.
Default value: 1 (enabled)

CloseCommit=0|1 (JDBC Driver)
When enabled, CA IDMS Server sends a COMMIT following a CLOSE operation. The default value is 0, disabled. This option is only in affect when auto-commit is on. The COMMIT (or COMMIT CONTINUE) is usually piggybacked onto the FETCH or CLOSE request when no other cursors are open and no updates are pending. This option can also be specified in a specific Data Source section.
Default value: 0 (disabled)

CommitBehavior=0|1|2
Specifies the way a COMMIT operation affects cursors in CA IDMS. This also determines the value returned by the ODBC SQLGetInfo function for the SQL_CURSOR_COMMIT_BEHAVIOR option and the JDBC default ResultSetHoldability. This value is set from the Commit Behavior field. Values are:

- 0 -- Specifies SQL_CB_DELETE, which is equivalent to ResultSet.CLOSE_CURSORS_AT_COMMIT. All open cursors are closed, and all prepared statements are deleted. Specified by selecting Close and Delete Cursors in the Commit Behavior field.
- 1 -- Specifies SQL_CB_CLOSE, which is equivalent to ResultSet.CLOSE_CURSORS_AT_COMMIT. All open cursors are closed, but prepared statements are not deleted. Specified by selecting Close Cursors in the Commit Behavior field.

- 2 -- Specifies SQL_CB_PRESERVE, which is equivalent to ResultSet.HOLD_CURSORS_OVER_COMMIT. All cursors remain open, and their position is preserved. Prepared statements are not deleted. Specified by selecting Preserve Cursors in the Commit Behavior field.

**CommitSuspend=0|1**
Causes the driver to issue a SUSPEND after each COMMIT. The default is 1, enabled. SuspendStrategy should generally be used instead of this option. The SuspendStrategy must be set to 3 (CUSTOM) to use this option.

**ConnectSuspend=0|1**
Causes the JDBC driver to issue a SUSPEND and end the task immediately after it establishes a connection. ODBC does not provide a way for the driver to be notified of this event. The default for a pooled connection is 1, otherwise it is 0. SuspendStrategy should generally be used instead of this option. The SuspendStrategy must be set to 3 (CUSTOM) to use this option.

**DefaultParmType=integer_value**
Specifies an ODBC SQL data type, as defined in the ODBC sql.h header file, that is used as the default type when an ODBC function requiring the SQL statement to be prepared is executed before all input parameters are bound. Recommended values are:

- 1 -- SQL_CHAR
- 4 -- SQL_INTEGER
- 5 -- SQL_SMALLINT

This feature is deprecated, CA IDMS and the drivers support DESCRIBE INPUT.

**DefaultSchema=schema_name**
Specifies the name of the default SQL Schema. This is an optional 1-to-18 character field. When specified, this field is used as the schema qualifier for all SQL table references that do not contain an explicit schema qualifier. The default is blank (unspecified)

**DescribeExtended=0|1**
When set to 1, the driver requests extended column descriptor information from a CA IDMS r17 system, including the names of the schema and table. The default for CA IDMS r17 SP0 is 0, for r17 SP1 it is 1. This is ignored for prior releases.

**EnableEnsure=0|1**
When set to 1, the ODBC driver honors the ENSURE parameter of the SQLStatistics function call. A setting of 0 disables this option. This value is set from the Enable Ensure field.

**FetchDouble=0|1**
When set to 1, CA IDMS converts single precision floating point numbers to double precision floating point before returning them to CA IDMS Server. This value is set from the Fetch Real as Double field.
**FetchRows=integer_value**  
Specifies the number of database rows CA IDMS Server fetches at a time. The default is 0, which causes the driver to request the number of rows that fits in a fetch buffer of the size specified by the FetchSize registry. This value is set from the Fetch Row Count field.

**FetchSize=integer_value**  
Specifies the maximum size that the JDBC driver attempts to use for a FETCH buffer. The default is 64,000 when using the IDMS native TCP/IP interface. The default and maximum is 29,000 when using CAICCI/PC. This should generally be left at the default setting. This value is set from the Fetch Row Size field.

**FetchSuspend=0|1**  
When enabled, CA IDMS Server causes a SUSPEND to be piggybacked onto each BULK FETCH, ending the IDMS-DC task. The default is 0, disabled. SuspendStrategy should generally be used instead of this option. The SuspendStrategy must be set to 3 (CUSTOM) to use this option.

**FetchSuspendClose=0|1**  
In prior releases caused a conditional SUSPEND to be piggybacked onto each FETCH. The SUSPEND was done only if the cursor reached the end. This option is no longer supported; the CloseCommit and CommitSuspend options specify the equivalent behavior.

**IgnoreDTC=0|1**  
A value of 1 specifies that the CA IDMS Server ODBC driver ignores requests for distributed transactions with Microsoft’s Distributed Transaction Coordinator (DTC). This option should normally be set to 0.

**InvalidDecimal=0|1|2|3**  
Specifies how the ODBC and JDBC drivers handle invalid packed or zoned decimal data returned in a result set column. This value is set from the invalid Decimal Action field. Options are:

- 0 -- Return error, the default
- 1 -- Return NULL
- 2 -- Return 0
- 3 -- Ignore, ODBC only

**LoginTimeout=integer_value**  
Specifies the system loginTimeout used when the JDBC DriverManager or DataSource setLoginTimeout is set to 0.

**PoolSuspendActive=0|1**  
In prior releases caused a pooled connection to be treated like a non-pooled connection. This is no longer supported, the CommitSuspend option specifies the equivalent behavior.

**PreservePrepared=0|1**  
Attempt to preserve prepared statements when the ODBC cursor commit behavior is preserve or JDBC result set holdability is enabled. The default is 0, disabled, which maximizes concurrency between transactions. The CA IDMS SQL statement caching feature can be used to minimize the overhead of re-preparing statements.
- **QueryTimeout**
  Specifies the default reply timeout for SQL requests which use a Java Statement object when using the JDBC driver.

- **ReadOnly=0|1**
  Specifies the default access mode for the ODBC and JDBC drivers. A setting of 0 specifies Read Write. A setting of 1 specifies Read Only. This value is set from the Access Mode field.

- **SuspendStrategy=0|1|2|3**
  Specifies how the driver uses pseudo-conversational processing. This value is set from the Suspend Strategy field, and is equivalent to specifying the detailed suspend options, as described in Section3, "Setting Up Your CA IDMS System." Values are:
  - 0 -- INTERACTIVE, the default.
  - 1 -- SERVICE, suspend when idle.
  - 2 -- BATCH, never suspend.
  - 3 -- CUSTOM, defined by the detailed options.

- **TxnIsolation=1|2**
  Specifies the degree to which your transactions impact, and are impacted by, other users accessing the same data. A setting of 1 specifies Read Uncommitted, 2 specifies the default setting, Read Committed. This value is set from the Transaction Isolation field.

- **WaitTimeOut**
  Specifies the default system reply timeout for the JDBC driver.

**HKEY_LOCAL_MACHINE Software CA CA IDMS Server**

This section contains all global data source and server options, as well as server definitions and system services. The following subkeys can be contained within this key:

- **Servers**
  Associates a server name with an ODBC driver name.

- **Server server_name**
  Specifies how to connect to the CA IDMS system and advanced communications option settings.

- **Options**
  Specifies global options..

- **Proxy**
  Contains information used to configure the JDBC server.

- **DBCS Types**
  The DBCS Types subkey identifies the languages that have DBCS support. The values are added when CA IDMS Server is installed.

- **Version**
  Contains the current version number for the latest installed release of IDMS Server.
Servers

The Servers sub-key lists all server_names defined using the CA IDMS Server ODBC Administrator dialog. Each server_name has the value "CA IDMS", the name of the ODBC driver.

Server server_name

The Server server_name sub-key contains information describing a CA IDMS system. The server_name part of the sub-key specifies a server name listed in the Servers section and referenced by a Data Source definition.

The following values describe how to connect to a CA IDMS system:

<table>
<thead>
<tr>
<th>Value Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccessType</td>
<td>I</td>
</tr>
<tr>
<td>CciServerName</td>
<td>cci_name</td>
</tr>
<tr>
<td>CciServerPort</td>
<td>integer_value</td>
</tr>
<tr>
<td>Host</td>
<td>host_name</td>
</tr>
<tr>
<td>Port</td>
<td>integer_value</td>
</tr>
<tr>
<td>Resource</td>
<td>node_name</td>
</tr>
<tr>
<td>SSL</td>
<td>0</td>
</tr>
<tr>
<td>WaitTimeOut</td>
<td>integer_value</td>
</tr>
</tbody>
</table>

Values

- **AccessType=Access_Type**
  Specifies how the CA IDMS ODBC Driver (or Type 2 JDBC Driver) communicates with CA IDMS. Valid values are:
  - I - The drivers use the CA IDMS TCP/IP feature to communicate directly with the CA IDMS system. CA IDMS r17, or later, is required.
  - C - The drivers use the CA Common Services CCI feature to communicate with the CA IDMS system. This is the default, and is supported for all releases of CA IDMS.

- **CciServerName=cci_name**
  Identifies the DNS name or IP address where the CCITCP Server is running. If not specified, the default server defined for CAICCI is used.

- **CciServerPort=cci_port**
  (Optional) Specifies the IP port identifying the CCITCP Server on the node defined by cci_name. If not specified, the default port defined for CAICCI is used. This is usually 1202, and typically should not be specified here.

- **Host=host_name**
  Identifies the DNS name or IP address where the IDMS CV is running. This option is only used when the IDMS Communications Protocol is selected (AccessType='I').
Port=
Specifies the TCP/IP port of the CA IDMS Listener running under CV. This option is only used when the IDMS Communications Protocol is selected (AccessType='I').

Resource=node_name
(Optional) Specifies the value of SYSTEMID. This is specified in the SYSTEM statement of the system generation of the target system. If a node_name is not specified, CA IDMS Server uses the first eight characters of the server_name to identify the target system. This option is only used when the CCI option is selected (AccessType='C').

SSL=0|1
Enables Secure Socket Layer (SSL) connections between the ODBC and Type 2 JDBC drivers and CA IDMS when using the IDMS communications protocol.

WaitTimeOut=integer_value
Specifies the number of seconds CAICCI waits for a response from the CA IDMS system. When this interval is exceeded, CA IDMS Server considers the connection to have failed. Set this to 0 to cause CAICCI to use the default value specified with the CAICCI/PC Properties dialog.

Server Advanced Options

The following advanced server options reside under the 'Server server_name' sub-key.

<table>
<thead>
<tr>
<th>Value Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlternateTask</td>
<td>task_code</td>
</tr>
<tr>
<td>AsciiEbcdicTables</td>
<td>translation_table_name</td>
</tr>
<tr>
<td>BufferLength</td>
<td>integer_value</td>
</tr>
<tr>
<td>ExternalWait</td>
<td>integer_value</td>
</tr>
<tr>
<td>Node</td>
<td>via_node_name</td>
</tr>
<tr>
<td>ResourceInterval</td>
<td>integer_value</td>
</tr>
<tr>
<td>Version</td>
<td>0</td>
</tr>
</tbody>
</table>

Values

- **AlternateTask=task_code**
  Identifies an alternate task defining the resource limits and timeout values for a session. The default is CASERVER. The task must be defined as a task on the CA IDMS system generation TASK statement. This value comes from the Task Code field on the Server tab of the CA IDMS Server ODBC Administrator dialog. For more information on resource limits for external user sessions, see CA IDMS System Generation and CA IDMS System Operations.

- **AsciiEbcdicTables=translation_table_name**
  Specifies the name of the CECP translation table selected to convert EBCDIC data on the server to ASCII data on the PC, and vice versa. The value comes from the International tab of the CA IDMS Server ODBC Administrator dialog.
- **BufferLength** = *integer_value*
  Specifies the size of the buffer used by the CA IDMS Server listener for TCPIP send and receive requests. This value comes from the “Buffer Length” field on the Advanced Server Options tab of the CA IDMS Server ODBC Administrator dialog.

- **ExternalWait** = *integer_value*
  Specifies the number of seconds that the CA IDMS Server listener waits for a request from the client when a task is active. This value overrides the EXTERNAL WAIT INTERVAL specified for the TASK when enabled by specifying TIMEOUT=-1 in the CA IDMS Server listener PTERM definition. This is optional. When set to 0, the System Default for all servers value is used, if any. For more information on TASK and PTERM System Generation statements see the CA IDMS Server System Generation section.

- **Node** = *via_node_name*
  Specifies the node with which CAICCI establishes a connection. The system identified by via_node_name must contain a RESOURCE table entry for the system identified by node name. Use this option when the system containing your tables does not directly communicate with CAICCI.

- **ResourceInterval** = *integer_value*
  Specifies the number of seconds that the CA IDMS Server listener waits for a request from the client when no task is active. This value overrides the RESOURCE INTERVAL specified for the TASK when enabled by specifying TIMEOUT=-1 in the CA IDMS Server listener PTERM definition. When set to 0, the System Default for all servers value is used, if any.

- **Version** = 0|1
  Specifies the version of the CA IDMS Server mainframe component installed on the CA IDMS CV.
  - 0 -- Indicates Version 4.2 or earlier.
  - 1 -- Indicates Version 4.3 or later. This is the default setting.

  Version 4.3 or later of the CA IDMS Server mainframe component supports password encryption using a proprietary algorithm. When this option is set to 1, the password is encrypted prior to being transmitted over the network and is decrypted by the CCI line driver in the CV. This encryption process is discrete and is not affected by any other encryption applied by technologies such as SSL. CA IDMS releases 14.0 SP4, 14.1 SP4 and all subsequent releases contain a Version 4.3 or later mainframe component and therefore support a setting of 1. For older CA IDMS releases, set this option to 0 to specify Version 4.2 or earlier.

### Options

The Options sub-key contains global options, including default data source and server options, log options, and internationalization options. The default options are documented in the Data Source and Server sections.

Global Options are set with the Options, Log Options, and International tabs of the CA IDMS Server ODBC Administrator. Options that are rarely used can be set by editing the registry.

<table>
<thead>
<tr>
<th>Value Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>cadcdc32.dll</td>
<td>dll_name</td>
</tr>
<tr>
<td>Value Name</td>
<td>Value</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>DbcPath</td>
<td>dbcs_path</td>
</tr>
<tr>
<td>DbcType</td>
<td>dbcs_type</td>
</tr>
<tr>
<td>JcliTraceWs</td>
<td>0</td>
</tr>
<tr>
<td>JdbcTraceID</td>
<td>0</td>
</tr>
<tr>
<td>LogFile</td>
<td>log_file_name</td>
</tr>
<tr>
<td>LogFileCount</td>
<td>integer_value</td>
</tr>
<tr>
<td>LogFileSize</td>
<td>integer_value</td>
</tr>
<tr>
<td>LoginTimeout</td>
<td>integer_value</td>
</tr>
<tr>
<td>LogOptions</td>
<td>log_option_values</td>
</tr>
<tr>
<td>MultiThread</td>
<td>0</td>
</tr>
<tr>
<td>Path</td>
<td>path_name</td>
</tr>
<tr>
<td>QueryTimeout</td>
<td>integer_value</td>
</tr>
<tr>
<td>SSLCertDir</td>
<td>certificate_directory_name</td>
</tr>
<tr>
<td>SSLCertClientCert</td>
<td>client_certificate_file_name</td>
</tr>
<tr>
<td>SSLPassword</td>
<td>client_certificate_password (encrypted)</td>
</tr>
<tr>
<td>SSLServerCert</td>
<td>server_certificate_file_name</td>
</tr>
<tr>
<td>XxxxTrace</td>
<td></td>
</tr>
<tr>
<td>CmTrace</td>
<td>integer_value</td>
</tr>
<tr>
<td>DnsTrace</td>
<td>integer_value</td>
</tr>
<tr>
<td>DtsTrace</td>
<td>integer_value</td>
</tr>
<tr>
<td>FdeTrace</td>
<td>integer_value</td>
</tr>
<tr>
<td>JcliTrace</td>
<td>integer_value</td>
</tr>
<tr>
<td>JdbcTrace</td>
<td>integer_value</td>
</tr>
<tr>
<td>OdbcTrace</td>
<td>integer_value</td>
</tr>
<tr>
<td>SQLTrace</td>
<td>integer_value</td>
</tr>
<tr>
<td>UtilTrace</td>
<td>integer_value</td>
</tr>
<tr>
<td>WaitTimeOut</td>
<td>integer_value</td>
</tr>
</tbody>
</table>

### Values

- **cadc32.dll=dll_name**
  Specifies the name of a user-supplied customized character conversion DLL, used by the native client interface to convert between ASCII and EBCDIC. The name can be qualified with a path. DbcType must be set to a non-zero value, typically 1, to enable the use of the specified DLL.

- **DbcsPath=dbcs_path**
  Specifies the path to the DBCS translation tables, typically the direction specified when CA IDMS Server is installed.
- **DbcsType** = `dbcs_type`
  Specifies the integer value identifying the DBCS Language, as defined by the DBCS Types subkey.

- **JcliTraceWs** = 0 | 1
  Enables Windows socket trace. This option is not exposed in the IDMS ODBC Administrator application and should only be set under the direction of CA Technical Support.

- **JdbcTraceId** = 0 | 1
  Causes the JDBC driver to prefix each line written to the JDBC log writer with the current timestamp and thread name. This can be useful to identify CA IDMS output in a JDBC DataSource. logWriter trace managed by an application server. The default is 0, disabled.

- **LogFile** = `log_file_name`
  Specifies the name of the log file, if other than the default log name. This value comes from the Log File field on the Log Options tab of the CA IDMS Server ODBC Administrator dialog.

- **LogFileCount** = `integer_value`
  Specifies the maximum number of archive log and enables the log file rollover feature when the LogFileSize value is greater than zero. The default is 0, which indicates a single log file. This value comes from the File Count field on the Log Options tab of the CA IDMS Server ODBC Administrator dialog.

- **LogFileSize** = 64-bit `integer_value`
  Specifies the maximum size (in bytes) of the log file when the log file rollover feature is enabled. The default is zero, which indicates no maximum size. This value comes from the Log File field on the Log Options tab of the CA IDMS Server ODBC Administrator dialog.

- **LoginTimeout** = `integer_value`
  Specifies the system loginTimeout used when the JDBC DriverManager or DataSource setLoginTimeout is set to 0.

- **LogOptions** = `log_option_values`
  Specifies log options as a bit mask. The bit flag, 0x0001, appends information to the existing log file, if any.

- **MultiThread** = 0 | 1
  Specifies whether CA IDMS Server processes ODBC connections or multiple threads concurrently. A setting of 1 enables multithreaded access, a setting of 0 disables it. The default is 1.

- **Path** = `path_name`
  Specifies the directory where files used by CA IDMS Server are installed.

- **QueryTimeout**
  Specifies the default reply timeout for SQL requests which use a Java Statement object when using the JDBC driver.

- **SSLCertDir** = `certificate_directory_name`
  Specifies the name of the certificate directory. This directory can contain individual certificates (in PEM format), and is searched for the resolution of signing certificates.
SSLClientCert=client_certificate_file_name
Specifies the fully qualified name of the client certificate file. This file is typically generated on the mainframe and transmitted to the Windows client. Use of this field is optional. A client certificate is only needed if client authentication is required for all SSL connections. All certificate files must be in PEM format.

SSLPassword=client_certificate_password
Identifies the password used for the client certificate. This is an optional field and is only necessary when a client certificate is specified. The SSLPassword field is encrypted before it is written to the registry. Users should not attempt to modify this field outside of the CA IDMS ODBC Administrator application.

SSLServerCert=server_certificate_file_name
Specifies the fully qualified name of the client certificate file. This file is typically generated on the mainframe and transmitted to the Windows client. Use of this field is optional. A client certificate is only needed if client authentication is required for all SSL connections. All certificate files must be in PEM format.

XxxxTrace=integer_value
Specifies the flag bits used to control tracing. Technical Support uses these trace flags to resolve CA IDMS Server problems. The integer_value must be in the range of 0, which signifies all options off, to 65535, which signifies all options on. This value can be specified as a decimal or hexadecimal integer.

WaitTimeOut
Specifies the default system reply timeout for the JDBC driver.

Descriptions of the individual bit flags are as follows:

- CmTrace (IDMSTD0D.DLL):
  - 0x0001 // trace CCI and internal function calls
  - 0x0002 // elapsed CCI call timings

- DnsTrace (IDMSTD0D.DLL):
  - 0x0010 // snap unconverted send data
  - 0x0020 // snap converted send data
  - 0x0040 // snap received data
  - 0x0080 // snap converted received

- DtsTrace (IDMSTD0D.DLL):
  - 0x0002 // trace external calls
  - 0x0004 // trace events
  - 0x0008 // trace events
0x0010 // snap user data arrays
0x0020 // trace events
0x0040 // snap PCE
0x0080 // snap LCE

FdeTrace (IDMSFDE.DLL):
0x0001 // trace external generate calls (for precompiler)
0x0002 // trace external convert calls
0x0004 // trace external ASCII-EBCDIC conversion calls
0x0010 // trace internal calls
0x0100 // snap format descriptors
0x1000 // snap input (unconverted) data
0x2000 // snap output (converted) data

JcliTrace (IDMSJCLI.DLL):
0x0001 // trace internal function calls
0x0002 // time all socket sends/recvs
0x0004 // trace all socket calls
0x0008 // trace ssl calls
0x0010 // snap all send & receive buffs
0x0020 // snap conversion functions
0x0040 // Snap SSL packets

JdbcTrace (ca.idms.*) -- Any non-zero value enables tracing

OdbcTrace (IDMSODBC.DLL):
0x0002 // Trace internal functions
0x0004 // Trace function parms
0x0008 // Trace thread locks
0x0010 // Snap SQL syntax
0x0100 // Snap environment block
The ODBC Administrator is usually used to enable and disable tracing. Because tracing can add overhead and affect performance, it should be disabled under normal circumstances.

Proxy

The Proxy subkey contains information used to configure the JDBC server, and has the following values:
### Values

- **Backlog=integer_value**
  Specifies the maximum length of the listener queue. When this is exceeded, connections are refused. This is not the maximum number of client connections that can be supported. The default is 50.

- **ClientAuth=0|1**
  Enables client authentication when the JDBC driver connects to this proxy server using SSL.

- **Encoding=character_encoding_name**
  Specifies the character encoding that the JDBC server requests the JDBC driver to use when sending and receiving character data. If not specified, the default encoding for the JVM is requested. The character encoding class must be accessible to the JDBC driver when invoked by the client application or applet.

- **Host=host_name**
  Specifies the DNS name or IP address the JDBC binds to when listening for client connection requests. This can be used to force the JDBC server to listen for connection requests on a specific TCP/IP protocol stack on a multi-homed host (a machine with multiple TCP/IP stacks). The default is to listen on all available stacks.

- **LogLevel=integer_value**
  Specifies the level of messages sent to the Windows Event Log.

  - 0 -- Disable messages
- 4 -- Error messages
- 6 -- Warning messages
- 8 -- Information messages, including start and stop events. This is the default.
- 10 -- Verbose information messages, including client start and stop events.
- 12 -- Debugging messages, not including general trace output.

**LogTrace=integer_value**
Specifies the level of log messages sent to the trace file. Options are identical to LogLevel options.

**Port=integer_value**
The IP port the JDBC server listens on for connection requests. The default value is 3709.

**RemoteHost=host_name**
(Optional) Specifies the DNS name or IP address of a CA IDMS system (r16 SP2 or later), or another JDBC server used to forward packets to the CA IDMS system.

**RemotePort=integer_value**
Specifies the IP port address of the remote host. If used, the default value is 3709.

**RemoteSSL=0|1**
Enables SSL when communicating with another proxy server.

**ReplyTimeOut=integer_value**
Specifies the number of seconds the JDBC server will wait for a response from the CA IDMS system. The default, 0, causes the JDBC server to wait indefinitely.

**Snap=0|1**
Enables display of data buffers sent and received in the log file.

**SocketTimeOut=integer_value**
Specifies the number of seconds the JDBC server waits, or blocks when reading data from a socket. While a socket is being read, the thread is blocked, and is not able to recognize an event that stops the thread. When this interval expires, the thread checks if the JDBC server is still running, and, if so, issues another read on the socket. It continues until the wait or reply timeout has expired. A high value reduces JDBC server overhead. A low value allows the server to respond to shutdown events more quickly. Setting this to 0 causes the thread to block forever, and is not recommended. The default is 60 seconds.

**SSL=0|1**
Enables Secure Socket Layer (SSL) connections between this proxy server and a JDBC driver client.

**Trace=0|1**
Enables tracing of internal function calls. Output is written to the log file.

**Unicode=0|1**
Enables the use of Unicode for character encoding when the JDBC driver is unable to use the requested encoding. The default value, 0, specifies the use of UTF-8, which is supported by all Java platforms.
• **WaitTimeOut**=*integer_value*
  Specifies the number of seconds the JDBC server will wait for a request from the JDBC driver. The default, 0, causes the JDBC server to wait indefinitely.

### HKEY_CURRENT_USER Software ODBC ODBC.INI

This section contains information on user data sources available only to the currently signed-on users of the system.

Use the ODBC Administrator to maintain this information. The structure of the information under this key is the same as the ODBC.INI sub-key of HKEY_LOCAL_MACHINE.

### HKEY_CURRENT_USER Software CA CA IDMS Server

This section contains information on user servers available only to the currently signed-on users of the system. The following sub-keys can be contained within this key:

- **Servers**
  Associates a server name with an ODBC driver name.

- **Server server_name**
  Defines each server's database access path information.
  The structure of the information under these keys is the same as system servers defined as sub-keys of HKEY_LOCAL_MACHINE.

### Configuration File Information

CA IDMS Server uses a text file to store configuration information on z/OS. This file contains database definitions, server definitions, global options, and JDBC server options on z/OS. This file is similar in format to a Windows .ini file. Information used by the native interfaces must be specified in the configuration file. Information used by the Java code can be specified in the configuration file or in the properties file.

**Note:** For more information, see the section "Properties File Information."

### Configuration Information

**Contents**

- Environment Variables (see page 87)
- Sections (see page 87)
  - Datasource (see page 87)
  - Server server_name (see page 90)
Data is organized into sections, identified by square brackets (for example, `[section_name]`). Within each section, parameters are defined by key-value pairs, delimited by an equal sign (for example, `key=value`). A comment is indicated by a semicolon (`;`).

Because many 3270 devices and emulators do not support square brackets (`[]`), you can use dollar signs (`$`) or percent symbols (`%`) instead. The closing symbol is also optional.

### Environment Variables

- **IDMS_CFG_PATH=path_name**
  By default, the configuration file is named caidms.cfg and is located in the CA IDMS Server installation directory. The IDMS_CFG_PATH environment variable can be used to specify a different file or directory.

- **IDMS_CFG_RELOAD=0|1**
  For optimal performance on z/OS, the configuration file is copied into a memory file when the libidmsutil.so DLL is initially loaded into a process. When the IDMS_CFG_RELOAD environment variable is set to 1 the configuration file is reloaded from the file system each time libidmsutil.so is loaded. This overrides the CacheConfig option set in the configuration file itself. The default value is 0.

### Sections

The configuration file includes the following sections:

- **[datasource_name]**
  Defines the SQL catalog and CA IDMS system for each database. This information is used by the native libraries. Database specific options used by the JDBC driver can also be specified in this section.

- **[Server server_name]**
  Defines access information for each CA IDMS system. This information is used by the native client interface.

- **[Options]**
  Contains global options, including data source defaults and log options. This information is used by the JDBC driver and server and by the native client interface.

- **[Proxy]**
  Contains information used by the JDBC server.

### Datasource

The `datasource_name` section identifies CA IDMS databases, and can be specified in the JDBC URL or the IdmsDataSource databaseName property. A datasource_name section may contain the following key-values:
- **AccessibleTables=0|1**
  Enables the use of an alternate view for the getTables method. The default is 1, a setting of 0 disables this option. The view name is specified with the CatalogTable key.

- **CallSelect=0|1**
  A value of 1 specifies that the ODBC and JDBC drivers should treat all SQL CALL statements as SELECT statements. This means that all parameters are returned in a result set. All procedures are essentially treated as TABLE PROCEDURES, which can be useful to solve compatibility problems with some applications. The default is 0, which allows the use of the CallableStatement methods with an SQL CALL statement. This feature is deprecated and should not be used with new applications.

- **CatalogTable=** *view_name*
  Specifies the name of the view that the JDBC driver uses for the getTables method, when AccessibleTables=1. If not specified the driver uses the SYSCA.ACCESSIBLE_TABLES view.

- **CloseCommit=0|1 (ODBC Driver)**
  When enabled, CA IDMS Server sends a COMMIT following a CLOSE when auto-commit is off. The default value is 1, enabled. This option is also considered enabled when auto-commit is on. The COMMIT (or COMMIT CONTINUE) is usually piggybacked onto the FETCH or CLOSE request when no other cursors are open and no updates are pending. This can also be specified in a specific Data Source section.
  Default value: 1 (enabled)

- **CloseCommit=0|1 (JDBC Driver)**
  When enabled, CA IDMS Server sends a COMMIT following a CLOSE operation. The default value is 0, disabled. This option is only in effect when auto-commit is on. The COMMIT (or COMMIT CONTINUE) is usually piggybacked onto the FETCH or CLOSE request when no other cursors are open and no updates are pending. This option can also be specified in a specific Data Source section.
  Default value: 0 (disabled)

- **CommitBehavior=0|2**
  Specifies the default as ResultSetHoldability, which is the way a COMMIT operation affects cursors in CA IDMS. The values set in the configuration file map to the JDBC ResultSet values as follows:
  - 0 -- ResultSet.CLOSE_CURSORS_AT_COMMIT. All open cursors are closed and all prepared statements are deleted (although the SQL statement is cached in the Statement object and prepared again if needed).
  - 2 -- ResultSet.HOLD_CURSORS_OVER_COMMIT. All cursors remain open and their position is preserved. Prepared statements are not deleted

  These values are consistent with those used to the set SQL_CURSOR_COMMIT_BEHAVIOR for ODBC (on Windows the ODBC and JDBC drivers use the same option in the registry). The JDBC driver interprets a value of 1 as 0.

- **CommitSuspend=0|1**
  Causes the driver to issue a SUSPEND after each COMMIT. The default is 1, enabled. SuspendStrategy should generally be used instead of this option. Only when the SuspendStrategy is set (or defaults) to “Custom” will the CommitSuspend option be in effect.
CA IDMS - 19.0

- **ConnectSuspend=0|1**
  Causes the JDBC driver to issue a SUSPEND and end the task immediately after it establishes a connection. The default for a pooled connection is 1, otherwise it is 0. SuspendStrategy should generally be used instead of this option. Only when the SuspendStrategy is set (or defaults) to “Custom” will the ConnectSuspend option be in effect.

- **Dictionary=** dict_name
  Specifies the name of the dictionary containing the SQL schema definitions for the tables or network records to be accessed. This name is defined in the DBNAME table on the target CA IDMS system. The default value is the first eight characters of the datasource_name.

- **DefaultSchema=** schema_name
  Specifies the name of the default SQL Schema. This is an optional 1-to-18 character field. When specified, this field is used as the schema qualifier for all SQL table references that do not contain an explicit schema qualifier. The default is blank (unspecified).

- **DescribeExtended=0|1**
  When set to 1, the driver requests extended column descriptor information from a CA IDMS r17 system, including the names of the schema and table. The default for CA IDMS r17 SP0 is 0, for r17 SP1 it is 1. This is ignored for prior releases.

- **FetchRows=** integer_value
  Specifies the default value for the JDBC statement object fetchSize property. This is the number of rows that the JDBC driver requests from CA IDMS for each BULK FETCH. The value can be set and queried at run time using the JDBC setFetchSize and getFetchSize methods. When set to 0, the default, the JDBC driver attempts to fetch as many rows as will fit in a buffer of size specified by the FetchSize option.
  **Note:** The name of this option in the configuration file does not correspond exactly to the JDBC object properties in order to remain compatible with previous versions of CA IDMS Server.

- **FetchSuspend=0|1**
  When enabled, CA IDMS Server causes a SUSPEND to be piggybacked onto each BULK FETCH, ending the IDMS-DC task. The default is 0, disabled. SuspendStrategy should generally be used instead of this option. Only when the SuspendStrategy is set (or defaults) to “Custom” will the FetchSuspend option be in effect.

- **FetchSuspendClose=0|1**
  In prior releases caused a conditional SUSPEND to be piggybacked onto each FETCH. The SUSPEND was done only if the cursor reached the end. This is no longer supported; the CloseCommit and CommitSuspend options specify the equivalent behavior.

- **InvalidDecimal=0|1|2**
  Specifies how the JDBC driver is to handle invalid packed or zoned decimal data returned in a result set column. Options are as follows:
  0 -- Return error. This is the default setting
  1 -- Return NULL
  2 -- Return 0

- **LoginTimeout=** integer_value
  Specifies the system loginTimeout used when the JDBC DriverManager or DataSource setLoginTimeout is set to 0.
PoolSuspendActive=0|1
In prior releases caused a pooled connection to be treated like a non-pooled connection. This is no longer supported, the CommitSuspend option specifies the equivalent behavior.

PreservePrepared=0|1
Attempt to preserve prepared statements when JDBC result set holdability is enabled. The default is 0, disabled, which maximizes concurrency between transactions. The CA IDMS SQL statement caching feature can be used to minimize the overhead of re-preparing statements.

QueryTimeout=integer_value
Specifies the default reply timeout for SQL requests which use a Java Statement object when using the JDBC driver.

ReadOnly=0|1
Specifies the default access mode for the JDBC driver. A setting of 0 specifies Read Write. A setting of 1 specifies Read Only.

Server=server_name
(Required) Specify the CV used to access the data. This name can be a NODE Name or a user-defined Server name, referring to a Server server_name section containing additional connection information.
A datasource_name section can also contain database specific options, described in the Options section. Default values that apply to all data sources can be set in the Options section.

SuspendStrategy=0|1|2|3
Specifies how the driver uses pseudo-conversational processing. This value is set from the Suspend Strategy field, and is equivalent to specifying the detailed suspend options, as described in Section3, "Setting Up Your CA IDMS System." Values are as follows:
0 -- INTERACTIVE, the default.
1 -- SERVICE, suspend when idle.
2 -- BATCH, never suspend.
3 -- CUSTOM, defined by the detailed options.

TxnIsolation=1|2
Specifies the degree to which your transactions impact, and are impacted by, other users accessing the same data. A setting of 1 specifies Read Uncommitted, 2 specifies the default setting, Read Committed.

WaitTimeOut=integer_value
Specifies the default system reply timeout for the JDBC driver.

Server server_name

The [Server server_name] section contains information describing a CA IDMS system. The Server server_name section can contain the following parameters:

AlternateTask=task_code
Identifies an alternate task defining the resource limits and timeout values for a session. The default is CASERVER. The task named must be defined as a task on the CA IDMS system generation TASK statement. For more information on resource limits for external user sessions, see the CA IDMS System Generation and CA IDMS System Operations sections.
- **Node=via_node_name**
  Specifies an intermediate node to route the connection to the target system. The system identified by via node must contain a RESOURCE table entry for the system identified by node name. Use this option when the system containing the tables to be accessed does not directly communicate with CAICCI.

- **Resource=node_name**
  Identifies the value of SYSTEMID as specified in the system generation parameters of the target system. If a node name is not specified, CA IDMS Server uses the first eight characters of server_name to identify the target system.

- **Version=0|1**
  Specifies the version of the CA IDMS Server mainframe component installed on the CA IDMS CV.
  
  0 -- Indicates Version 4.2 or earlier.
  
  1 -- Indicates Version 4.3 or later. This is the default setting.
  
  **Note:** Version 4.3 or later of the CA IDMS Server mainframe component supports password encryption using a proprietary algorithm. When this option is set to 1, the password is encrypted prior to being transmitted over the network. This encryption process is not affected by any other encryption applied by technologies such as SSL. All currently supported releases of CA IDMS releases 14.0 SP4, 14.1 SP4 and all subsequent releases contain a Version 4.3 or later mainframe component and therefore support a setting of 1. For CA IDMS releases prior to 14.0 SP4, set this option to 0.

### Options

The [Options] section contains global options, including path information, logging options, and debugging flags. Other than the log and trace options, most can also be specified for a specific data source. The Options section can contain the following parameters:

- **CacheConfig=0|1**
  Enables or disables caching of the configuration file in memory. The default value is 1, enabled. The IDMS_CFG_RELOAD environment value can be used to override this setting when necessary to refresh the cache. This is a global option.

- **DefaultSchema=schema_name**
  Specifies the name of the default SQL Schema. This is an optional 1-to-18 character field. When specified, this field is used as the schema qualifier for all SQL table references that do not contain an explicit schema qualifier. The default is blank (unspecified).

- **FetchSize=integer_value**
  Specifies the maximum size that the JDBC driver attempts to use for a FETCH buffer. The default is 64,000. Depending on the platform and implementation of the CAICCI interface, a smaller buffer may be used. This usually is left at the default setting. Specifying too large a value may cause the Java Virtual Machine to run out of memory.
  
  **Note:** The name of this option in the configuration file does not correspond exactly to the JDBC object properties in order to remain compatible with previous versions of CA IDMS Server.

- **LogFile=log_file_path**
  Specifies the location and name of the log file. A path name should be specified as an absolute path, for example, /idmsdir/log/caidms.log. If the LogFile value ends in a ‘/’, the default file name of caidms.log is appended to the path.
- **LogFileCount**\texttt{=integer\_value}  
  Specifies the maximum number of archive log files to keep. If a value greater than zero is specified, the LogFileSize value must also be greater than zero. The default is 0, which indicates a single log file.

- **LogFilePid=0|1**  
  When set to 1, the process id is appended to the log file name to make it unique. The default is 0, disabled. In z/OS UNIX System Services, the log file cannot be shared among different processes. This option allows a common configuration file to be used by multiple processes, but still allowing each process to have a unique log file name.

- **LogFileSize**\texttt{=integer\_value}  
  Specifies the maximum size, in bytes, of the active log file. If a value greater than zero is specified, then the LogFileCount value must also be greater than zero. The default is zero, which indicates no maximum size.

- **LogOptions**\texttt{=integer\_value}  
  Specifies log options as a bitmask. The bit flags are:
  - 0x0010 -- Display 8-byte thread ID in trace (z/OS)
  - 0x0020 -- Send messages to the system log (SYSLOG). This is the default
  - 0x0040 -- Send messages to the system console (z/OS)

- **LoginTimeout**\texttt{=integer\_value}  
  Specifies the system loginTimeout used when the JDBC DriverManager or DataSource setLoginTimeout is set to 0.

- **JdbcTraceId=0|1**  
  Causes the JDBC driver to prefix each line written to the JDBC log writer with the current timestamp and thread name. This can be useful to identify CA IDMS output in a JDBC DataSource. logWriter trace managed by an application server. The default is 0, disabled.

- **QueryTimeout**\texttt{=integer\_value}  
  Specifies the default reply timeout for SQL requests which use a Java Statement object when using the JDBC driver.

- **WaitTimeOut**\texttt{=integer\_value}  
  Specifies the default system reply timeout for the JDBC driver.

- **XxxxTrace**\texttt{=integer\_value}  
  Specifies the flag bits used to control tracing. Customer Support uses these flags to diagnose CA IDMS Server problems. The integer_value is a bit mask used to specify individual trace options. A setting of 0 turns all options off, and a setting of 65535, or 0xFFFF, turns all options on. Specify this value as a decimal or hexadecimal integer. Descriptions of the bit flags are as follows:
  - CmTrace (libtd0d.so):
    - 0x0001 -- Trace CAICCI and internal function calls
    - 0x0002 -- Elapsed CAICCI call timings
- 0x0004 -- Snap control blocks
- 0x0008 -- Debug #CAICCI calls on z/OS
- 0x0010 -- Trace signon failures

- DnsTrace (libtd0d.so):
  - 0x0010 -- Snap unconverted send data
  - 0x0020 -- Snap converted send data
  - 0x0040 -- Snap received data
  - 0x0080 -- Snap converted received

- DtsTrace (libtd0d.so):
  - 0x0002 -- Trace external calls
  - 0x0004 -- Trace events
  - 0x0008 -- Trace events
  - 0x0010 -- Snap user data arrays
  - 0x0020 -- Trace events
  - 0x0040 -- Snap PCE
  - 0x0080 -- Snap LCE

- JdbcTrace (idmsjdbc.jar) -- Any non-zero value enables tracing

- SqlTrace (libcli.so):
  - 0x0002 -- Time SQL calls
  - 0x0004 -- Snap SQL SQLSID
  - 0x0008 -- Snap SQL DSICB
  - 0x0010 -- Snap SQL SQLCA
  - 0x0020 -- Snap SQL SQLCIB
  - 0x0040 -- Snap SQL SQLPIB
  - 0x0080 -- Snap SQL parm buffer
  - 0x0100 -- Snap SQL tuple buffer
  - 0x0200 -- Snap SQL input SQLDA
Proxy

The [Proxy] section contains information used to configure the JDBC server. It can contain the following parameters:

- **Backlog=integer_value**
  Specifies the maximum length of the listener queue. When this length is exceeded, new connections are refused. This is not the maximum number of client connections that can be supported. The default is 50.

- **ClientAuth=0|1**
  Requires a client certificate when JDBC driver clients connect to this proxy server using SSL.

- **Encoding=character_encoding_name**
  Specifies the character encoding that the JDBC server requests the JDBC driver to use when sending and receiving character data. If not specified, the default encoding for the JVM is requested. The character encoding class must be accessible to the JDBC driver when invoked by the client application or applet.
  In Java, all character data is represented internally as Unicode. Ultimately this data must be converted to the native platform encoding used by CA IDMS, a variant of EBCDIC specified by the code page. The Java platform includes classes to convert between Unicode and the various character encodings. The encodings supported by a particular Java implementation depend on the vendor.
  In the absence of documentation, it might be possible to determine the encodings supported by converted classes supplied with the Java implementation. These are generally named ByteToChar xxxx.class and CharToBytexxxxx.class, where xxxx is the encoding name. A minimal subset of the converter classes is installed in the base library for the Java Run Time Environment, jre/lib/rt.jar. Additional converter classes are included with the international version of the Java 2 Platform, installed in the same subdirectory, but the actual filenames vary by release. For Java 1.4 the international converter classes are in charsets.jar.

- **Host=host_name**
  Specifies the DNS name or IP address the JDBC binds to when it listens for client connection requests. This can be used to force the JDBC server to listen for connection requests on a specific TCP/IP protocol stack on a multi-homed host (a machine with multiple TCP/IP stacks). The default is to listen on all available stacks.
- **LogLevel=integer_value**
  Specifies the level of messages sent to the system log or console. Choose one of the following options:
  - 0 -- Disable messages
  - 4 -- Error messages
  - 6 -- Warning messages
  - 8 -- Information messages, including start and stop events. This is the default.
  - 10 -- Verbose information messages, including client start and stop events
  - 12 -- Debugging messages, not including general trace output.

- **LogTrace=integer_value**
  Specifies the level of log messages sent to the trace file. Options are identical to the options for LogLevel.

- **Port=port**
  Specifies the IP port that the JDBC server listens on for connection requests. The default is 3709.

- **RemoteControl=0|1**
  Enables a remote client to control the JDBC server; to SUSPEND, RESUME, or STOP it. The default value, 0, allows remote clients only to check the STATUS of the JDBC server.

- **RemoteHost=host_name**
  (Optional) Specifies the DNS name or IP address of a CA IDMS system (r16 SP2 or later), or another JDBC server used to forward packets to the CA IDMS system.

- **RemotePort=port**
  Specifies the IP port address of the remote host. The default value is 3709.

- **RemoteSSL=0|1**
  Enables SSL when communicating with another proxy server.

- **ReplyTimeOut=integer_value**
  Specifies the number of seconds that the JDBC server waits for a response from the CA IDMS system. The default, 0, causes the JDBC server to wait indefinitely.

- **SSL=0|1**
  Enables Secure Socket Layer (SSL) connections between this proxy server and a JDBC driver client.

- **Snap=0|1**
  Enables display of data buffers, sent and received, in the log file.

- **SocketTimeOut=integer_value**
  Specifies the number of seconds the JDBC server waits, or blocks, when reading data from a socket. While a socket is being read, the thread is blocked, and is not able to recognize an event that stops the thread. When this interval expires, the thread checks if the JDBC server is still running, and, if so, issues another read on the socket, continuing until the wait or reply timeout.
has expired. A high value reduces JDBC server overhead, while a low value allows the server to respond to shutdown events more quickly. Setting this to 0 causes the thread to block forever, and is not recommended. The default is 60 seconds.

- **Trace=0|1**
  Enables tracing of internal function calls. Output is written to the log file.

- **Unicode=0|1**
  Enables the use of Unicode as the character encoding when the JDBC driver is unable to use the requested encoding. The default value, 0, specifies the use of UTF-8, which is supported by all Java platforms.

- **WaitTimeOut=integer_value**
  Specifies the number of seconds that the JDBC server waits for a request from the JDBC driver before assuming the connection has been terminated. The default, 0, causes the JDBC server to wait indefinitely. It is usually best to set a timeout value to drop the connection when the client has been inactive for some reasonable time interval. For example, set this value to 1800 to specify a timeout of 30 minutes.

## Properties File Information

CA IDMS Server can use a standard Java properties file for configuration information on all platforms. A Java properties file is simply a text file where each property consists of a key name and value, separated by an equal sign (=). Comments can be included by prefacing them with a pound sign (#).

The properties file can include JDBC driver and JDBC server options which previously could be specified only in the configuration file. This allows Java options to be specified in a consistent format on all platforms, including those where the native methods are not implemented. Because the native methods do not use Java properties files, options that they use must be specified in the registry or configuration file.

The default name of the properties file is caidms.properties. You can override this name by specifying a system property, ca.idms.properties=filename. The Java class loader loads the file using the same rules for loading classes, so the properties file must be located in a directory included in the CLASSPATH. If more than one properties file exists, the first one found in the CLASSPATH directory list is loaded.

A sample properties file is installed in the product installation directory.

## Setting CA IDMS Server Options as Properties

**Contents**

- JDBC Driver Options (see page 97)
- JDBC Server Options (see page 97)
- Global Options (see page 97)
Any option that can be specified in the registry or configuration file can be specified in the properties file, or even as a system property. There are also options that can only be specified as properties. To specify a configuration file option in the properties file, prefix the key name with the section name. To specify a property as a system property, prefix it with ca.idms.

For example, you can enable the global JDBC trace in caidms.cfg on z/OS using the following:

```properties
[Options]
  JdbcTrace=1
```

This can also be specified in the caidms.properties file as:

```properties
Options.JdbcTrace=1
```

Or it can be specified as a system property as an argument to the java launcher with:

```
-Dca.idms.Options.JdbcTrace=1
```

An option value specified in the registry or configuration file overrides the value specified in the properties file, which in turn overrides the value specified as a system property.

### JDBC Driver Options

Options used by the JDBC driver can be specified in the [Options] or [datasource_name] section of the registry or configuration file, or prefixed with "Options." or a datasource_name in the properties file.

⚠️ **Note:** For more information on these options, see the appendices "Windows Registry Information" or "Configuration File Information," or the installed javadoc for the ca.idms.jdbc.IdmsConnectOptions class.

### JDBC Server Options

Options used by the JDBC server can be specified in the [Proxy] section of the registry or configuration file, or prefixed with "Proxy." in the properties file.

⚠️ **Note:** For more information on these options, see appendices "Windows Registry Information" or "Configuration File Information," or the installed javadoc for the ca.idms.proxy.ProxyOptions class.

### Global Options

Options that can only be specified in the properties file include:
• `cecp.network`
• `exclude`
• `include`
• `reload`
• `snap`
• `snap.bytes`
• `snap.native`
• `snap.object`
• `snap.sql`
• `trace`
• `trace.file`
• `trace.life`
• `trace.native`
• `trace.product`

⚠️ **Note:** For more information on these options, see the installed javadoc for the `ca.idms.io` TraceObject package.

The following property can be set in the properties file to enable the JDBC driver to get the external identity from compatible identity managers other than CA Siteminder.

`security.principal.class=<java.security.Principal_class_name>`

A compatible identity manager provides an implementation of the `java.security. Principal` interface that represents the user identity and can be accessed using the `javax.security.auth.Subject. getPrincipal` method.

**Note:** For more information on the `cecp.network` option see the installed javadoc for the `ca.idms.io` NativeCodePage class. Use of the `cecp.network` option causes IDMS Server to use CECP before the server-specified network encoding. This allows for the use of a customized set of code page tables using IDMS Server's CECP support.