CA IDMS™ Server

User Guide

r17

technologies

CA Technologies Product References

This document references the following CA Technologies products:

- CA IDMS™ Server
- CA IDMS™
- CA IDMS™ Visual DBA (CA IDMS VDBA)

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Introduction

CA IDMS Server provides open access to data stored in CA IDMS databases, letting you maintain existing corporate databases and make your data available to new client-server and web-based applications. CA IDMS Server provides support for dynamic SQL using both the ODBC and JDBC application program interfaces.

Note: Unless specifically designated, *Windows* refers to any supported Microsoft Windows operating environment.

This section contains the following topics:

Who Should Use this Document

This guide assumes you are a new user of CA IDMS Server with experience using Windows or z/OS, and have familiarity with CA IDMS and database access.

Use this document if you are a database administrator or system administrator, an ODBC or JDBC application developer, or the end user of an application using CA IDMS Server to access a CA IDMS database.

- If you are a database administrator or end user, use this document to define data sources to access CA IDMS data from a client application.
- System administrators should use this document to set up a CA IDMS system for access by CA IDMS Server.
- If you are an application developer, use this document to understand how the ODBC and JDBC Application Program Interfaces (APIs) requests are implemented by CA IDMS Server.

Components

CA IDMS Server consists of both host and client components.

- The *host component* is installed on your Central Version (CV) and performs license verification to permit access to the CV from the client components.
- The *client components* include the ODBC driver, the JDBC driver, the JDBC server, the native SQL client interface, and, on Windows, CAICCI/PC. These components have a client relationship with CA IDMS but from an application perspective, they form part of the application server.

ODBC and JDBC Drivers

The ODBC and JDBC drivers translate industry standard SQL requests into the form used by the native client interface. The native interface implements the same protocol used by CA IDMS online and batch applications written in COBOL and ADS. It communicates with CA IDMS from Windows and z/OS Unix System Services (USS) using CAICCI or TCP/IP. CAICCI uses TCP/IP for Windows clients and Cross Memory Services for z/OS clients.

ODBC Driver

The ODBC driver can be used on the Windows platform and implements the ODBC 3.5 specification. It also provides the functions defined in the ODBC 2.5 specification to continue support for older applications.

The ODBC driver always calls the native CA IDMS SQL client interface directly. The native interface uses CAICCI/PC to communicate with any supported release of CA IDMS running on z/OS or z/VSE. This is referred to as the CCI Protocol in this document. It can also use TCP/IP to communicate directly with CA IDMS r17, or later, referred to as the IDMS protocol.

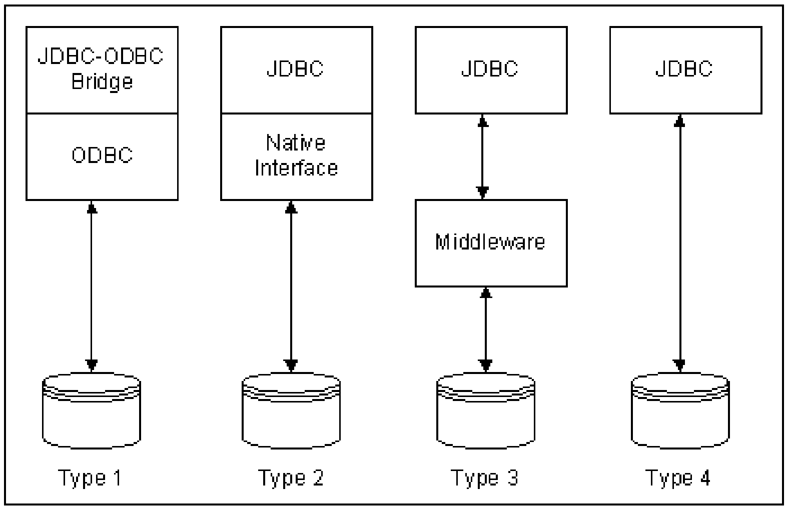
JDBC Driver

You can use the JDBC driver on Java 1.6 (or later) platforms, including those on Windows, z/OS, Linux, and UNIX. The JDBC driver implements the JDBC 4.0 specification.

There are four types of JDBC drivers that differ in the way they communicate with the database and whether they use native code on the client platform:

- **Type 1 JDBC driver**—Uses an ODBC driver to communicate with the database. ODBC drivers are always implemented in native code. The JDBC-ODBC Bridge is a basic Type 1 driver that is rarely used now that database specific drivers are available.
- **Type 2 JDBC driver**—Invokes the native client interface to communicate with the database.

- **Type 3 JDBC driver**—Uses a generic network protocol to communicate with a middleware server that invokes the native client interface to communicate with the database. It uses no native code on the client platform.
- **Type 4 JDBC driver**—Communicates directly with the database using its proprietary protocol. It uses no native code on the client platform.



The CA IDMS Server JDBC driver is a universal driver that can function as a Type 2, 3, or 4 driver on supported platforms depending only on the connection parameters specified.

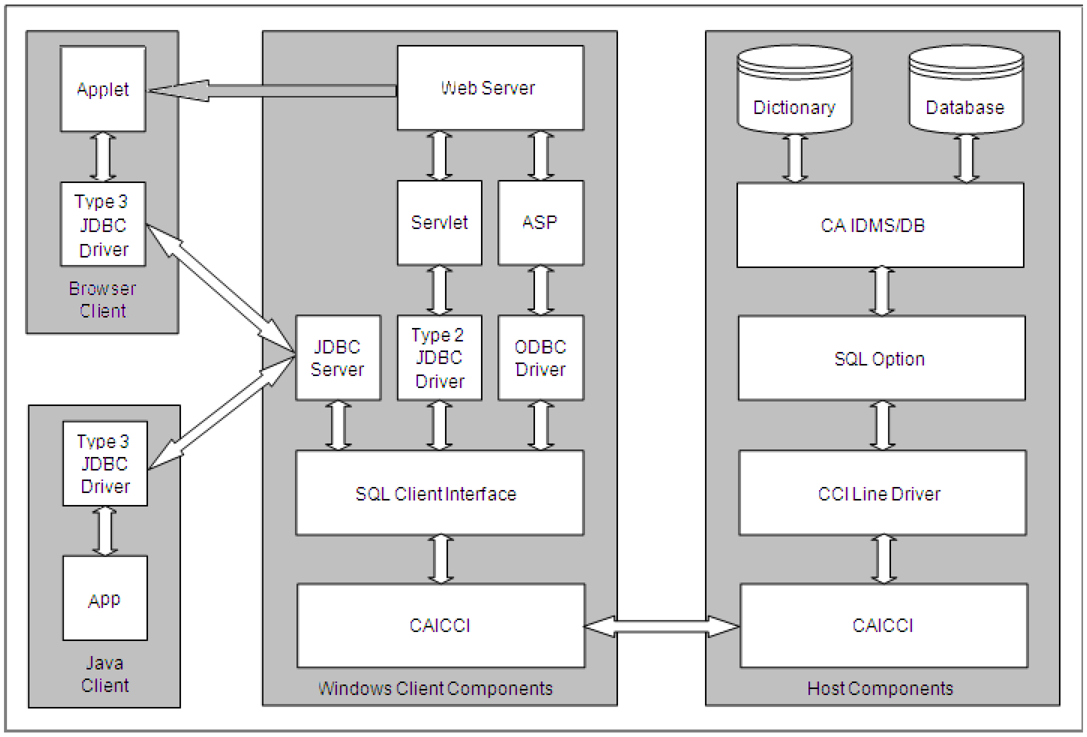
The JDBC driver can be used as a Type 2 driver on Windows and z/OS. It calls the native SQL client interface, which can use CAICCI to communicate with any supported release of CA IDMS running on z/OS or VSE. CAICCI uses TCP/IP from Windows and cross memory services on z/OS.

The JDBC driver can be used as a Type 3 driver on any Java platform. It uses TCP/IP to communicate with the JDBC server, which can run on any Java platform. When the JDBC server runs on Windows or z/OS, it can invoke the native SQL client interface and CAICCI to communicate with any supported release of CA IDMS running on z/OS or VSE. The JDBC server can use TCP/IP from any Java platform to communicate directly with CA IDMS r16 SP2 or later running on z/OS, VSE, or z/VM.

The JDBC driver can be used as a Type 4 driver on any Java platform. It uses TCP/IP to communicate directly with CA IDMS r16 SP2 or later running on z/OS, VSE, or z/VM.

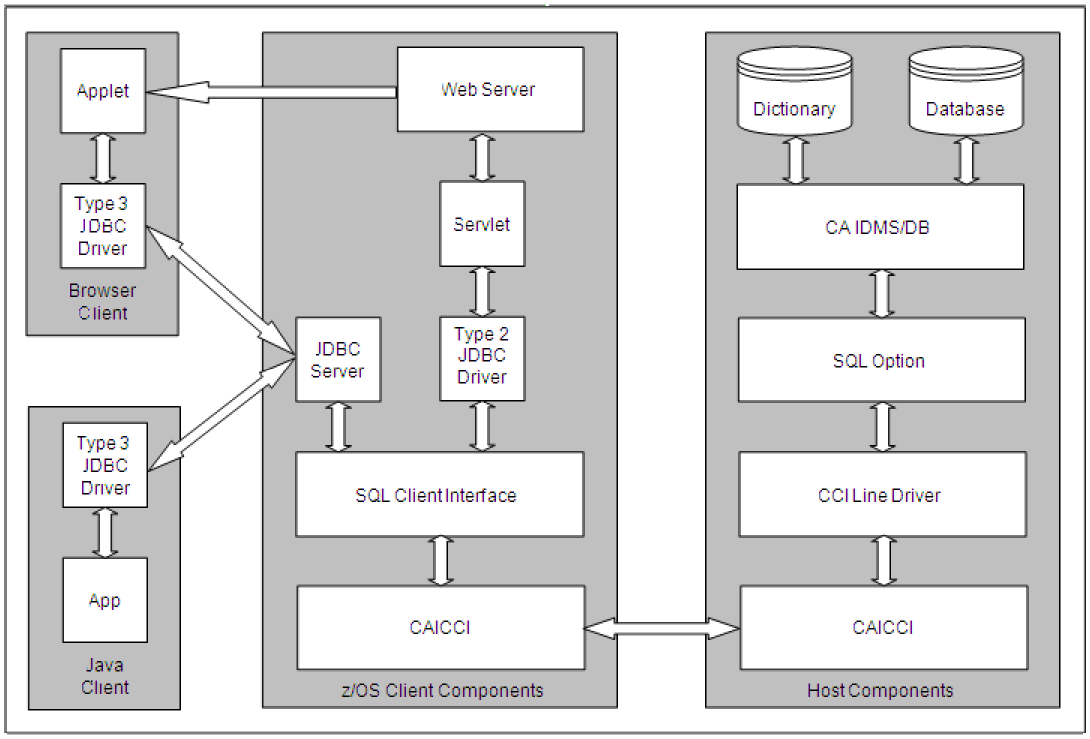
Server Architecture

The following diagram illustrates the way in which the CA IDMS Server software components fit together when the client platform is Windows:



Both the ODBC and JDBC drivers are supported for Windows applications, represented here by the web server. The Windows client and CA IDMS communicate using TCP/IP. In this diagram the combination of the web server, Servlet, and ASP boxes represent the applications. CA IDMS Server also supports traditional ODBC and JDBC client-server applications.

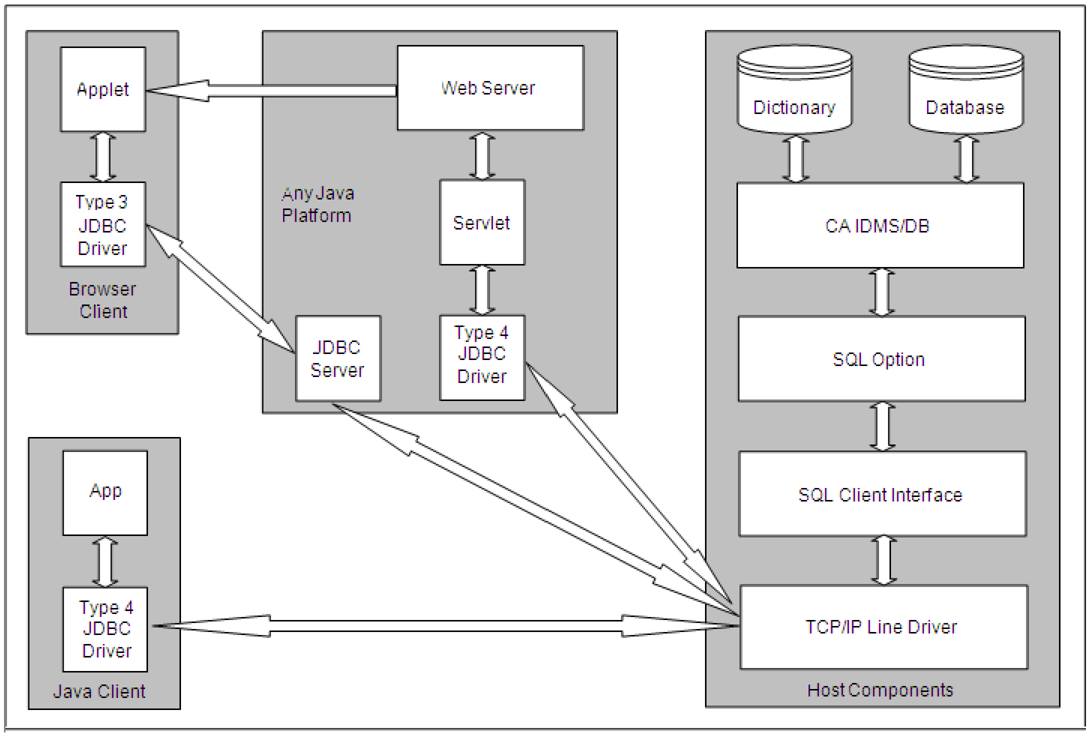
The following diagram illustrates how the CA IDMS Server software components fit together when the client platform is z/OS:



The JDBC driver is supported for z/OS applications, represented here by the web server. The z/OS client and host components communicate using Cross Memory Services.

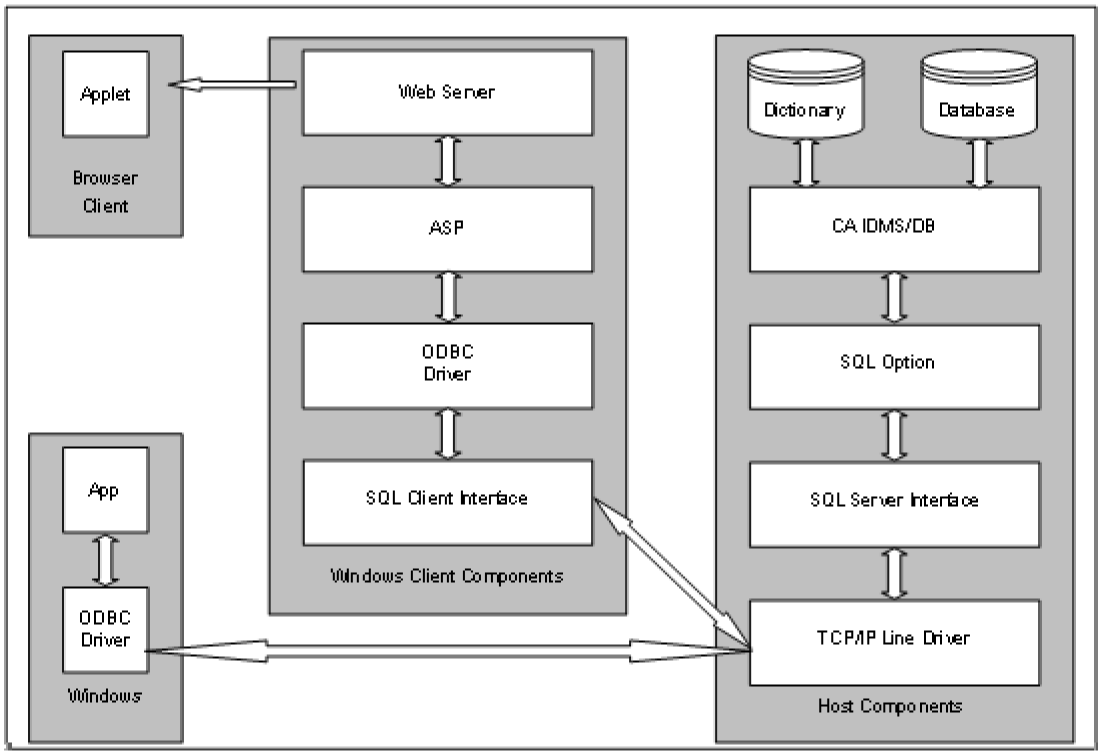
The host components are the same whether the native client is installed on Windows or z/OS.

The following diagram illustrates how the CA IDMS Server JDBC driver software components fit together from any Java platform when used with CA IDMS r16 SP2 or later:



The Type 4 JDBC driver and JDBC Server communicate from any Java platform directly to the CV using TCP/IP.

The following diagram illustrates how the CA IDMS Server ODBC driver software components fit together from Windows when used with CA IDMS r17 or later:



The Wire Protocol ODBC driver communicates from Windows directly to the CV using TCP/IP.

More Information

We recommend that you refer to the following documents when setting up your CA IDMS system to work with CA IDMS Server:

- *CA IDMS Installation and Maintenance – z/OS Guide*
- *CA IDMS System Generation Guide*
- *CA IDMS System Operations Guide*
- *CA IDMS Database Administration Guide*

- *CA IDMS Database SQL Option Reference Guide*
- *CA IDMS Database SQL Option Programming Guide*
- *CA IDMS DML Reference - COBOL Guide*
- CA Common Services documentation

Information about ODBC is available from the Microsoft website. Information about JDBC is available from Sun Microsystems' JavaSoft website: http://www.java.sun.com

Chapter 2: Preparing for Installation

This section contains the following topics:

Mainframe Software Prerequisites

CA IDMS Server requires the installation of the CA IDMS SQL option. The host component of CA IDMS Server is delivered on the CA IDMS installation media. CA IDMS r16 SP2, or later, is required to use the Type 4 JDBC driver and distributed transactions. CA IDMS r17, or later, is required to use the Wire Protocol ODBC driver. See the *CA IDMS Installation and Maintenance Guide* for more information about installing CA IDMS Server on the mainframe.

Use of the ODBC or JDBC driver with the CCI protocol requires the installation of the CA Common Services components CAIENF (Event Notification Facility) and CAICCI (Common Communications Interface) on the mainframe. For information about installing CAIENF and CAICCI on the mainframe, see your CA Common Services documentation.

Note: The CA Common Services components are not required to use the Type 4 JDBC driver or Wire Protocol ODBC driver.

z/OS Software Prerequisites

The JDBC driver and JDBC server run in the UNIX System Services (USS) environment on z/OS. They must be installed and set up on the mainframe. The IBM SDK for z/OS Java 2 Technology Edition V1.6 or later must also be installed and available in the USS environment. TCP/IP must be installed and configured correctly to use the JDBC server. For more information about installing USS, TCP/IP, and the SDK, see IBM's documentation.

Use of Secure Sockets Layer (SSL) with CAICCI requires r2.1, or later, of CAICCI which is part of CA Common Services r11 SP6. For more information, see the CA Common Services documentation.

Use of SSL with the JDBC Type 4 driver or ODBC Wire Protocol driver requires z/OS r1.7, or later, and Application Transparent - Transport Layer Security (AT-TLS). For more information, see PIB QI83006 on http://ca.com/support.

Linux Software Prerequisites

The Java Runtime Environment (JRE) 1.6, or later, is required to use the CA IDMS Server client on Linux and other platforms.

Windows Software Prerequisites

The CA IDMS Server client requires Microsoft Windows XP SP2, Windows Server 2003 SP1, Windows Vista, Windows Server 2008, or Windows 7. Java 1.6 or later is required for JDBC support. CAICCI/PC and the ODBC Driver Manager are required to use the ODBC driver. Both 32 and 64 bit editions of Windows are supported.

CAICCI/PC

CAICCI/PC provides a common interface between CA IDMS Server and the TCP/IP protocol. CAICCI/PC r2.1 and higher supports SSL when communicating with an SSL-enabled mainframe CCITCP address space on z/OS. CAICCI/PC is distributed on the CA Common Services tape for your mainframe operating system. The 32-bit version CAICCI/PC is included on the CA IDMS Server installation CD as part of the CA IDMS Server installation for the Windows 32-bit editions. The 64-bit version of CAICCI/PC is installed automatically with the 64-bit ODBC driver on Windows 64-bit editions.

CAICCI/PC *must* be installed to use ODBC with versions of CA IDMS prior to r17 or to use JDBC with versions of CA IDMS prior to r16 SP2. It is not necessary to install CAICCI/PC on client workstations which use the ODBC Wire Protocol or JDBC Type 3 or Type 4 drivers.

Note: For more information about CAICCI/PC, see the appropriate CA Common Services manual.

ODBC Driver Manager

The ODBC Driver Manager provides the link between ODBC-enabled applications and ODBC drivers and is required by the ODBC driver.

The ODBC Driver Manager is not required to use the JDBC driver, but it may be convenient to use the ODBC Administrator included with it to define connection information used by the Type 2 JDBC driver.

The ODBC Driver Manager is installed automatically by many Microsoft products, including Windows, Office, and Internet Explorer. It is also included in the Microsoft Data Access Components (MDAC), along with ActiveX Data Objects, the OLE DB Provider for ODBC, and updated ODBC drivers for Microsoft SQL Server, Microsoft Access, and Oracle. Additional information about ODBC and MDAC is available from the Microsoft website.

The latest version of MDAC can be downloaded (at no cost) directly from the Microsoft website (www.microsoft.com).

Java Runtime Environment

A Java Virtual Machine (JVM) is an interpreter that executes Java programs, which are stored on disk as class files. CA IDMS Server r17 conforms to the JDBC 4.0 specification and requires a Java 1.6 or later JVM.

The JVM is delivered in the Java Runtime Environment (JRE). The JRE is suitable for deploying applications. A Java Software Development Kit (JDK) is required to develop applications. The JDK includes the JRE and is available from Sun Microsystems. The JRE is available as a free download from Sun Microsystems (www.java.sun.com) and IBM (www.ibm.com).

The Microsoft Java VM supports Java 1.1 and JDBC 1.2. It cannot run class files compiled with Java 1.2 or later. The class file format is incompatible and necessary run-time classes are missing. For these reasons, the Microsoft Java VM is not compatible with CA IDMS Server r17.

The Sun Java Plug-In is required to use the JDBC driver with applets running in Internet Explorer. The Java Plug-In is installed with the Sun JRE.

Note: For detailed information about CA IDMS Server conformance to the JDBC standard, see the chapter "JDBC Programmer Reference." Additional information about JDBC and Java is available from Sun Microsystems' website: http://www.java.sun.com

Installing the Java Runtime Environment

The Type 4 JDBC driver performs all data conversion on the client platform, using the character converter classes provided by the Java Runtime Environment (JRE). The JRE includes converter classes for most of the character encodings in use around the world. However, by default, the JRE installer installs only European language support on machines that support only European languages. The mainframe encodings based on EBCDIC, such as Cp037, are not included.

The JDBC driver includes built-in support for Cp037 and Cp1047. For other mainframe character sets, include the complete set of character encodings when installing the JRE by selecting the Custom option and then Support for Additional Languages.

This is generally not necessary when using the JRE included with the Java Software Development Kit (JDK), which includes charsets.jar.

Note: For details on managing SSL certificates and keystores for JDBC Type 3 and 4 drivers, see Sun's documentation on JSSE, the Java Secure Sockets Extension.

Delivery of Components

Delivery of the software components utilized by CA IDMS Server is shown in the following table:

Software Component	Base Product	Installation Medium
CA IDMS Server ODBC and JDBC drivers	CA IDMS Server	CA IDMS Server CD
CAICCI/PC	CA Common Services	Base product tape or CA IDMS Server CD
CAICCI and CAIENF	CA Common Services	Base product tape
CA IDMS	CA IDMS	CA IDMS tape
CA IDMS Server (host component)	CA IDMS Server	CA IDMS tape
ODBC Driver Manager	MDAC	Download
Java Virtual Machine	JRE or JDK	Download

Chapter 3: Setting Up Your System

This chapter describes the mainframe procedures necessary to establish communications between Windows applications and CA IDMS using CA IDMS Server, and provides information to help you access existing CA IDMS databases. Also included is a description of how network records appear to SQL, and information about numeric data and pseudo-conversational processing.

This section contains the following topics:

Installing the Host Component

CA IDMS Server is licensed on the mainframe. For CA IDMS r17 and earlier it is enabled by linking an object module, RHDCD0LB with the CAICCI line driver, RHDCD0LV, and the CA IDMS Server listener, IDMSJSRV. The object module is delivered on the CA IDMS tape for r15.0 and later.

Setting Up CA IDMS Server

Each CV to be accessed by CA IDMS server must be generated with the following definitions:

- A CCI line to use the ODBC driver or JDBC driver with the CCI protocol.
- A CASERVER task for the CCI line
- A TCP/IP line to use the ODBC or JDBC deriver with the IDMS TCP/IP protocol.
- A listener PTERM for the TCP/IP line
- An IDMSJSRV task for the TCP/IP line
- A PTERM/LTERM pair for each concurrent connection on each line
- The SQL definitions in the catalog area of the dictionary associated with the CA IDMS database
- The startup JCL may also need to be modified for the TCP/IP line.

The following sections describe the procedure to define the CA IDMS system generation parameters to support communications using CA IDMS Server.

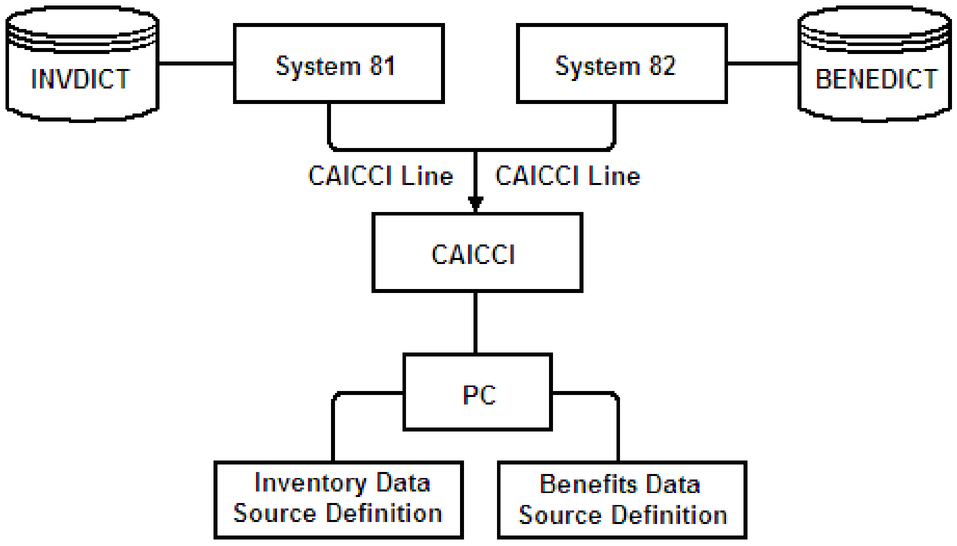
The examples in this chapter assume that CA IDMS Server is used to access two sample CA IDMS systems, System 81 and System 82, from a PC.

An Inventory dictionary, INVDICT, is associated with System 81. Associated with System 82 is BENEDICT, a Benefits dictionary. INVDICT and BENEDICT contain the definitions of the tables to be accessed from the PC. Data source definitions on the PC refer to the mainframe dictionary names.

Note: For detailed information about these system generation statements and how to use the system generation compiler, see the *CA IDMS System Generation Guide*. For detailed information about configuring and maintaining CA IDMS systems including startup JCL, see the *CA IDMS System Operations Guide*.

Defining the CA IDMS System Using CAICCI

CAICCI provides communication between mainframes or between mainframes and PCs. CAICCI r2.1 and higher provides Secure Sockets Layer (SSL) support for the ODBC and JDBC drivers. The following diagram illustrates the sample CA IDMS system network using CAICCI:


INVDICT
System 81
System 82
BENEDICT
CAICCI Line
CAICCI Line
CAICCI
PC
Inventory Data Source Definition
Benefits Data Source Definition

Defining a CCI Line

A CAICCI line connects a CV with the CAICCI network. Define a CAICCI line in each CA IDMS system to be accessed by CA IDMS Server.

Add one CAICCI line, plus a physical terminal (PTERM) and logical terminal (LTERM) pair for each concurrent connection with the CA IDMS system.

Each ODBC or JDBC connection uses one PTERM/LTERM pair. For example, if you have a single PC running a World Wide Web server application, and you want to support concurrent access to CA IDMS by 25 World Wide Web browser clients, define 25 PTERM/LTERM pairs.

Example:

The following example defines a CAICCI line for System 81. The LINE, PTERM, and LTERM statements define a CAICCI line with two physical terminals, allowing two PC users to be logged on to System 81 at the same time:

```
ADD SYSTEM 81
    SYSTEM ID IS SYST0081.

ADD LINE CCILINE
    TYPE IS CCI.

ADD PTERM PTECCI01
    TYPE IS BULK.

ADD LTERM LTECCI01
    PTERM IS PTECCI01.

ADD PTERM PTECCI02
    TYPE IS BULK.

ADD LTERM LTECCI02
    PTERM IS PTECCI02.
```

Creating the CASERVER Task

The TASK statement defines a task and its characteristics, including the code used to invoke the task. The default task code is CASERVER. You can override the default task code if you want to control resources per user or to apply additional security.

The CASERVER task code is similar to the RHDCNP3S task code, which controls the resource limits and time-out values for CA IDMS external user sessions.

To define a CASERVER task code

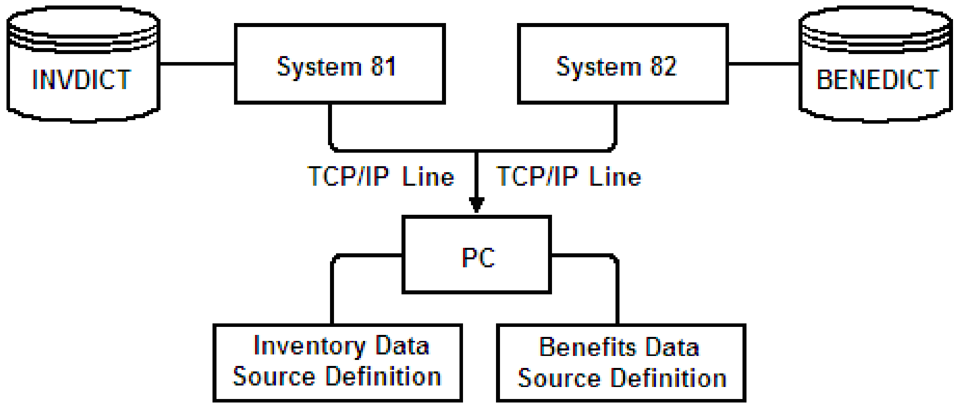
1. From the system-generation compiler, enter this command:
   `DISPLAY TASK RHDCNP3S AS SYNTAX.`

   The system-generation compiler displays the definition of the RHDCNP3S task code.
2. Erase the DISPLAY TASK statement at the top of the screen.
3. Change the name of the task code, RHDCNP3S, to CASERVER (or the task code name you have chosen). Modify the task definition to add the INTERNAL parameter, if it is not already there, and to set an appropriate value, in seconds, for the EXTERNAL WAIT parameter for your users (for example, set 1800 for a 30-minute wait).

Note: If you do not define a CASERVER task code, CA IDMS uses the RHDCNP3S task code to define the characteristics for a CA IDMS Server session.

Defining the CA IDMS System Using TCP/IP

TCP/IP provides direct connection between a client system using the JDBC type 4 driver and CA IDMS r16 SP2 or later, or using the ODBC wire protocol driver and CA IDMS r17 or later. The following diagram illustrates the sample CA IDMS system network using TCP/IP:


INVDICT
System 81
System 82
BENEDICT
TCP/IP Line
TCP/IP Line
PC
Inventory Data Source Definition
Benefits Data Source Definition

Updating the System Startup JCL

For DNS functions to operate correctly, a SYSTCPD card must be added to the central version JCL.

Defining a TCP/IP Line

Define a TCP/IP line in each CA IDMS system to be accessed by CA IDMS Server.

Add one TCP/IP line, a listener PTERM, plus a physical terminal (PTERM) and logical terminal (LTERM) pair for each concurrent connection with the CA IDMS system.

Define the listener PTERM/LTERM pair for the built-in server program, IDMSJSRV. This PTERM must specify the RHDCNP3J task defined during installation, SYSTEM mode, and the port used by the driver. The default port used by the drivers and registered with the Internet Assigned Number Authority (IANA) for CA IDMS is 3709. This can be used if only a single DC/UCF system is used on the host machine. Otherwise, a recommended convention is to append the system number to 37. The listener PTERM should also specify a secured task.

Each JDBC or ODBC connection uses one PTERM/LTERM pair.

Example:

The following example defines a TCP/IP line for System 81. The LINE, PTERM, and LTERM statements define a TCP/IP line with 100 physical terminals, allowing 100 concurrent JDBC or ODBC connections.

```
ADD LINE TCPIP
        TYPE IS SOCKET
        MODULE RHDCD1IP.
ADD PTERM TCPJSRV
        TYPE IS LISTENER
        PORT IS 3781
        TASK IS RHDCNP3J
        MODE IS SYSTEM
        PARM IS 'TASK=IDMSJSRV'.
ADD LTERM TCLJSRV
        PTERM IS TCPJSRV.
ADD PTERM TCP0001
        TYPE IS BULK
        REPEAT COUNT IS 99.
ADD LTERM TCL0001
        PTERM IS TCP0001.
```

The TCP/IP line can support SSL using the AT-TLS facility of z/OS 1.7 and later. For more information, see PIB QI83006 on supportconnect.ca.com.

Creating the IDMSJSRV Task

You must grant execute authority on task RHDCNP3J to group PUBLIC or all groups because the line driver invokes it before the user id and password are received. Alternatively, you can turn off security for task RHDCNP3J by including an entry in the SRTT.

The task specified in the listener PTERM definition PARM string can be restricted to specific users or groups. IDMSJSRV is a sample task that is similar to the RHDCNP3J task that can be secured. You can also define and secure a different task code and override it at runtime.

Note: For more information about securing tasks, see the *CA IDMS Security Administration Guide*.

Setting Up Database Access

The ODBC and JDBC drivers use dynamic SQL to access a CA IDMS database from an ODBC or JDBC application. Both the SQL Option and the host component of CA IDMS Server must be installed on the CV. The database can be defined using the Schema compiler or SQL Data Description Language (DDL). In either case, you must include the appropriate SQL definitions in the dictionary associated with the CA IDMS system. The SQL definitions reside in the catalog area of the dictionary.

Setting Up SQL Access

The following suggestions are useful when setting up SQL access to CA IDMS databases:

- To access a non-SQL-defined database using SQL, define an SQL schema that identifies the network schema and the segment where the data is stored. The network schema must conform to the rules described throughout this chapter.
- If the application does not qualify table references with schema names, define one or more CA IDMS/DC profiles that set a default schema name. You may need to ask your Database Administrator (DBA) for assistance.
- To limit the size of the list of tables returned by the driver metadata functions, create an Accessible Tables View that returns a subset of the default view, and set it as an option for a specific data source. Using such a view of accessible tables can generate a more meaningful list of tables for each user and improve performance.
- Define views in the catalog to provide easy access to non-SQL-defined databases or application-specific data. For example, consider using a view when joining tables using the set-name condition. However, if you choose to do so, remember that views created by joining two or more tables cannot be updated.
- Implement table procedures to provide easy access to non-SQL-defined databases or application-specific data. For example, consider using a table procedure to navigate a complicated network database. Table procedures can also be used to update databases.

Utilizing Page Groups

A page group is a physical database definition attribute set by the database administrator during database definition. The catalog and the target database can be in different page groups. Unless the Mixed Page Group feature of CA IDMS is activated, tables from mixed page groups cannot be accessed with a single request. Additionally, once a table from one page group has been accessed, a COMMIT command must be issued before a table from a different page group can be accessed.

To use data sources defining data from mixed page groups:

- Define a different data source and a different Accessible Tables View for each page group when the catalog contains definitions of tables from mixed page groups. Each Accessible Tables View should include tables from a single page group so that the end user cannot accidentally access mixed page groups after a table list function is performed.
- Use the Automatic Commit option when accessing tables in different page groups.

  Note: Automatic Commit is enabled by default, and can be disabled using an ODBC or JDBC function.

Mixed page groups are supported starting with CA IDMS r14.1, so these restrictions do not apply when the data source is on a 14.1 or later system and the Mixed Page Group Binds feature has been activated.

Note: For more information about using page groups, see the *CA IDMS Database Administration Guide - Volume 1*.

Setting Up SQL Access to Non-SQL Databases

This section reviews the transformations used by the SQL engine when reading definitions of non-SQL records. When using SQL to access non-SQL records, the entity names coded in the SQL syntax must follow the conventions described in the following sections.

Accessing Non-SQL Records Using SQL Statements

To reference an SQL table in SQL statements, code the table name preceded by a schema name qualifier. For example, in this statement:

```
SELECT * FROM DEMOSCH.SAMPLE
```

SAMPLE is the table name and DEMOSCH is the SQL schema in which it is defined.

The combination of schema name and table name allows the SQL compiler to look up the definition of the table in the SQL catalog.

To access a non-SQL record from an SQL statement, code the record name in the same way. Define an SQL schema that maps to the corresponding non-SQL schema, and use the SQL schema name to qualify all subsequent references to non-SQL records in SQL DML statements. For example:

```
CREATE SCHEMA SQLNET

    FOR NONSQL SCHEMA PRODDICT.CUSTSCHM;

SELECT * FROM SQLNET."ORDER-REC";
```

Note: For more information about defining SQL schemas, see the *CA IDMS Database SQL Option Reference Guide* for syntax and information about accessing non-SQL databases and *CA IDMS Database Administration Guide* for process-related information.

Transforming Non-SQL Record and Set Names

Non-SQL record and set names may contain embedded hyphens, which are allowed in the naming conventions for non-SQL schemas, but not in the naming conventions for SQL schemas. To use record and set names with embedded hyphens in an SQL statement, enclose the names in double quotes, for example, "CUST-REC-123."

Transforming Non-SQL Element Names

In non-SQL element names, CA IDMS automatically transforms embedded hyphens to underscores when they are referenced through SQL. For example, to access the CUST-NUMBER element in a non-SQL record, you must code CUST_NUMBER in an SQL statement.

Creating SQL Synonyms

When a FOR LANGUAGE SQL synonym is defined for a non-SQL record, CA IDMS uses the element synonyms for all SQL access. SQL synonyms are used *only* for element names.

Defining SQL synonyms for non-SQL records is sometimes the only way to overcome column name limitations within SQL. Some non-SQL element names do not make satisfactory SQL column names, even after hyphens are changed to underscores. For example, if a non-SQL element name begins with a numeric character, you must still use double quotes around the element name. For example, to access 123-ORD-NUM, you would code "123_ORD_NUM" in an SQL statement.

Elements that cannot be Transformed

Group elements, REDEFINES elements, FILLERS, and OCCURS DEPENDING ON elements are not available for access by SQL. To the SQL user, it is as if these elements were not defined in the non-SQL record. The subordinate elements of a group definition are available for access, as are the base elements to which a REDEFINES is directed.

Fixed OCCURS Element Definitions

Although OCCURS…DEPENDING ON declarations are not available for SQL access, fixed OCCURS definitions are available. To the SQL user, a fixed OCCURS element appears as one column for each occurrence of the element. The column name for each occurrence is the original element name followed by an underscore and an occurrence number. If the element is declared with multiple OCCURS levels, the corresponding column names contain one underscore and one occurrence number for each dimension of the OCCURS declaration.

For example, the element definition BUD-AMT OCCURS 12 TIMES generates the following column names:

```
BUD_AMT_01, BUD_AMT_02, BUD_AMT_03...BUD_AMT_12.
```

Note: The occurrence number attached to the column name must be large enough to accommodate the largest subscript from the corresponding element definition.

The base element name, combined with the appended occurrence information, cannot have more than 32 characters. If it does, you must define an SQL synonym for the non-SQL record.

Although the CA IDMS SQL implementation allows 32-character column names, other SQL implementations restrict column names to 18 characters. Some ODBC client software, in particular, may require SQL synonyms for non-SQL records to limit the size of the transformed column names to 18 characters.

Note: Another way to define shorter names is to create a view of the record and specify view column names.

Defining Keys

To access a control-key definition (of a CALC, INDEX, or sorted set) using SQL, the control-key definition must not include a FILLER element. If it does, change the non-SQL record definition, assigning a name other than FILLER to the elements in question.

Additionally, the control-key definition cannot incorporate the subordinate elements of a group level REDEFINES when these elements are smaller in size than the base element being redefined, as in the following example:

```
02 ELEM1 PIC X(8).
02 ELEM1REDEF REDEFINES ELEM1.
03 ELEM1A PIC S9(8) COMP.
03 ELEM1B PIC S9(8) COMP.
```

An error occurs if ELEM1A and ELEM1B are used in the control key definition, because they are smaller than the element they redefine (although combined they are equal to ELEM1). When this condition occurs, change the redefining group, which contains the smallest subordinate elements, into the base-element definition. Use the base-element definition in the control key specification. For example, ELEM1REDEF should be the base-element definition in the sample above, and ELEM1 should be coded so that it redefines ELEM1REDEF.

Note: For more information about accessing non-SQL-defined databases using SQL, see the *CA IDMS Database SQL Option Reference Guide*.

Defining Sets as Referential Constraints

Because members of sets usually do not contain foreign keys that reference the owner record, the set specification statement can be used to include the set name in the join criteria for two non-SQL records. For example:

```
SELECT M.NAME FROM OWNER O, MEMBER M
       WHERE O.NAME='BILL' AND "OWNER-MEMBER"
```

- The schema compiler SET definition PRIMARY KEY and FOREIGN KEY options can be used to define primary and foreign keys for non-SQL records. Adding foreign keys to set definitions provides several advantages for applications that use SQL and can enhance the compatibility of CA IDMS with tools that build SQL applications.
- Referential integrity is enforced when SQL is used to update non SQL records.
- Standard SQL predicates can be used in join statements.
- Standard JDBC and ODBC foreign key metadata APIs can be used to discover relationships between non-SQL records accessed using SQL.
- The foreign key columns must be elements in the member record. If these elements are added to an existing record, you must restructure the database, populate the foreign key fields, and recompile network applications that reference the member record.
- Network applications that STORE or CONNECT members in the set must update the foreign keys to maintain referential integrity. You can use subschema views that do not include the added fields to minimize changes to programs that do not access the foreign keys.

Note: For more information about modifying set definitions, see the *CA IDMS Database Administration Guide*.

Setting up Catalog Views

Both ODBC and JDBC provide metadata APIs that return information about schemas, tables, columns, and indexes from the SQL catalog. CA IDMS Server uses table procedures and views in the SYSCA schema to access catalog information. Typically, these are installed into the catalog when CA IDMS is installed. Additional views may require definition, depending on the version of CA IDMS you use.

SYSCA.ODBC_INDEX

This table procedure is used to return index and CALC key information for network records as well as SQL tables. The ODBC driver uses this table procedure to implement the SQLStatistics, SQLSpecialColumns, SQLPrimaryKeys, and SQLForeignKeys functions. If this table procedure is not installed, the ODBC driver queries the catalog SYSTEM.INDEX and SYSTEM.INDEXKEY tables, and returns information only for SQL defined tables.

The JDBC driver uses this table procedure to implement the corresponding DatabaseMetaData.getIndexInfo, getBestRowIdentifer, getPrimaryKeys, getExportedKeys, getImportedKeys, and getCrossReference methods. This table procedure is required for JDBC support.

This table procedure is defined for all releases since Genlevel 9506 of CA IDMS 12.01.

SYSCA.ACCESSIBLE_ SCHEMAS

This view returns a list of schemas containing tables accessible to the user. It is used by the JDBC driver to implement the DatabaseMetaData.getSchemas method, and is required for JDBC support. The ODBC driver does not use this view. This view is defined for releases after CA IDMS r15.0.

SYSCA.ACCESSIBLE_PROCS

This view returns a list of procedures. It is used by the JDBC driver to implement the DatabaseMetaData.getProcedures method and by the ODBC driver SQLProcedures function. This view is defined for CA IDMS r16.0 and later.

Defining Catalog Views

These views are added to the catalog when you upgrade CA IDMS and follow the instructions for upgrading the catalog. DDL to create them is also provided in the file accviews.txt, which is installed in the Java subdirectory on Windows, and the product installation root directory on other platforms. These views should be defined in all catalogs that are accessed using CA IDMS Server.

Passing Auditing Information to CA IDMS

You can use CA IDMS Server to pass auditing information from the client to the CA IDMS system. This auditing information includes be site-defined accounting information supplied when the application connects to the database and the user identity as is it known to the application, which may be different from the user id used to sign on to CA IDMS.

Supplying Accounting Information

You can supply as many as 32 bytes of accounting information using one of the following methods:

- Enter the value in the Account field of the ODBC DriverConnect dialog.
- Pass the value as a connection attribute with the key **ACCT** to the ODBC SQLDriverConnect function.
- Pass the value as a DriverPropertyInfo object with the key **acct** to the JDBC DriverManager.getConnection method.
- Pass the value as an IdmsDataSource property.

The first character of accounting information must be a space or the information must be ignored.

Note:

- For information about the ODBC DriverConnect dialog, see the chapter "Using the Client on Windows."
- For information about the ODBC SQLDriverConnect connection string, see the chapter "ODBC Programmer Reference."
- For information about the JDBC DriverManager getConnection method, see the chapter "JDBC Programmer Reference."

Using Accounting Information

The supplied accounting information is passed to the back end, and is stored in an area accessible from the PTE for use by accounting and security exits.

For example, the information could be used by either exit 4 or 5 to change header information in the statistics block described by #STRDS.

The following table describes how the accounting data can be found:

Field Name	Control Block	Comments
TCELTEA	TCE	Points to the LTE. Determine whether it is a CAICCI-related LTE (type is LTEBULK).
LTEPTEA	LTE	Points to the PTE.
PTELACCT	PTE	Points to a 32-byte accounting information area. This area must be binary zeros if no accounting information was passed from the PC or if the front end is not a PC.

When using the Type 4 JDBC driver with CA IDMS r16 SP2 or later, you can add this information to the user profile at sign on. Define a system or user profile attribute and specify the ACCT=profile-key-name option in the listener PTERM definition. This allows SQL statements to access the accounting information using the PROFILE function. Procedures written in COBOL or CA ADS can use the IDMSIN01 GETPROF callable service to access the accounting information.

Note:

- For more information about programs invoked by user exits, see the *CA IDMS System Operations Guide*.
- For more information about field names in various IDMS control blocks, see the *CA IDMS DSECT Reference Guide*.
- For information about defining the listener for the Type 4 JDBC driver, see the "Setting Up Your System" chapter and the *CA IDMS System Generation Guide*.

Example

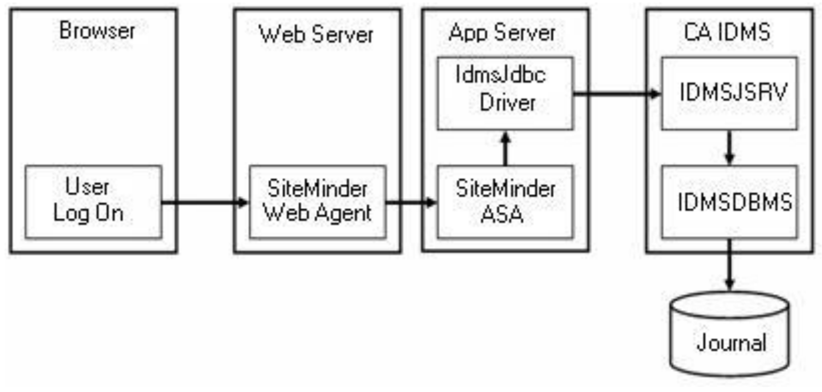
The following is sample code for locating accounting information:

```
USING TCE,R9

    L    R5,TCELTEA           Get the LTE

USING LTE,R5

    CLI  LTETYP,LTEBULK       Is this a BULK type
    BNE  RETURN               No....Return
    L    R6,LTEPTEA           Get the PTE address

USING PTE,R6

    L    R7,PTELACCT          R7 points to 32-byte area
```

Setting the External Identity

The user of a web-based application is typically an external user or customer who logs on to the application using an identity manager such as CA SiteMinder. The external user identity used to sign on to the web application is generally not the user ID used to access the back end database. Instead, a generic user ID associated with the application is used by all connections initiated by the application. CA IDMS Server and CA IDMS can record the external user identity in the journal when a transaction is started. This provides an end-to-end audit trail that correlates database activity with the external users who initiate each transaction.

When running in an application server managed by CA SiteMinder, the JDBC driver gets the external identity from the SiteMinder Application Server Agent when a new transaction is started. If this identity has changed since the last transaction, the driver sends it to CA IDMS.



You can set an external user identity in a standalone JDBC application using the IdmsConnection setIdentity method. You can set the external user identity in an ODBC application using the SQLSetConnectAttr function with the IDMS_ATTR_EXTERNAL_IDENTITY attribute type.

Notes:

- For more information about CA SiteMinder and CA IDMS prerequisites, see the chapter "Preparing for Installation."
- For more information about enabling identity auditing, see the chapter "JDBC Programmer Reference."
- For more information about using other identity managers, see the appendix "Properties File Information."
- For more information about driver-specific attributes, see the chapter “ODBC Programmer Reference."

Auditing the External Identity

When an external identity has been set for the user session, CA IDMS includes its value along with actual user ID in the BGIN transaction journal record. Journal reports can be used to audit this information.

Journal Analyzer Chronological Event Report

The report for the BGIN record includes the external user identity, if present.

```
--------EVENT--------- ----------IDENT--------- ---QUIESCE LVL/USER/EXT ID –-
TIME    TYPE   DURATION  RUN UNIT      PROGRAM

hh:mm:ss  BGIN                1633 JAVAPROG ONL X
```

JREPORT 000

Supports a REC card that allows the use of the external identity as selection criteria when running existing JREPORTs.

JREPORT 008

Displays the external identity information when reporting the BGIN record.

```
BGIN   TECHDC30  12/27/05  20.07.13.56     529276      170604 ….
       USER ID                        EXTERNAL ID
       ABBWI01                        BILL2007
```

JREPORT 010

Lists the user ID, external identity, date, time, program name, and run unit id. This provides enough information to run JREPORT 008 with SELECT criteria to provide details on any activity involving a particular user.

```
REPORT NO. 10                       IDMS JOURNAL REPORTS                     R16.0
       JREPORT 010                   EXTERNAL USER IDENTITY JOURNAL REPORT
         USER                  EXT                 TRANSACT  PROGRAM    LOCAL       LOCAL
          ID                    ID                   IDX      NAME      DATE        TIME
        ABBWI01              JACK2006                  5  IDMSJDBC  12/22/05  14.19.29.66
        USER ID NOT CAPTURED EXT ID NOT CAPTURED       6  IDMSDDDL  12/22/05  14.19.29.67
        ABBWI01              EXT ID NOT CAPTURED       7  RHDCRUAL  12/22/05  14.19.29.68
        NO USER SIGNON                                 8  RHDCRUAL  12/22/05  14.19.29.69
        ABBWI01              SUSIE666                  9  JAVAPROG  12/22/05  14.19.29.70

       C750009 RECORDS WRITTEN FOR REPORT 10 --      8
```

Note: For more information about running and interpreting journal reports, see the *CA IDMS Reports Guide* and the *CA IDMS Journal Analyzer User Guide*.

Handling Invalid Numeric Data

One of the most useful features of CA IDMS is its ability to access network records using SQL. These network records are often redefined and have multiple formats, causing problems when data in a record occurrence is not in the correct format for the type of the SQL column derived from the network schema record definition. In particular, decimal fields are sometimes redefined as character fields. They can contain spaces, low values, or other data that is not a valid packed or zoned decimal value. This violates the data integrity provided by the CA IDMS SQL Option, which ensures that data stored in an SQL table is valid for the column type.

In this situation, an application like the Online Command Facility (OCF), with direct access to the fetch buffer returned by CA IDMS, can display a special indicator for the value (for example, a string of asterisks). Interfaces like ODBC and JDBC, however, are expected to convert the data to the format requested by the application, and data integrity is assumed.

CA IDMS Server provides an Invalid Decimal option to handle this situation, allowing the client to specify what the ODBC and JDBC drivers should return to the application when invalid data is received from CA IDMS. Options are:

Return Error

(Default) Drivers return an error to the application. The ODBC driver returns SQL_ERROR to the application, which can use the SQLError function to retrieve the associated error message. The JDBC driver returns a SQLException containing the error message. No value is returned in the output buffer provided by the application.

Return Null

The drivers attempt to return NULL for the column value. The ODBC driver sets the length/indicator value to SQL_NULL_DATA. It returns an error if the pointer to the length/ indicator buffer supplied by the application is 0. The JDBC driver returns either 0 or null, as specified by the ResultSet.getXXX method, or true for ResultSet.wasNull. The drivers attempt to return NULL even if the column in the result set is NOT NULL.

Return 0

The driver always returns zero. This can be useful when the application does not provide a length/indicator buffer for a column that is NOT NULL.

Ignore

The value returned to the application is undefined. This option is provided for compatibility with previous versions of the ODBC driver, and is not supported by the JDBC driver.

The ODBC driver prints a message to the log when tracing is enabled, no matter which option is selected.

The Return Null option does not work if the application does not check for a NULL value when the result set column is defined as NOT NULL. By default, an SQL column returned for a network record is treated as NOT NULL because there is no NULL indicator field in the database. Since a result set column that is an expression does allow NULL values, one solution is to enclose the column name in a VALUE scalar function, as in the following example:

```
SELECT VALUE(WARD_TOTAL_0430) FROM EMPDEMO."HOSPITAL-CLAIM"
```

This forces CA IDMS to build a result set that includes a NULL indicator for the WARD-TOTAL-430 field in the HOSPITAL-CLAIM network record, and the drivers report that the column allows NULL values.

It may be convenient to define views on network records that wrap DECIMAL and NUMERIC columns in VALUE functions, at least when the database is known to contain invalid data.

Pseudo-Conversational Processing

CA IDMS Server uses pseudo-conversational processing to minimize resource use on the CA IDMS system. The ODBC and JDBC drivers support pseudo-conversational processing by automatically issuing an internal SUSPEND command, which causes the current task to end, freeing resources on the CA IDMS CV.

By default, pseudo-conversational processing is optimized for interactive applications, in which a user enters a command and then works with the results for a while before entering another command, similar to the way an on-line application would be used. The drivers suspend the connection and end the task after most SQL commands.

You can customize this behavior can be by selecting a "suspend strategy" that specifies a set of suspend options appropriate for a particular type of application. These suspend strategies are:

- Interactive—Intended for use by applications with a user interface, in which database activity and user input are intermixed. The driver suspends the connection when the transaction is committed. This is the default strategy for ODBC and non-pooled JDBC connections.
- Service—Intended for use by JDBC applications that run in an application server that pools connections and allocates them temporarily to units of work that that access the database one or more times without waiting for user input. The driver suspends only when it determines that the connection is idle. This is the default for pooled JDBC connections. It is of limited use for ODBC applications, since the ODBC driver manager does not notify the driver when a connection is returned to the idle pool.

- Batch—Intended for use by applications access the database many times and terminate without waiting for user input. The driver does not use pseudo-conversational processing at all.
- Custom—Intended for compatibility with previous releases. The driver's behavior is determined by the options specified in the Windows registry, or equivalently, in the USS configuration file or Java properties file. These options are documented in the appendices.

The following table shows the detailed suspend options that correspond to each strategy, 0 = disabled, 1 = enabled, X = either:

Option	**Strategy**			
	INT	**SER**	**BAT**	**CUS**
ConnectSuspend	0	1	0	X
CloseCommit	1	1	0	X
CommitSuspend	1	0	0	X
FetchSuspend	0	0	0	X

Configuring Secure Sockets

You may need to install certificates to use SSL with both the ODBC and JDBC drivers. Certificates are used to prove the identity of one or both parties in a secure socket connection. Certificates are stored in a database called a keystore by the Java implementation of SSL. Other implementations may use different terms for the certificate database, for example, mainframe security systems call it a keyring.

The SSL software used within the IDMS ODBC driver does not use a keyring or a keystore per-se, but simply requires you to name both the Client and Server certificate, as well as a common certificate directory where all signing certificates must reside.

A certificate that has been signed by a recognized Certificate Authority (CA) is automatically trusted by the SSL implementation.

Certificates on the Client System

If your server does not have a certificate signed by a Certificate Authority, you must install the server's certificate as a trusted certificate in the client keystore. If your server requires client authentication, you must install the client certificate in the client keystore.

Sample certificates have been provided in the CA IDMS Server 'certificates' sub-directory that you can use for both Server and Client authentication. The certificates are:

- JSRVCERT—Server Certificate
- JCLICERT—Client Certificate

Several different versions (or formats) of these certificates have been included for use with the ODBC driver, JDBC Driver, and for importing on the mainframe. The file extensions for these versions are:

- .PEM—(Privacy Enhanced Mail) Base64 encoded certificates. These certificates are suitable for use with the ODBC Driver when used with the 'IDMS' communicatios protocol.
- .CERTDER.CER—Public-Key Infrastructure (X.509), or PKIX, certificates. These certificates are suitable for use with the JDBC Driver.
- .CER - DER encoded PKCS#12 package certificates. These certificates were used to create the '.PEM'-format certificates, and are suitable for importing onto the keyring on the mainframe.

All certificates contain encoded data and should always be transmitted in BINARY mode.

Certificates on the Server

You must install the server certificate in the server's certificate database (keystore, keyring, or equivalent). If your server requires client authentication and your client does not have a certificate signed by a recognized CA, you must install the client certificate as a trusted certificate in the server's keystore or keyring.

Certificate Administration

You can administer certificates on the client and server by using the tools described in this section.

JDBC Driver

Use the keytool utility supplied with the JDK tools to maintain the Java keystore file for standalone Java applications using the JDBC driver and for the JDBC server. Define system properties on the Java command line to specify the keystore at run time. For more information, see the JDK Tools documentation on java.sun.com. Refer to the vendor's documentation on how to specify the keystore file when running a Java application in an application server such as Websphere or Weblogic.

ODBC Driver

All certificates used by the CA IDMS ODBC driver must be in PEM format. If you wish to use your own certificates (generated on the mainframe for example), then you can use the openssl utility to convert your certificate from PKCS12DER format to PEM format. The openssl utility is provided with the CA IDMS Server installation, and can be found in the CA IDMS Server installation directory. An example of the openssl command to perform this conversion follows:

```
openssl pkcs12 -in C:\temp\PKC12DER.CER -out C:\temp\PKC12DER.PEM
```

For more information about the use of the OpenSSL utility, see the OpenSSL site: http://www.openssl.org/docs/apps/openssl.html

Mainframe

Use your security system's commands to maintain the keyring and configuration policy for AT-TLS. If you chose to use the sample certificates provided with the CA IDMS Server installation, then those certificates need to be uploaded and installed on your mainframe security system. The certificates with the ".CER" extension are suitable for this purpose. All certificates contain encoded data and should always be transmitted in BINARY mode.

Configuring Timeouts

Timeouts can be configured for each component used with CA IDMS Server, including the application server, CAICCI/PC, the ODBC and JDBC drivers, the JDBC server, and CA IDMS. The ODBC and JDBC drivers also support setting timeouts at runtime. Although there seem to be a large number of timeout settings, they can be grouped into three types:

- Reply Timeouts limit how long the client waits for a reply from the server after making a request.
- Idle Timeouts limit how long the server waits for the client to make a request.
- Other Timeouts are used to tune socket connections.

This section describes these timeouts and how they are related. On Windows, timeouts are generally defined in the registry, using the CAICCI/PC and ODBC administrator applications. See the appropriate CA Common Services manual or chapter "Configuring the Client on Windows" for information on using these applications. Detailed descriptions of each setting are included in the appendix "Windows Registry Information." On z/OS, timeouts are defined in the caidms.cfg configuration file. For detailed information on these settings, see the appendix "Configuration File Information." Certain timeouts can also be set in the caidms.properties Java properties file. For information on these settings, see the appendix "Properties File Information."

Reply Timeouts

Reply timeouts are necessary to prevent a client from waiting indefinitely for a reply from the server. When an error occurs that prevents the server from responding, the client receives an error after the timeout expires, and can retry the request or report the error. If a timeout is not set or is set too high, the application thread never regains control, and it may be necessary to terminate the entire application. When connection pooling is in use this can cause unusable connections to remain in the pool.

The optimal length of the reply timeouts depends on the type of application. Short queries and updates can use a relatively short timeout. Queries that require scanning large databases require longer timeouts, especially if they include SORT BY or ORDER BY clauses, which cause CA IDMS to retrieve the entire result set before returning the first row.

Note: Reply timeouts should be longer the further away from the server they are set, to allow the client to handle timeout errors more accurately and efficiently.

For ODBC or the JDBC type 2 driver, a reply timeout can be set globally within CAICCI/PC, for a specific CA IDMS CV in the Server *servername* definition, or at run time. This timeout is used by CAICCI/PC on Windows or the CCI service of ENF on z/OS (referred to as CAICCI/ENF). For JDBC type 3 and type 4 drivers, a reply timeout can be set globally using the IdmsConnectionOptions WaitTimeOut property, or at run time. This timeout is used as the default socket timeout. Reply timeouts can also be set for the JDBC Server and for the CASERVER and IDMSJSRV tasks.

CAICCI/PC

On Windows, you can use the TCP/IP tab of the CAICCI/PC Configuration dialog to set the default Reply Wait timeout for all connections. This value is the timeout for the sockets that are used to communicate with the CCITCP address space on the mainframe.

The default is -1, which causes CAICCI/PC to wait indefinitely. Typical values range from 30 to 300 seconds.

Server servername Definition

You can use the Server *servername* definition to specify the reply timeout used by CAICCI for a specific data source or group of data sources that connect to the same CV. It is supported on Windows and on z/OS.

On Windows, use the CA IDMS Server ODBC Administrator Server tab to set the Wait Timeout. The setting is passed to CAICCI/PC when the connection is established, and overrides the CAICCI/PC setting. Setting the value to 0 causes the CAICCI/PC value to be used.

On z/OS, you can define WaitTimeOut in a Server *servername* section in the *caidms.cfg* configuration file. This value is used with CAICCI/ENF, and is not a socket timeout. The default is 0, which causes CAICCI/ENF to wait indefinitely. Unlike the Windows setting, there is no global default setting.

ODBC Driver

An ODBC application can set the Server *servername* definition timeout for specific connection using the DRIVER, DICT, NODE, and WAIT keywords with the SQLDriverConnect API. This overrides the CAICCI/PC Reply Timeout setting. A connection established this way does not use a Data Source or Server *servername* definition in the registry.

Note: For more information, see the chapter "ODBC Programmer Reference."

JDBC Driver

A JDBC application can set the reply timeout for specific Connection or Statement object using the methods specified by the JDBC API.

The application can use the DriverManager.setLoginTimeout method to set the reply timeout for JDBC Connections established using the DriverManager.

The application can use the DataSource.setLoginTimeout method to set the reply timeout for all connections established using a specific DataSource. Application servers may provide administrative tools to maintain these properties.

The IdmsConnectOptions LoginTimeout or WaitTimeOut properties can be set to define the default system login timeout.

A JDBC application can also use the Statement.setQueryTimeout method to set the reply timeout for all SQL requests made using a specific Statement object. The IdmsConnectOptions QueryTimeout or WaitTimeOut properties can be set to define the default system query timeout.

For the JDBC type 2 driver, setting a reply timeout to a non-zero value using any of these methods overrides the reply wait timeout set for CAICCI/PC on Windows and sets the timeout for CAICCI/ENF on z/OS. For the JDBC type 3 and type 4 drivers, the reply timeout is used as the socket timeout. In this case, a value of zero means that the reply timeout is disabled and the driver waits forever.

Note: For more information, see the chapter "JDBC Programmer Reference."

JDBC Server

When the JDBC driver is used with the JDBC Server, the reply timeout specified using one of the JDBC methods is incremented slightly and used as the timeout value for socket requests from the JDBC driver to the JDBC server.

If the connection was made without a data source defined in the registry or configuration file, the original loginTimeout value is passed to the JDBC server and used as the default for CAICCI/ENF.

If the connection is made using a data source defined in the registry or configuration file for the JDBC Server, the value of the Server *servername* section WaitTimeOut is used as the default for CAICCI/ENF. In this case the loginTimeout is used only for the socket connection from the JDBC driver to the JDBC Server, and should be set at least as high as the WaitTimeOut setting.

The original queryTimeout value is passed to the JDBC server and overrides the default for CAICCI/ENF no matter how it was set.

When an intermediate JDBC Server is used, as might be done to support applets from a web server running on Linux, the reply timeout for socket requests from the intermediate server to the server where the native SQL client runs should be specified on the intermediate server.

On Windows, use the CA IDMS Server ODBC Administrator Proxy tab to set the Reply Timeout for the JDBC Server. On z/OS and Linux, you can define ReplyTimeOut in the Proxy section in the caidms.cfg configuration file. You can define this option in the caidms.properties file on all platforms. The intermediate server reply timeout is used only for the socket connection between the two JDBC servers, and should be longer than the reply timeout set in the Server *servername* section on the destination JDBC Server.

CASERVER and IDMSJSRV Task

In CA IDMS, you can define the INACTIVE INTERVAL on the CASERVER and IDMSJSRV task definitions to specify how long the task waits for a resource (this can be modified dynamically using the DCMT VARY TASK STALL INTERVAL command). This setting defines how long the task waits for a specific resource, such as a database record. This value often defaults to a SYSTEM setting and should be shorter than any client reply timeout setting. There is no timeout that can be set to specify how long the task should wait for the overall request to be satisfied (because the task is busy satisfying the request and is not waiting).

Idle Timeouts

Idle timeouts are necessary to prevent a server from waiting indefinitely for a request from a client, which unnecessarily ties up resources. When the timeout is exceeded, the server assumes that the client no longer needs the connection, possibly because of an error, and frees connection resources in an orderly fashion. This includes closing any connections to a server closer to the CV. Properly configured idle timeouts are particularly important for efficient connection pooling. The optimal length of the idle timeouts depends on the type of application or application server.

Note: Idle timeouts should generally be shorter the further away from the server they are set, to allow the servers to handle timeout errors more accurately and efficiently. Depending on how heavily loaded the servers are, the differences may be relatively large.

Idle timeouts can be specified for the application server's connection pool, the JDBC driver, the JDBC Server, and the CASERVER and IDMSJSRV tasks.

Application Servers

Application servers typically provide administrative tools to maintain idle timeouts for connections in their pool. When the timeout is exceeded, the application server closes the connection and removes it from the pool.

Application servers might support global timeouts in an implementation specific way or might honor the timeout properties specified in a ConnectionPoolDataSource object. The IdmsConnectionPoolDataSource class supports the maxIdleTime and propertyCycle properties as recommended in the JDBC 4.0 specification. The effective idle time for an application server that uses these or equivalent properties could be as large as the sum of the two intervals. The Type 4 JDBC driver uses these properties to adjust the RESOURCE TIMEOUT INTERVAL of the task automatically. You may need to specify the equivalent idle time property in both the application server's ConnectionPoolDataSource implementation and as a custom property for the IdmsConnectionPoolDataSource that it references.

Note: Not all application servers use these properties.

The idle timeout set for a pooled connection should be shorter than the idle timeouts set closer to the CV, possibly by as much as 300 to 600 seconds.

JDBC Server

You can specify the interval that the JDBC server waits for a request from the JDBC driver. When this interval is exceeded, the JDBC server assumes that an error has occurred, releases the SQL session with CA IDMS, and closes the socket connection with the JDBC driver. If the application attempts to use the connection after this, the JDBC driver returns a SQLException indicating that a communications error has occurred.

On Windows, you can use the CA IDMS Server ODBC Administrator Proxy tab to set the Wait Timeout for the JDBC Server. On z/OS and Linux, you can define WaitTimeOut in the Proxy section in the caidms.cfg configuration file. You can define this option in the caidms.properties file on all platforms. The idle timeout for the JDBC server should be longer than the timeout set for the application server and shorter than the timeouts set for the CASERVER or IDMSJSRV task.

CASERVER and IDMSJSRV Task

You can specify two idle timeouts for the CASERVER and IDMSJSRV tasks:

RESOURCE TIMEOUT INTERVAL

Specifies how long an idle connection persists before terminating the connection if no task is active.

EXTERNAL WAIT

Specifies how long an idle connection persists before terminating the task and connection when a task is active.

When the application server's connection pool implementation recognizes the IdmsConnectionPoolDataSource or the IdmsXADataSource, the RESOURCE TIMEOUT INTERVAL should be longer than the maxIdleTime or equivalent application server property. When the application server uses the DriverManager of IdmsDataSource to get a connection, both of these should be longer than timeouts set in the application server and JDBC server.

Other Timeouts

You can use other timeouts to tune the way CA IDMS Server uses TCP/IP sockets.

CAICCI/PC

On Windows, you can use the CAICCI/PC Properties TCP/IP tab to set two additional timeout values.

Ready to Receive

Specifies how long CAICCI/PC waits for subsequent TCP/IP packets after receiving the initial reply packet. Unless the network is extremely congested these usually arrive very quickly, and this timeout is seldom exceeded. The default value is '-1', which causes CAICCI/PC to wait indefinitely. This rarely causes problems, but a value of 60 seconds is reasonable to ensure that no reply can cause the application thread to hang.

Ready to Send

Specifies how long CAICCI/PC waits to send data on a socket. Since sockets are full duplex, there is almost never any wait at all. The default setting of 60 seconds is generally acceptable.

These timeouts are not needed when using the Type 4 JDBC driver.

JDBC Server

When the JDBC server does socket I/O, the thread is blocked and cannot check for events such as server shutdown. You can specify the internal timeout interval that the JDBC server uses when reading from a socket or waiting for a connection request. When this interval is exceeded, the JDBC server checks for shutdown and other events, then resumes reading from the socket unless an idle or reply timeout has expired.

This timeout is set by as the SocketTimeOut value in the Proxy section of the registry on Windows and the caidms.cfg configuration file on z/OS. It can also be specified in the caidms.properties file on all platforms. The default setting of 60 seconds is generally acceptable.

Chapter 4: Installing the Client on Windows

Instructions are provided for the installation of CA IDMS Server on Windows for use with the ODBC driver, the Type 2 JDBC driver, and the JDBC server. Although the complete CA IDMS Server installation is not required for applications that use only the Type 3 or Type 4 JDBC drivers, installing the product is recommended for convenient access to the user documentation, the Javadoc that describes the JDBC implementation, and the Java Command Facility (JCF) demo program.

Note: For more information about procedures for using these driver types without the full installation, see the chapter "Using the Java Client."

This section contains the following topics:

Preparing to Install CA IDMS Server

The following must be installed before you install CA IDMS Server:

- CAICCI/PC, required to use the ODBC driver with CA IDMS releases prior to r17 and the JDBC driver with CA IDMS releases prior to r16 SP2.
- ODBC Driver Manager and Administrator, required to use the ODBC driver or Type 2 JDBC driver
- Java Runtime Environment (JRE) 1.6 or later, required to use the JDBC driver

Obtain the following information from your CA IDMS system administrator:

- The name of the dictionary containing the definitions of the tables you access.
- For the CCI protocol you need the node name of the system on which the dictionary resides and the host name and port for CCITCP (the default port is 1202).
- For the IDMS protocol you need the host name and port for the CA IDMS system
- The task code defined for CA IDMS Server. The default for the CCI protocol is CASERVER and the default for the IDMS protocol is IDMSJSRV.

You must use a user ID with administrative privileges when installing CA IDMS Server and CAICCI/PC.

For your convenience, the complete CAICCI/PC installation is included in the CCI directory on the CA IDMS Server CD. It can be installed from the CA IDMS Server Product Explorer that starts automatically when the CD is inserted, or after double-clicking Setup.exe on the CD's setup directory.

Uninstalling Previous Versions

CA IDMS Server r17 can be installed on the same machine without uninstalling a prior version and the installer does not automatically uninstall the previous release. Existing data sources continue to use the version of CA IDMS Server used to create them. You should uninstall the prior release and remove its data sources after you have tested your applications with CA IDMS Server r17.

You can use uninst40.bat or uninst41.bat to uninstall CA IDMS Server r4 or r4.1. You can find these files in the product directory, typically, \CA_APPSW\CAID. You might need to edit the file before running it.

You can use the Windows Control Panel Add/Remove Programs applet to uninstall CA IDMS Server r4.2 or later. In previous versions, the default location for the CA IDMS log file was the CA IDMS Server directory. Log files are not deleted by the uninstall process and must be deleted manually.

Installing CA IDMS Server on Windows

To install CA IDMS Server on Windows

1. Sign on with a user ID that has administrative authority.
2. To allow shared components to be updated properly, exit all Windows applications, including Microsoft Office tool bars, before beginning the installation of CA IDMS Server.
3. Insert the CA IDMS Server CD into your CD-ROM drive. The CA Product Explorer begins automatically. If it does not, right-click on the CD icon in the My Computer window and select Auto Start.
4. Select the Install CA IDMS Server option.
5. Before copying the files from the installation disk onto your system, the Installer displays the readme.txt file, containing information unavailable when this document was prepared.

6. Choose one of the following installation options:

   - Select Typical to install both the ODBC driver and the JDBC driver
   - Select Compact to install the ODBC driver only
   - Select Custom to choose which driver to install

   Note: The ODBC driver is always selected, because it installs components also used by the JDBC driver.

7. User-defined settings such as data sources and options which were created for an earlier release of CA IDMS Server are not compatible with r17. The CA IDMS Data Source Utility provides a way of converting these settings to make them usable. The utility is started automatically by the installer. Several conversion options are available:

   - No Conversion means that no data source conversion takes place. This is the default. Data sources for r17 must then be entered manually by the user after installation is complete, using the CA IDMS ODBC Administrator.
   - Convert in Place changes existing data sources to use the r17 driver. These data sources no longer work with the prior release of CA IDMS Server.
   - Create Copies with Prefix copies the data sources to new data sources that use the r17 driver, named with a prefix that you can specify.. The existing data sources are not changed.
   - Global Defaults copies options settings from the prior release.
   - JDBC Server copies settings for the JDBC server from the prior release.

Note: For more information about defining data sources, see the chapter "Configuring the Client on Windows."

Chapter 5: Configuring the Client on Windows

An ODBC data source specifies the information needed to connect to a database from an ODBC application. The Type 2 JDBC driver can also use ODBC data source definitions to access CA IDMS databases by specifying the data source name as part of the JDBC URL or as a property in an IdmsDataSource object. The JDBC driver does *not* use the ODBC driver at runtime.

This section contains the following topics:

Configuring Windows Applications

CA IDMS Server and CAICCI/PC attempt to write messages to a log file when an error occurs. The user ID used to run applications must have write access to the directories where these log files are written. This includes applications that run as Windows services, such as the Microsoft Internet Information Server (IIS).

Configuring CAICCI/PC

When you use the ODBC and Type 2 JDBC drivers you must specify the host name or TCP/IP address. You must specify the host for CAICCI to use the ODBC and Type 2 JDBC drivers. You can also specify the port, socket timeouts, and enable tracing for CAICCI.

Use the CAICCI Configurator to configure CAICCI with settings that apply to all data sources or you can use the CA IDMS Server ODBC Administrator to configure CAICCI for a specific data source. You must have administrative authority to run the CAICCI Configurator which is started from the CAICCI Menu. At run time the user ID used to run applications must have write access to the directory specified for the trace file. For detailed information about CAICCI error messages as they relate to CA IDMS Server and troubleshooting tips, see PIB QI43460 on supportconnect.ca.com.

SSL support for ODBC and JDBC type 2 drivers is provided by CAICCI and must be configured using the CAICCI Configurator.

Note: For detailed information about setting the CAICCI/PC options, see the CA Common Services documentation.

Configuring CA IDMS Server

Use the CA IDMS Server ODBC Administrator to define data sources. Start the ODBC Administrator from the CA IDMS menu. The CA IDMS Server ODBC Administrator provides a context-sensitive online Help facility. Click Help on any tab to obtain Help about that dialog.

You can also use the ODBC Administrator to set default data source options, language options, and configure the JDBC server. If you are using User Account Control (UAC) on Windows Vista, you must have administrative authority to set these options. They must be grayed out if you are not permitted to set them.

Defining Data Sources

You can define new data sources as User or System data sources. User data sources are available only to the user who defined them. System data sources are available to all users and services of the machine. Typically, data sources meant to be used by Windows services must be defined as System data sources. You must have administrative authority to define a system data source.

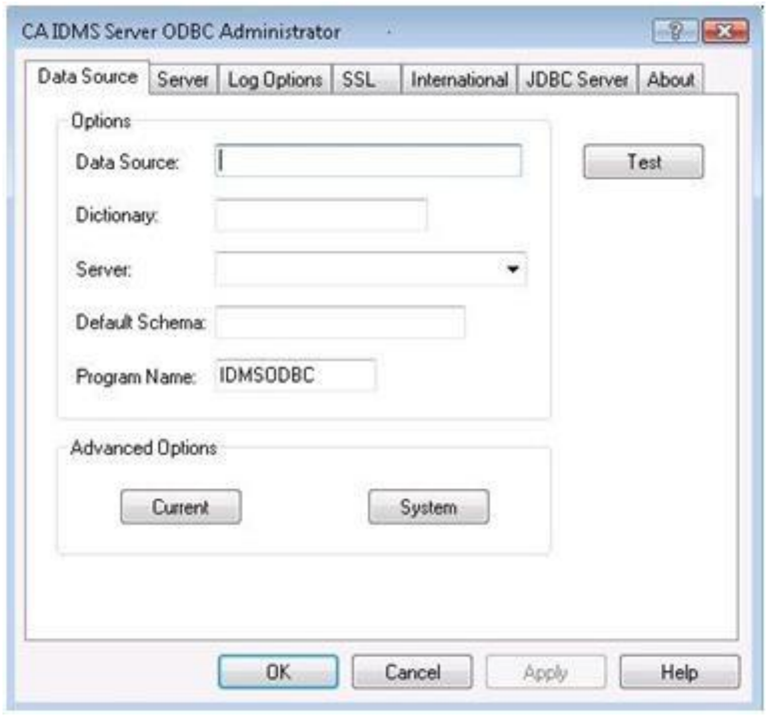
Adding a New Data Source

When adding a new data source, you must specify the type of data source, and include the name of the dictionary where the SQL tables are defined as well as the node name of the CV containing the dictionary. Use the ODBC Data Source Administrator to begin the process of adding a new data source.

To access the ODBC Data Source Administrator, select Start, Programs, CA, CA IDMS Server, ODBC Administrator. Note that on 64-bit Windows, there might be shortcuts to both 32-bit and 64-bit versions of the ODBC Administrator.

The ODBC Data Source Administrator dialog lists the names of each defined data source, followed by the database driver in parentheses. If no Data Source Name (DSN) is listed, click the User DSN or System DSN tab and click Add to invoke the Create New Data Source dialog.

The Create New Data Source dialog lists all installed drivers. Select CA-IDMS and click Finish. The CA IDMS Server ODBC Administrator dialog appears:



Program Name: IDMSODBC

The Data Source tab of the CA IDMS Server ODBC Administrator dialog lets you define a new data source or modify the dictionary or server of an existing data source. You must supply the following essential information to define a new data source or modify.

Options

The ODBC and Type 2 JDBC drivers use the following information:

Data Source

Specifies the data source name. To add a new data source, enter a character string of up to 32 characters in this field. Use a combination of letters, numbers, spaces, or special characters. When modifying an existing data source, the name cannot be changed.

Dictionary

(Optional) 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. The default is the first eight characters of the Data Source.

Server

Specifies a user-defined name which represents the CA IDMS system you want to access. Enter a new 1-to-32 character name or select an existing name from the pull-down list. This field is required. 'Server' is a logical construct, which contains various options needed to access your IDMS CV.

Default Schema

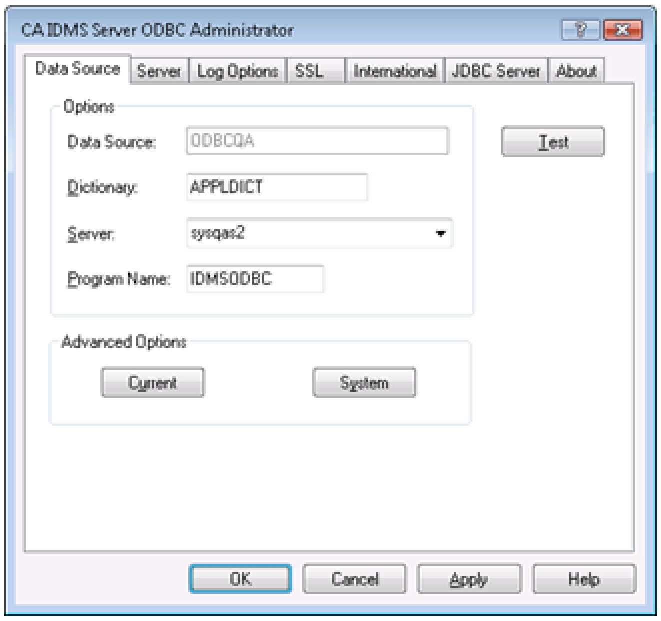
Specifies the name of the default SQL Schema. This is an optional 1-to-18 character field. When specified, this field is used as the schema qualifier for all SQL table references that do not contain an explicit schema qualifier. The default is blank (unspecified).

Program Name

Specifies a program name that are associated with the task on the CA IDMS system. This name is recorded in the CA IDMS journal and allows you to correlate a database update to a specific application. The default Program Name is IDMSODBC.

Advanced Options

Advanced Data Source options can be specified at both the Data Source level (by clicking the "Current" button) as well as at the System level (by clicking the "System" button). Options specified at the Data Source level override options specified at the System level. Details on these options are covered in the sections "Setting Advanced Data Source Options" and "Setting System Default Data Source Options."



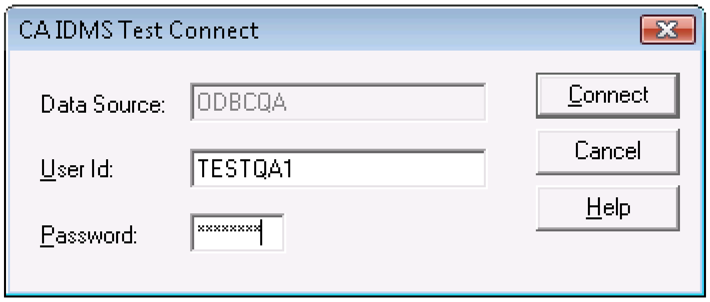
Data Source: ODBCQA
Dictionary: APPLDICT
Server: sysqas2
Program Name: IDMSODBC

Saving the Data Source Definition

After you have created your data source definition, click OK to save the definition to the registry, close the CA IDMS Server ODBC Administrator dialog, and return to the Data Sources dialog. To save the definition without closing the CA IDMS Server ODBC Administrator, click Apply. Click Cancel to return to the Data Sources dialog without saving the definition.

Testing the Data Source Definition

You can verify that your data source is defined correctly, and that CA IDMS Server is installed correctly, using the Test Connect application. Click Test to invoke the CA IDMS Test Connect dialog:



Data Source: ODBCQA
User Id: TESTQA1
Password: ********

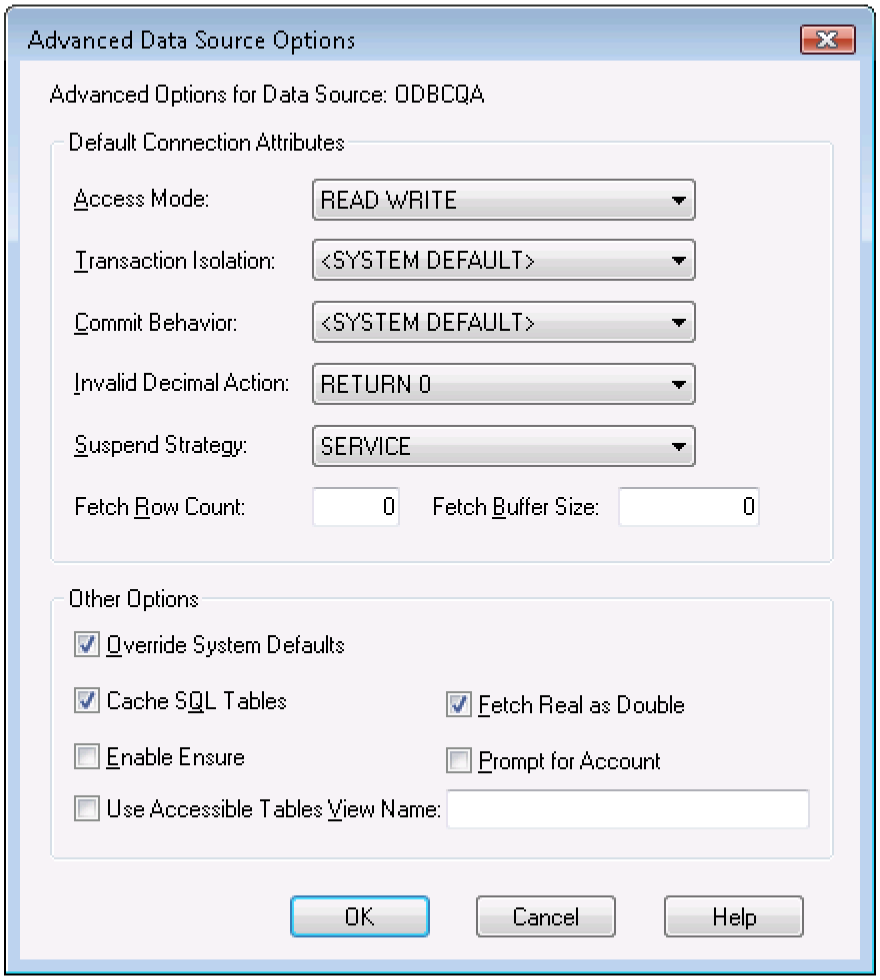
The DSN identified in the ODBC Administrator appears in the Data Source field, and cannot be changed. In the User ID field, enter a valid user ID for the CA IDMS system. If required for the system, supply a password in the Password field and click Connect. The test program connects to the data source using the ODBC driver.

Editing the Data Source Definition

Once you have saved your data source definition, you might find it necessary to edit or update the information. To edit a data source definition, access the Microsoft ODBC Administrator and select the data source to be edited. Click Configure to invoke the CA IDMS Server ODBC Administrator dialog. Edit the information and click Apply or OK.

Setting Advanced Data Source Options

The Advanced Data Source Options panel is reached by clicking on the "Current" button from the Data Source tab. The Data Source name appears at the top of the dialog.



Advanced Options for Data Source: ODBCQA

Default Connection Attributes

Access Mode: READ WRITE

Transaction Isolation: <SYSTEM DEFAULT>

Commit Behavior: <SYSTEM DEFAULT>

Invalid Decimal Action: RETURN 0

Suspend Strategy: SERVICE

Fetch Row Count: 0 Fetch Buffer Size: 0

Other Options

[✓] Override System Defaults

[✓] Cache SQL Tables [✓] Fetch Real as Double

[ ] Enable Ensure [ ] Prompt for Account

[ ] Use Accessible Tables View Name:

Options specified at the Data Source level override options specified at the System level (see the section entitled "Setting System Default Data Source Options"). All drop-down boxes specifying the value "<SYSTEM DEFAULT>" default to the setting specified at the System level. Likewise, numeric fields having a value of 0 also default to the System level setting. To enable the setting of the checkbox-style options, check the "Override System Defaults" box in the "Other Options" section. A detailed description of each option follows:

Default Connection Attributes

Access Mode

Specifies the default setting for the access mode connection attribute, which is defined by both ODBC and JDBC. An application can use the ODBC or JDBC interfaces to set and query this attribute at run-time. Options are:

- **READ WRITE—**Allows applications to read and update the database.
- **READ ONLY—**Allows applications to read the database. This option is recommended, unless you intend to update data in the CA IDMS database. This option can be set and queried by the application using the SQLSetConnectOption and SQLGetConnectOption functions.
- **<SYSTEM DEFAULT>—**Indicates that the System-level setting for this option should be used.

If no specification is made at both the Data Source and the System level, then a runtime default of "READ WRITE" is used.

Transaction Isolation

Specifies the degree to which your transactions impact, and are impacted by, other users accessing the same data for the ODBC driver. Choose one of the following:

- **READ COMMITTED—**Prevents access to data updated by another user, before it has been committed. This corresponds to the CURSOR STABILITY option of the SQL SET TRANSACTION statement, and is the default setting.
- **READ UNCOMMITTED—**Permits only retrieval operations to be executed by the user; update requests are rejected. This option can only be selected in conjunction with a Read Only Access Mode, and corresponds to the TRANSIENT READ option on the SET TRANSACTION statement.
- **<SYSTEM DEFAULT>—**Indicates that the System-level setting for this option should be used.

If no specification is made at both the Data Source and the System level, then a runtime default of "READ COMMITTED" is used.

This option can be set and queried by the application using the SQLSetConnectOption and SQLGetConnectOption functions.

Note: For more information about the SET TRANSACTION statement, see the *CA IDMS Database SQL Option Reference Guide*.

Commit Behavior

Specifies the way in which COMMIT operations affect cursors in the ODBC and JDBC drivers. Choose one of the following options:

- **CLOSE AND DELETE CURSORS—**Forces the application to prepare and execute the next statement.

- **CLOSE CURSORS—**Allows applications to execute a statement without calling prepare again. This is the default setting.
- **PRESERVE CURSORS—**Maintains cursors in the same position as before the Commit operation, allowing applications to execute or fetch without preparing the statement again.
- **<SYSTEM DEFAULT>—**Indicates that the System-level setting for this option should be used.

If no specification is made at either the Data Source or the System level, then a runtime default of "CLOSE CURSORS" is used.

This is a CA IDMS Server extension allowing you to optimize ODBC usage by different applications. An ODBC application can use the SQLGetInfo function to query this setting. A JDBC application can use the GetResultSetHoldability method.

Invalid Decimal Action

Specifies how the CA IDMS ODBC and JDBC drivers handle invalid packed or zoned decimal data in a result set column. This option is useful when accessing network databases, where records may be redefined so that decimal fields can contain non-numeric data. Options are:

- **RETURN ERROR—**The drivers return an error to the application. The ODBC driver returns SQL_ERROR to the application, which can use the SQLError function to retrieve the associated error message. The JDBC driver throws a SQLException containing the error message. No value is returned in the output buffer provided by the application.
- **RETURN NULL—**The drivers attempt to return NULL for the column value. The ODBC driver does this by setting the length/indicator value to SQL_NULL_DATA. It returns an error if the pointer to the length/value indicator supplied by the application is 0. The JDBC driver returns either 0 or null as specified by the ResultSet getter method, and returns true for the ResultSet wasNull method. Note that the drivers attempt to return NULL even if the column in the result set is defined as NOT NULL.
- **RETURN 0—**The driver always returns 0. This can be useful when the application does not provide the length/indicator value for a column that is NOT NULL.
- **IGNORE—**The value returned to the application is undefined. This option is provided for compatibility with previous versions of the ODBC driver. It is not supported by the JDBC driver.
- **<SYSTEM DEFAULT>—**Indicates that the System-level setting for this option should be used.

If not specified for a data source or as the system default, the default is RETURN ERROR. The ODBC driver prints a warning message to the log when tracing is enabled, no matter which option is selected.

Suspend Strategy

Specifies how the CA IDMS ODBC and JDBC drivers use SQL SUSPEND and COMMIT requests to optimize task resource usage on the CA IDMS system. These strategies replace detailed suspend and commit options that were set in the Windows registry directly for prior releases. The strategies are:

- **INTERACTIVE—**Select this setting when the Data Source is to be used with an interactive ("fat client") application, such as Visual Express or Visual DBA. The drivers SUSPEND after each COMMIT.
- **SERVICE—**Select this setting when the Data Source is to be used as a service, such as an applications that uses a pooled connection in an application server (WebSphere, Weblogic, etc). The drivers SUSPEND when the connection is returned to the idle pool.
- **BATCH—**Select this setting when the Data Source is to be used by a batch-type application, where pseudo-conversational processing is not used. The drivers do not SUSPEND at all.
- **CUSTOM—**Select this setting when you need to specify detailed suspend and commit options in the registry. This setting should only be used by advanced users, or under the direction of CA Technical Support, and requires manual editing of the Windows Registry.
- **<SYSTEM DEFAULT>—**Indicates that the System-level setting for this option should be used.

If no specification is made at either the Data Source or the System level, then a runtime default of "CUSTOM" is used. Note that if no detailed options have been specified this is equivalent to "INTERACTIVE." Refer to Chapter 3, "Setting Up Your CA IDMS System", and the "Windows Registry Information" Appendix for information about using the detailed options.

Fetch Row Count

Specifies the number of database rows to be fetched in a single database request. CA IDMS supports a BULK FETCH feature than can improve performance by fetching multiple rows with a single database request. This option specifies the default value for the number of rows that the driver attempts to fetch. An application can use ODBC and JDBC interfaces to set and query this value at run-time. Valid values are 0 to 30000. The drivers use a smaller value if the value specified would require a buffer larger than the Fetch Buffer Size.

When the value is 0 for a data source, the drivers use the system default, if any. If the system default is also 0, the drivers compute the number of rows that fit in the fetch buffer.

The default is usually appropriate for most applications.

Fetch Buffer Size

Specifies the maximum size of the fetch buffer. The drivers adjust the number of rows to fit in the buffer if necessary. Valid values are 0 to 1048576, although the maximum when using CCI is 30000.

When the value is 0 for a data source, the drivers use the system default, if any. When the system default is also 0 the drivers use 30000 for the CCI communications protocol and 64000 for the IDMS communications protocol.

The default is usually appropriate for most applications.

Other Options

Override System Defaults

If enabled, you can override all of the flag-type settings within the "Other Options" section of the Data Source Options panel. This allows you to establish Data-Source-specific settings for these options.

Cache SQL Tables

Specifies that the ODBC driver caches table lists returned by the SQLTables function. This option improves performance by reducing the amount of time it takes to retrieve a list of tables, but does not always provide the most current view of existing tables. When selected, the ODBC driver uses the cached result to process repeated SQLTables requests. The ODBC driver flushes the cache whenever you turn off this option, change the request parameters, change the name of the Accessible Tables view, or disconnect from a session.

Enable Ensure

Enables the ENSURE parameter of the ODBC SQLStatistics function.

The ENSURE parameter of the SQLStatistics function call usually causes the ODBC driver to issue an UPDATE STATISTICS command to CA IDMS SQL against the named table. For large tables, this can cause deadlocks or communication timeout errors. The default, disabling the Ensure option, is recommended unless a specific application requires otherwise.

Fetch Real as Double

Forces the ODBC driver to return single precision floating point numbers as double precision to avoid the rounding that can occur when numbers are passed from the mainframe to the PC.

Note: Some loss of precision is unavoidable when converting between the floating point formats, because different numbers of bits are used to encode the exponent and mantissa.

Prompt for Account

Causes the SQLDriverConnect function to display a dialog if the optional Account parameter is not passed on the connection string.

Use Accessible Tables View Name

When enabled, you can enter the name of a view to use for the ODBC SQLTables function and JDBC getTables method, instead of using the catalog tables directly. Use this field to specify the default view, SYSCA.ACCESSIBLE_TABLES, or define a different view in the catalog and enter it in this field.

When a catalog contains a large number of table definitions, performance can be improved by specifying a view name, to create a tailored view of the tables of interest to the end user. For example, the SYSCA.ACCESSIBLE_TABLES view returns only those tables to which the user has Select authority. You can also limit tables based on schema or authorization. In addition, this feature is useful when security requirements do not allow direct access to the catalog tables.

If you specify a different name, be sure that it contains at least the same columns as SYSCA.ACCESSIBLE_TABLES, although it can contain additional columns.

The view definition must include the following columns:

- SCHEMA (CHAR(18)
- TABLE (CHAR(18))
- TYPE (CHAR(1))

Click Apply or OK to save changes to the defaults in the registry. Click Cancel to close the dialog without saving any new changes.

Note: For more information about the SYSCA.ACCESSIBLE_TABLES view, see the *CA IDMS Database SQL Option Reference Guide*.

Performance Considerations for ODBC Options

The following ODBC options can affect the performance of the ODBC driver:

Cache SQL Tables

Reduces the time it takes to retrieve a list of tables, but does not always provide the most current view of existing tables.

Enable Ensure

Prevents the SQLStatistics function from issuing commands to update table statistics when disabled.

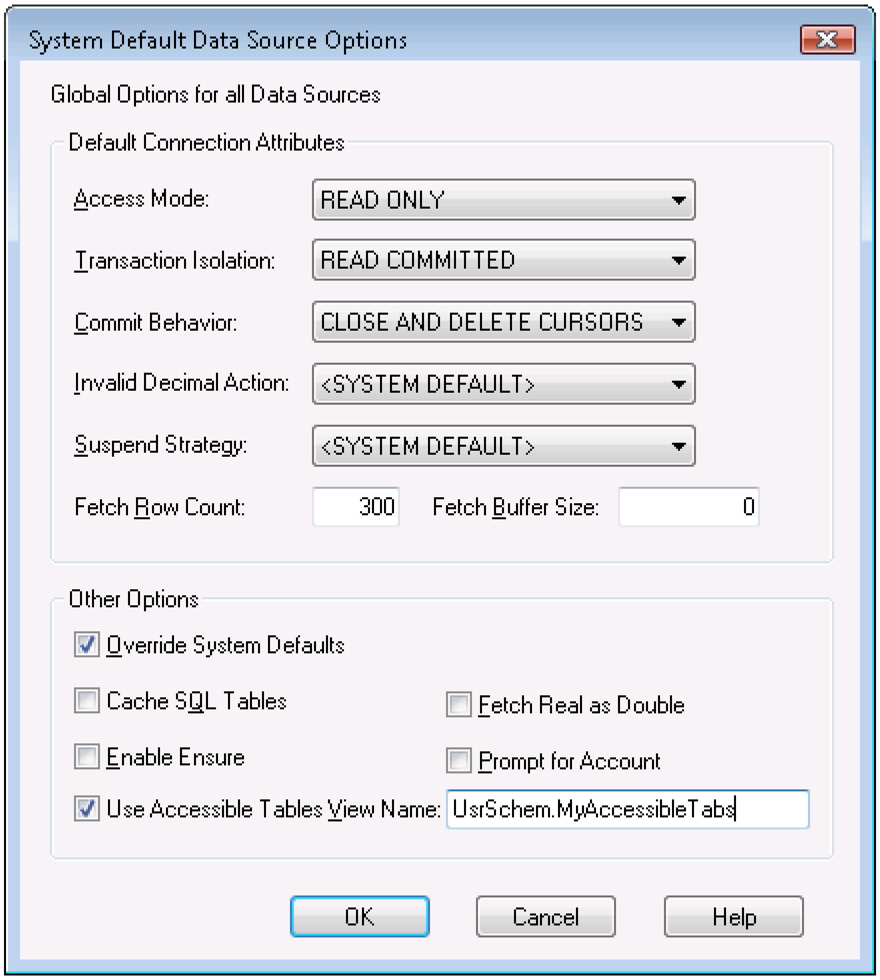
Use Accessible Tables View

Specifies the name of a view, so that only a list of the tables of interest to the end user is returned.

Note: For more information about the various ODBC functions mentioned in the previous descriptions, see Microsoft's documentation for ODBC software.

Setting System Default Data Source Options

The System Default Data Source Options panel is reached by clicking on the "System" button from the Data Source tab. Changes to these options apply to all data sources on the Windows system. For this reason, if you are using UAC on Windows Vista, you must have administrative authority to set these options.


System Default Data Source Options

Global Options for all Data Sources

Default Connection Attributes

Access Mode: READ ONLY

Transaction Isolation: READ COMMITTED

Commit Behavior: CLOSE AND DELETE CURSORS

Invalid Decimal Action: <SYSTEM DEFAULT>

Suspend Strategy: <SYSTEM DEFAULT>

Fetch Row Count: 300 Fetch Buffer Size: 0

Other Options

[✓] Override System Defaults

[ ] Cache SQL Tables [ ] Fetch Real as Double

[ ] Enable Ensure [ ] Prompt for Account

[✓] Use Accessible Tables View Name: UsrSchem.MyAccessibleTabs

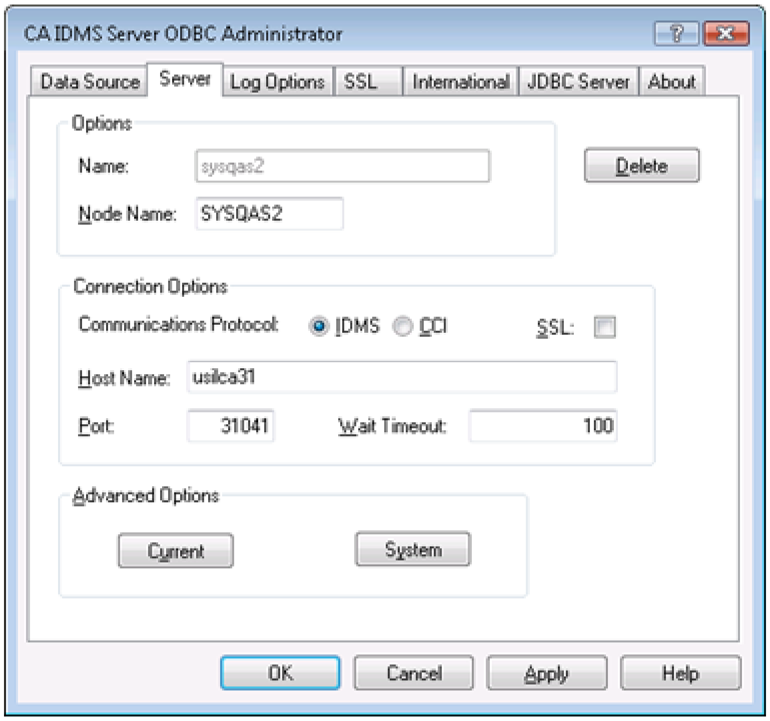
The fields on the System Default Data Source Options panel are identical to those on the Advanced Data Source Options panel. For a description of each option, see the section entitled "Setting Advanced Data Source Options" earlier in this chapter.

Setting Up a Server

Use the Server tab to specify additional information needed to connect to a CA IDMS system. Depending on the Communications Protocol selected, you may override certain default CAICCI connection options on this tab. The Type 3 and Type 4 JDBC drivers do not use the Server definition. A Server definition can be shared by multiple data sources. Like data sources, there are two types of servers, system and user.

A system server can be used by any data source. It is created when a new name is entered into the Server field of the Data Source tab for a system data source. If you are using UAC on Windows Vista, you must have administrative authority to define a system server.

A user server can only be used by the user's data sources. It is created when adding or editing a user data source by typing a new name in the Server field of the Data Source tab source or modifying an existing system server. When a user server is copied from an existing system server, the system server remains unchanged.



Name: sysqas2

Node Name: SYSQAS2

Communications Protocol: [✓ IDMS] CCI

SSL: ☐

Host Name: usilca31

Port: 31041

Wait Timeout: 100

Options

The following fields are present under the Options section:

Name

Displays the selected Server name, from the Data Source tab. The name cannot be changed here.

Node Name

Specifies the Node Name of the system containing the tables you want to access. This is the System ID specified in the system generation parameters. This field is optional, and defaults to the first eight characters of the server name, which must be in upper case, if nothing is specified.

Delete Button

Deletes the server definition and close the dialog.

Connection Options

Communications Protocol

Specifies how the CA IDMS ODBC Driver (or Type 2 JDBC Driver) communicates with CA IDMS. Options are:

- IDMS—The drivers use the CA IDMS TCP/IP feature to communicate directly with the CA IDMS system. CA IDMS r17, or later, is required.
- CCI—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.

SSL

Enables secure communications between the ODBC and Type 2 JDBC drivers and CA IDMS when using the IDMS communications protocol. Use the SSL tab to configure SSL options.

You can enable SSL for the Type 3 and Type 4 JDBC drivers using a special format of the URL or a IdmsDataSource object, and configure it using utilities supplied with the Java Run-time Environment.

You must use your mainframe security system to configure AT-TLS (Application Transparent Transport Level Security) to enable and configure SSL on z/OS when communicating directly with CA IDMS. SSL is not currently supported when communicating with z/VSE backend systems.

You can enable and configure SSL for the CCI communications protocol with the CAICCI Configuration application on Windows. You must also configure the CCITCP address space on z/OS to use SSL.

Host Name

Specifies the DNS name or TCP/IP address of the CCI server or the CA IDMS system. The Communication Protocol setting determines how the Host Name is used.

When used with CCI, this value overrides the default CCI server name specified using the CCI Configurator for this ODBC server only, and allows concurrent access to multiple CCI servers.

Port

Specifies the TCP/IP port of the CCI server or the CA IDMS Listener The Communication Protocol setting determines how the Host Name is used.

When used with CCI, this overrides the default CCI server port specified using the CCI Configurator for this ODBC server only. Enter 0 to use the default value set by CCI, which is normally 1202.

Wait Timeout

Specifies the number of seconds to wait for a reply from the server. This setting overrides the Reply Wait Timeout specified using the CAICCI/PC Properties dialog *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. Choose one of the following options:

- Enter 0 to use the default value set by CAICCI/PC
- Enter –1 to specify an indefinite wait (this is interpreted as the largest positive integer)
- Enter a specific time, in seconds

Advanced Options

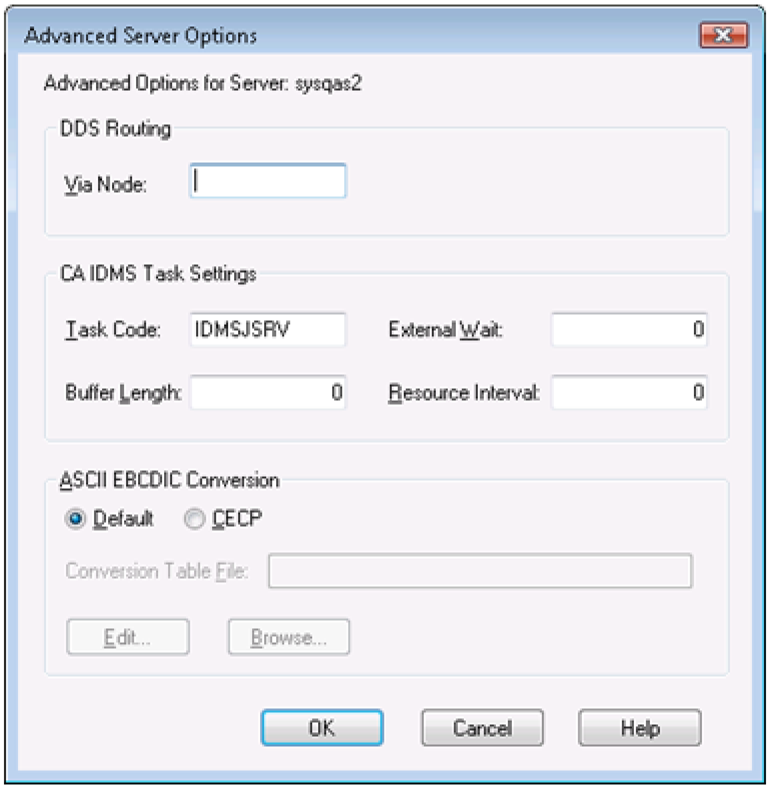
Advanced Server options can be specified at both the Server level (by clicking the "Current" button) as well as at the System level (by clicking the "System" button). Options specified at the Server level override options specified at the System level. Details on these options are covered in the sections entitled "Setting Advanced Server Options" and "Setting System Default Server Options."

Deleting a Server

Click the Delete key on the Server tab. When the "Are you sure…?" confirmation dialog box appears, click Yes.

Setting Advanced Server Options

The Advanced Server Options panel is reached by clicking on the "Current" button from the Server tab. The Server name appears at the top of the dialog.



Options specified at the Server level override options specified at the System level (see the section entitled "Setting System Default Server Options"). All numeric fields having a value of 0 default to the System level setting. A detailed description of each option follows:

DDS Routing

Via Node

Used for DDS node hopping within CA IDMS. This option identifies the DDS node that is used when establishing a connection with the IDMS CV.

This is an optional parameter. When specified, the Via Node must be defined in the RESOURCE name table on the CA IDMS system. Use this option when the destination CV does not directly communicate with CCI.

CA IDMS Task Settings

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.

Buffer Length

Specifies the size of the buffer used by the CA IDMS Server listener for TCPIP send and receive requests. This value overrides the BUFFLEN value specified for the CA IDMS Server listener using the PTERM system generation statement.

This is optional. When set to 0, the System Default for all servers value is used, if any.

External Wait

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 about TASK and PTERM System Generation statements see the CA IDMS Server System Generation guide.

Resource Interval

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.

This is optional. When set to 0, the System Default for all servers value is used, if any.

ASCII-EBCDIC Conversion

By default, CA IDMS Server translates between ASCII and EBCDIC using US English code pages. You can change this behavior and use conversion tables for other languages. Country Extended Code Pages or CECP, can be used to convert single byte character sets. CECP conversion table names can be specified for either your entire system or for a specific Server.

Check the CECP button to specify a CECP table for your Server. This enables the “Conversion Table File” option, allowing you to specify a CECP conversion file.

Edit

Brings up the CA IDMS Code Page Editor, where you can modify the conversion tables to meet your specific requirements.

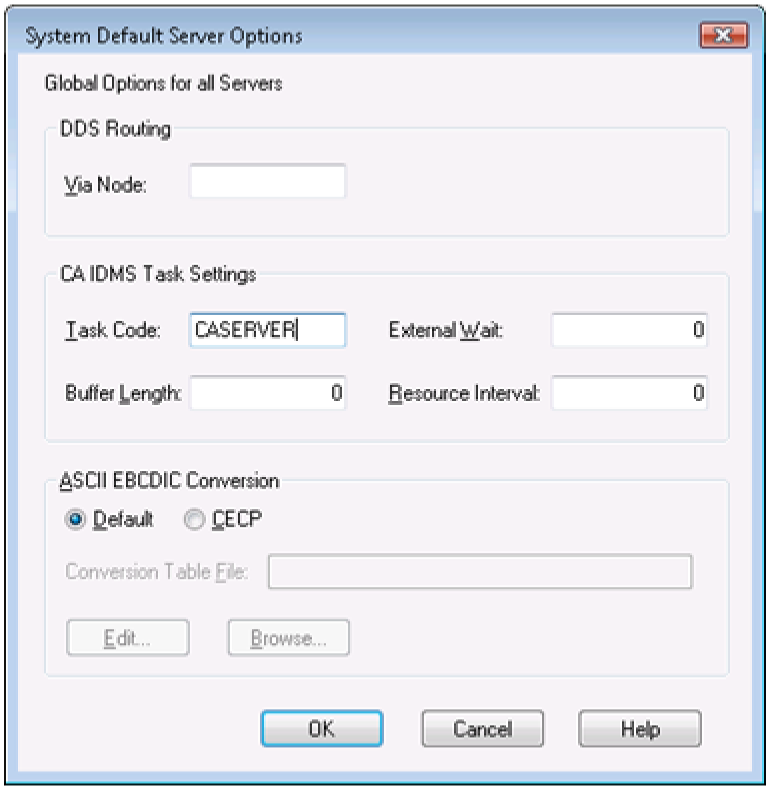
Browse

Brings up a standard “Browse” window where you can select a conversion table file.

See Chapter 4 for more information about CECP and other Internationalization options, as well as the section entitled “Setting Language Options” later in this chapter.

Setting System Default Server Options

The System Default Server Options panel is reached by clicking on the "System" button from the Server tab. Changes to these options apply to all Servers on the Windows system. For this reason, if you are using UAC on Windows Vista, you must have administrative authority to set these options.


System Default Server Options

Global Options for all Servers

DDS Routing

Via Node:

CA IDMS Task Settings

Task Code: CASERVER

External Wait: 0

Buffer Length: 0

Resource Interval: 0

ASCII EBCDIC Conversion

[✓ Default] CECP

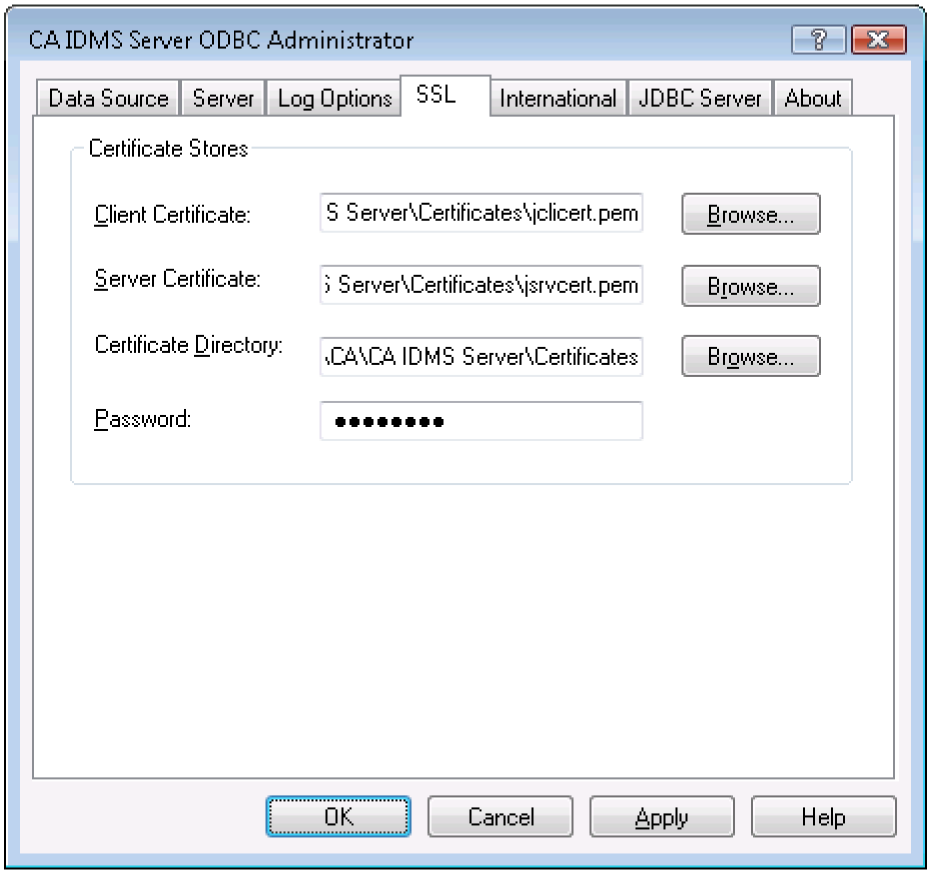
Conversion Table File:

The fields on the System Default Server Options panel are identical to those on the Advanced Server Options panel. For a description of each option, see the section entitled "Setting Advanced Server Options" earlier in this chapter.

Logging Errors and Trace Information

CA IDMS Server writes messages for some types of errors to a log file. Specify the name and directory of this log file using the Log Options tab. You can also use this tab to override default log file specification and options, or to enable tracing of JDBC, ODBC, SQL, and internal function calls.

If you are using UAC on Windows Vista, you do not have the authority to update Logging and Trace parameters without signing on as an administrator. Alternately, you can remain signed on as a standard user, but right-click the ODBC Administrator executable and select *Run as Administrator*.



Client Certificate: S Server\Certificates\jclicert.pem

Server Certificate: ; Server\Certificates\jsrvcert.pem

Certificate Directory: \CA\CA IDMS Server\Certificates

Password: ●●●●●●●●●

Log options affect all data sources. For example, if you specify a log file name, all trace entries are written to the specified file. You cannot specify different log options for different data sources. A detailed description of each option follows:

Log File Options

Log File

Specify the name of the log file for messages indicating the status of the database connection. The log file must be in a directory available for write access by all users. The log file name cannot be set or queried at runtime. The default log file name is caidms.log. If you omit path information in the file name, CA IDMS Server creates the file in the directory recommended by Microsoft for common application data as follows:

Non-Vista:

\Documents and Settings\All Users\ApplicationData\CA\IDMS Server

Vista:

\ProgramData\CA\CA IDMS Server

Max Size

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.

Max Count

Specifies the maximum number of log files. When this is greater than zero, the driver appends a numeric suffix to the "Log File Name." When the log file exceeds Max Size this number is incremented and a new log file is created. When the Max Count value is exceeded the driver re-uses the lowest log file name. The default is zero, which disables the log file rollover feature.

Both Max Size and Max Count must be greater than zero to enable the log file rollover feature.

Append To Current Log File

This option causes new information to be appended to the existing log file. If this option is chosen, care should be taken to clean out file information that is no longer needed.

Typically, tracing is enabled only to research a problem in conjunction with Technical Support. Select the check boxes under Client Trace Options as requested by Technical Support to collect trace information. Note that if Max Size and Max Count are greater than zero, this option is not available.

Client Trace Options

Client trace options control tracing on the PC, as follows:

ODBC

Enables tracing of calls to the ODBC driver.

JDBC

Enables tracing of calls to the JDBC driver.

SQL

Enables tracing of calls to the native SQL client interface.

DLL

Enables tracing of the DLL initialization function.

DTS

Enables tracing of calls to the Data Transport Services (DTS) interface.

DTS CCI

Enables tracing of calls from DTS to CAICCI.

DTS JCLI

Enables tracing of calls to the CA IDMS TCPIP interface.

FDE

Enables tracing of Format Descriptor Element (FDE) conversion calls.

FDE GEN

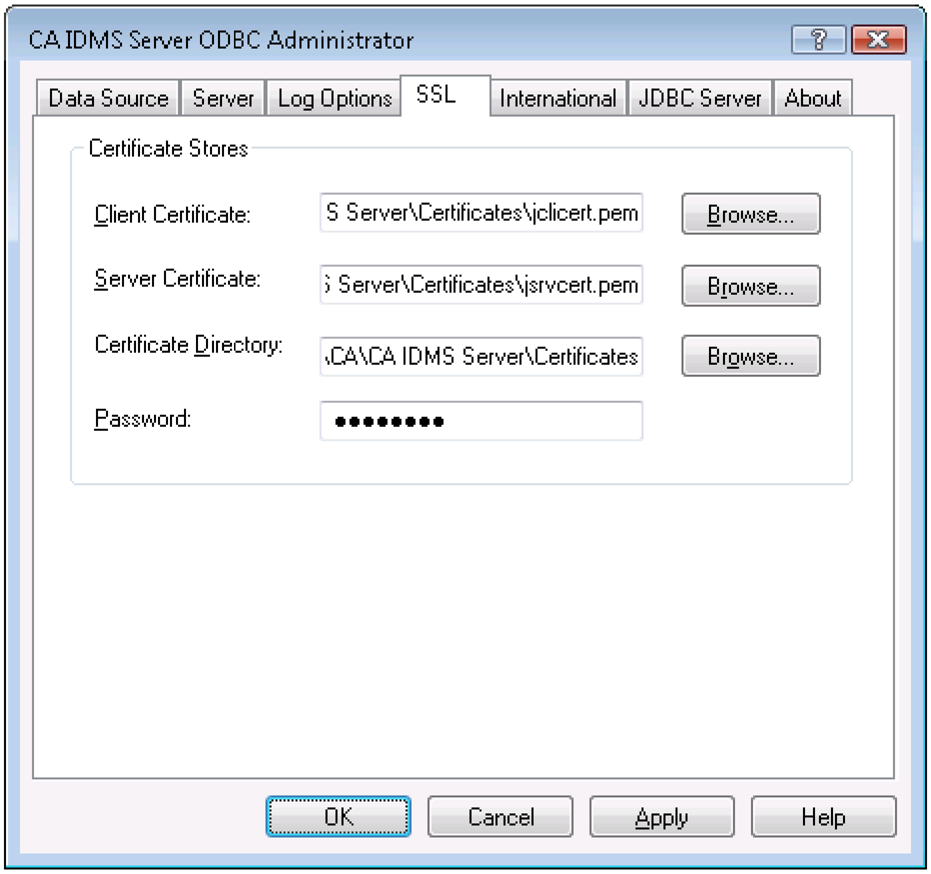
Usage is reserved.

UTIL

Enables tracing of internal utility calls.

Setting SSL Options

CA IDMS Server uses Secure Socket Layer, or SSL, to provide secure communications between the Windows client and the Mainframe server. The SSL Options tab applies only to the IDMS communications protocol. Use the CAICCI Configuration application to configure SSL for the CCI protocol.



Client Certificate: S Server\Certificates\jclicert.pem

Server Certificate: ; Server\Certificates\jsrvcert.pem

Certificate Directory: \CA\CA IDMS Server\Certificates

Password: ●●●●●●●●●

The following options apply to all Servers that use the IDMS protocol. Note that you must use the Server tab to enable the use of SSL for a connection. For more information about enabling SSL for a Server, see the section entitled "Setting Up a Server" earlier in this chapter.

Certificate Stores

Client Certificate

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. See your Network Security Administrator to determine the security configuration at your site. All certificate files must be in PEM format.

Browse

Displays a Browse dialog where you can select the client certificate file.

Server Certificate

Specifies the fully qualified name of the server certificate file. This file contains one or more "trusted" certificates that terminate the certificate chain. A certificate in this file can identify the server itself or be a signing certificate. A certificate for a server is typically generated on the mainframe and transmitted to the Windows client. All SSL connections require either a Server Certificate or one or more signing certificates. All certificate files must be in PEM format.

Browse

Displays a Browse dialog where you can select the server certificate file.

Certificate Directory

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.

Browse

Displays a standard Browse window where you can select the certificate directory.

Password

Identifies the password used for the client certificate.

This is an optional field and is only necessary when a client certificate is specified.

Setting Language Options

When CA IDMS Server transfers character data between the host system and a PC it uses translation tables based on English as spoken in the United States (U.S. English). You can override the default and create a customized translation table if your host system or PC uses code pages based on another language.

If you are using UAC on Windows Vista, you do not have the authority to update Language Options without signing on as an administrator. Alternately, you can remain signed on as a standard user, but right-click the ODBC Administrator executable and select "Run as Administrator."

The Type 2 JDBC driver first converts Unicode to the local character encoding for Windows. This data is then converted to mainframe format by the native SQL interface. The Type 3 and Type 4 JDBC drivers convert Unicode directly to mainframe format, and these options are not used.

Using the International Tab

Use the International tab to select the Country Extended Code Page (CECP) or Double Byte Character Set (DBCS) used to translate character data transferred between the PC and the host.

Under ASCII-EBCDIC Conversion, select one of the following options:

Default

Specifies the use of the default conversion tables.

CECP

Enables the CECP options in the Country Extended Code Pages box.

DBCS

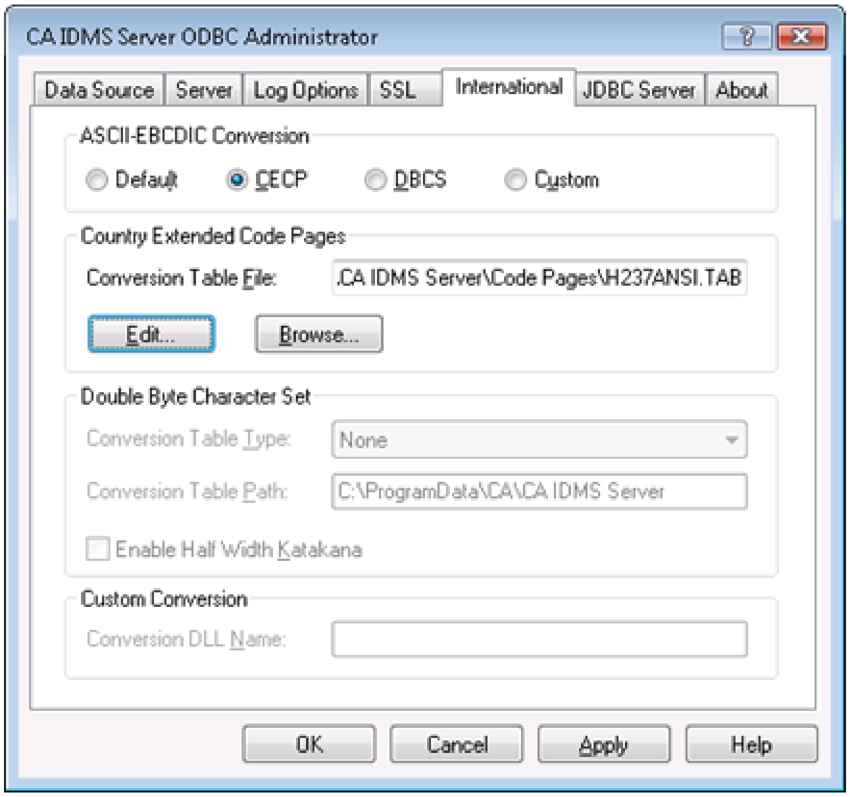
Enables the DBCS options in the Double Byte Character Set box.

Custom

Enables Custom Conversion options.

Selecting, Creating, and Editing CECP Translation Tables

Under ASCII–EBCDIC Conversion, select CECP to enable the CECP options to convert data transferred between CA IDMS and the application. Under Country Extended Code Pages, select the file containing the conversion tables.



ASCII-EBCDIC Conversion: Default [✓ CECP] DBCS Custom

Conversion Table File: .CA IDMS Server\Code Pages\H237ANSI.TAB

Conversion Table Type: None

Conversion Table Path: C:\ProgramData\CA\CA IDMS Server

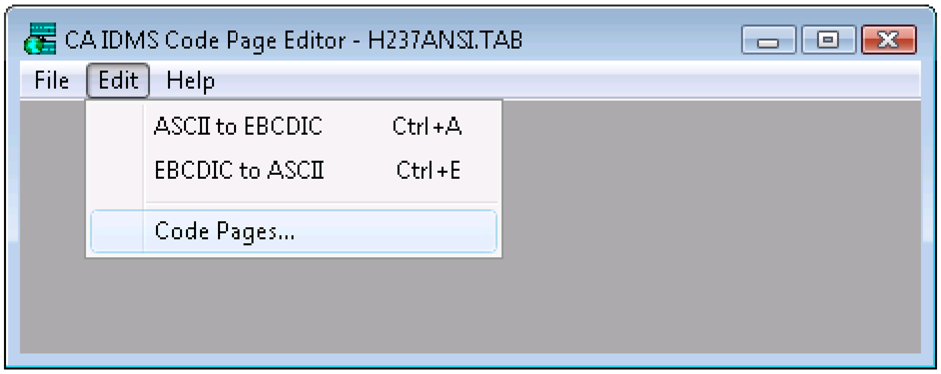
Conversion DLL Name:

To select a translation table, enter the name of the table file in the Conversion Table File field or click Browse to select from the list of available files.

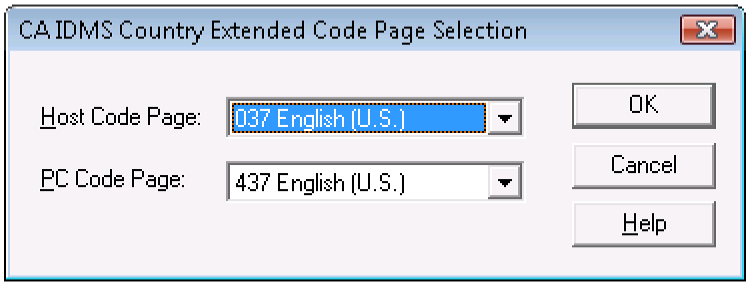
Click Edit to activate the Translation Editor to create or edit a translation table.

Creating or Editing a Translation Table

From the menu bar, select File, Open to open an existing translation table.



For a new or existing translation table, select Edit, Code Pages to access the CA IDMS Country Extended Code Page Selection dialog. This dialog lets you select the code pages to use for your translation table.



Host Code Page: 037 English (U.S.)
PC Code Page: 437 English (U.S.)

The Host Code Page list includes the following Code Pages for the EBCDIC character set on the mainframe:

037 English (U.S.)

English and most other European languages

273 German, Austrian

German and Austrian German

277 Norwegian

Norwegian

278 Finnish, Swedish

Finnish and Swedish

280 Italian

Italian

284 Spanish

Spanish

285 English (U.K.)

English and most European languages

297 French (AZERTY)

French, using the AZERTY keyboard

500 Belgian, Swiss

Belgian, Swiss French, and Swiss German

The *PC Code Page* list box includes the following Code Pages for the ASCII character set:

437 English (U.S.)

English and most other European languages

850 Multilingual (Latin I)

Most languages using the Latin alphabet

852 Slavic (Latin II)

Slavic languages using the Latin alphabet

860 Portuguese

English and Portuguese

863 Canadian-French

English and French Canadian

865 Nordic

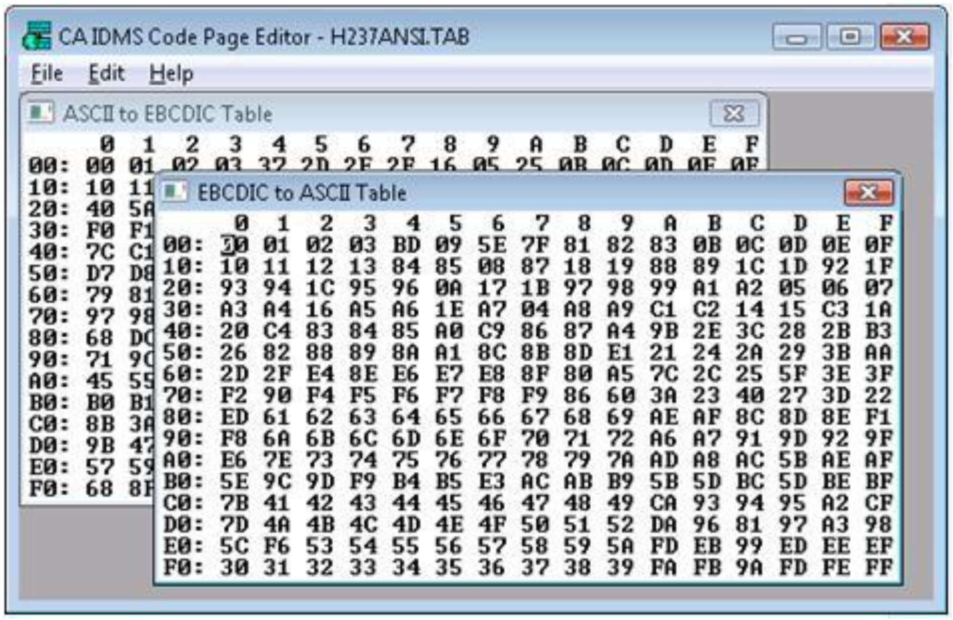
Scandinavian languages (Swedish, Norwegian)

Customizing a Translation Table

After creating a translation table, you may need to add EBCDIC/ASCII conversions that are not supported in the standard code pages. The Translation Table Editor provides two edit windows: one for the ASCII to EBCDIC translation table and the other for the EBCDIC to ASCII translation table. To activate either window, select the appropriate option from the Edit menu.

Each window displays a table of 256 hexadecimal values. Each entry in the table represents the output character set code value indexed by the input character set code value.

For example, the following window represents the ASCII to EBCDIC translation table for Canadian French on the PC and U.S. English on the host machine. The ASCII value for a space (' ') in the Canadian French code page is 20 (in hexadecimal). The corresponding EBCDIC value for a space in the U.S. English code page is 40.



ASCII to EBCDIC Table

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00: 00 01 02 03 37 2D 2E 2F 16 05 25 0B 0C 0D 0E 0F
10: 10 11
20: 40 5A
30: F0 F1
40: 7C C1
50: D7 D8
60: 79 81
70: 97 98
80: 68 DC
90: 71 9C
A0: 45 55
B0: B0 B1
C0: 8B 3A
D0: 9B 47
E0: 57 59
F0: 68 8F
```

EBCDIC to ASCII Table

```
     0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
00: 00 01 02 03 BD 09 5E 7F 81 82 83 0B 0C 0D 0E 0F
10: 10 11 12 13 84 85 08 87 18 19 88 89 1C 1D 92 1F
20: 93 94 1C 95 96 0A 17 1B 97 98 99 A1 A2 05 06 07
30: A3 A4 16 A5 A6 1E A7 04 A8 A9 C1 C2 14 15 C3 1A
40: 20 C4 83 84 85 A0 C9 86 87 A4 9B 2E 3C 28 2B B3
50: 26 82 88 89 8A A1 8C 8B 8D E1 21 24 2A 29 3B AA
60: 2D 2F E4 8E E6 E7 E8 8F 80 A5 7C 2C 25 5F 3E 3F
70: F2 90 F4 F5 F6 F7 F8 F9 86 60 3A 23 40 27 3D 22
80: ED 61 62 63 64 65 66 67 68 69 AE AF 8C 8D 8E F1
90: F8 6A 6B 6C 6D 6E 6F 70 71 72 A6 A7 91 9D 92 9F
A0: E6 7E 73 74 75 76 77 78 79 7A AD A8 AC 5B AE AF
B0: 5E 9C 9D F9 B4 B5 E3 AC AB B9 5B 5D BC 5D BE BF
C0: 7B 41 42 43 44 45 46 47 48 49 CA 93 94 95 A2 CF
D0: 7D 4A 4B 4C 4D 4E 4F 50 51 52 DA 96 81 97 A3 98
E0: 5C F6 53 54 55 56 57 58 59 5A FD EB 99 ED EE EF
F0: 30 31 32 33 34 35 36 37 38 39 FA FB 9A FD FE FF
```

The generated tables convert control codes between their ASCII and EBCDIC equivalents, where possible. (Releases prior to CA IDMS Server r4.0 converted all control codes to x'00' or null bytes.)

To customize the table, use the mouse or keyboard to select a hexadecimal value and replace it with another. The editor ignores all characters except the numbers 0 through 9 and letters A - F (including lowercase). Use the mouse to move the cursor or use the following keystrokes:

Key	Moves Cursor
Arrow keys	One digit in the direction of the arrow
Home	To beginning of row
End	To end of row
PageUp	To top row
PageDown	To bottom row
Enter	Beginning of next row
Ctrl+Left Arrow, Right Arrow	Left or right one entry
Ctrl+Home, End	Beginning or end of table

Saving a Translation Table

To save a translation table, choose Save from the File menu. To save a translation table under a new name, select File, Save As. The default file extension for translation table files is .tab.

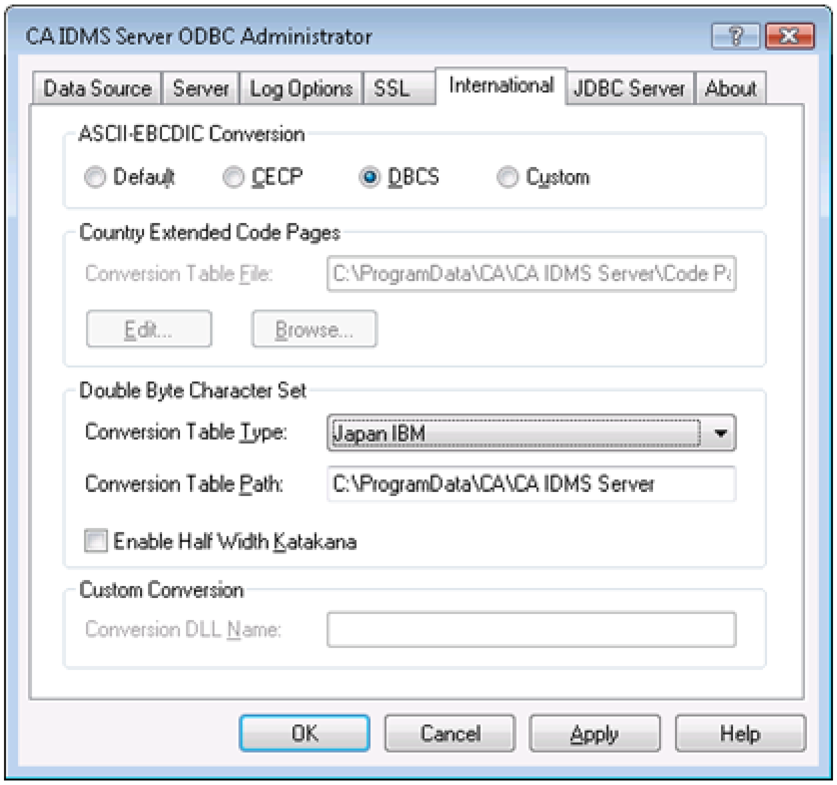
Included Tables

Several code page conversion tables are provided with this release and are installed in the CA IDMS Server directory. These files are identified by the extension .tab and contain tables used at runtime to convert between ASCII and EBCDIC. These tables are:

Table	Converts
cp037.tab	Host code page 037 to ANSI 8859-1. The Type 3 and Type 4 JDBC drivers use this conversion table when the Cp037 converter requested by the host is not available.
cp1047.tab	Host code page 1047 to ANSI 8859-1. The Type 3 and Type 4 JDBC drivers use this conversion table when the Cp1047 converter requested by the host is not available.
danish.tab	Host code page 037 to pc code page 850. Control codes are converted to their corresponding values.
h237ansi.tab	Host code page 237 (Austrian/German) to ANSI. Control codes are converted from EBCDIC to x'01' and from ASCII to x'00'.
sgeransi.tab	Siemens German to ANSI. Control codes are converted from EBCDIC to x'01' and from ASCII to x'00'.
swedish.tab	Host code page 037 to pc code page 850. Control codes are converted to their corresponding values.

Enabling DBCS Processing

DBCS options enable the conversion of multi-byte character data exchanged by CA IDMS Server and the CA IDMS system. Under ASCII-EBCDIC Conversion, select the DBCS option.



ASCII-EBCDIC Conversion
Default [✓ DBCS] CECP Custom

Country Extended Code Pages
Conversion Table File: C:\ProgramData\CA\CA IDMS Server\Code P

Edit... Browse...

Double Byte Character Set
Conversion Table Type: Japan IBM
Conversion Table Path: C:\ProgramData\CA\CA IDMS Server
☐ Enable Half Width Katakana

Custom Conversion
Conversion DLL Name:

Under Double Byte Character Set, set the following options:

Conversion Table Type

Allows you to select the types of DBCS used by your CA IDMS system from a drop down menu.

Conversion Table Path

Specifies the path of the subdirectory containing the DBCS conversion tables. Typically, the default is accepted.

Enable Half Width Katakana

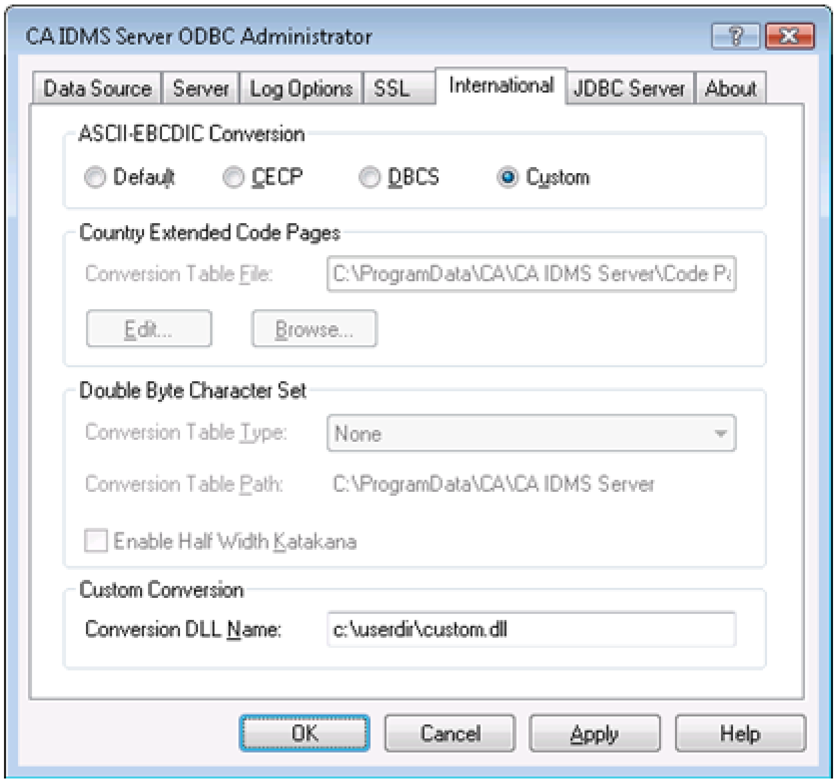
Enables half width Katakana support when DBCS is enabled when this box is checked. All lowercase characters in CHAR and VARCHAR data are treated as half width Katakana. This does not affect GRAPHIC, VARGRAPHIC, and mixed data within SO and SI in CHAR and VARCHAR types. Only uppercase Roman text can be transferred between the mainframe and the PC when this option is enabled.

Using a Custom Conversion DLL

A Custom Conversion DLL is used to convert character data exchanged by CA IDMS Server and the CA IDMS system. This can be useful when the ASCII - EBCDIC conversions cannot be specified by modifying the CECP tables. The following sections describe implementation information for a Custom Conversion DLL.

Enabling a Custom Conversion DLL

On the International tab, under ASCII-EBCDIC Conversion, select Custom.



Under Custom Conversion, in the Conversion DLL Name field, specify the name of the Custom Conversion DLL. Include the path if the DLL is not in a directory that is searched automatically by Windows, such as the SYSTEM32 subdirectory or a directory specified in the PATH.

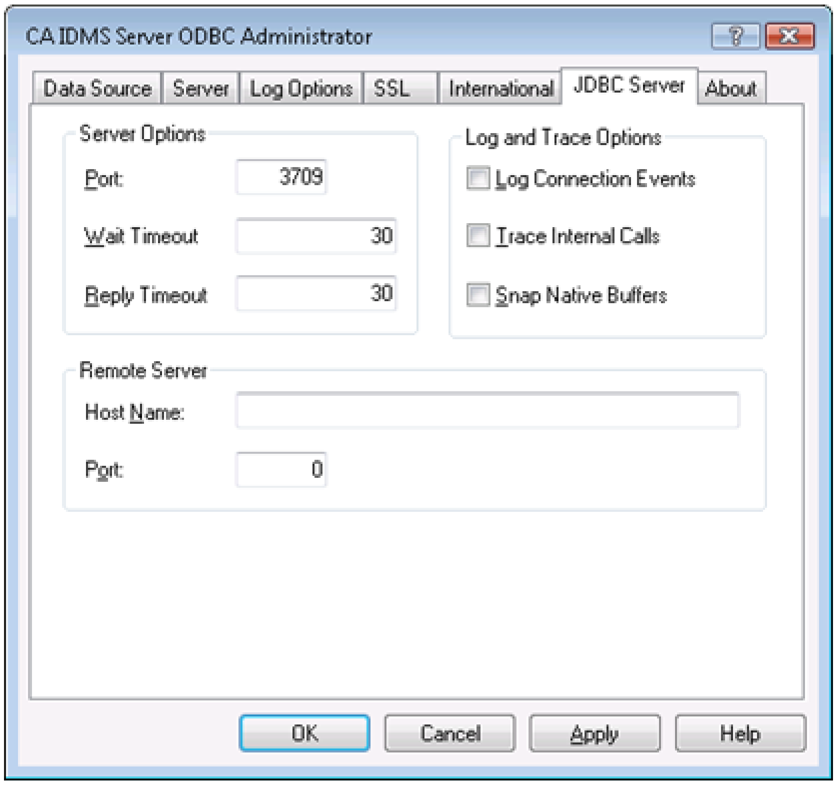
Developing a Custom Conversion DLL

You can develop a custom conversion DLL in any language that supports the Microsoft Windows DLL calling conventions. See the ODBC Programmer's Reference for information about implementing a custom conversion DLL.

Configuring the JDBC Server

Use the CA IDMS Server ODBC Administrator to configure the JDBC server to allow access to CA IDMS from client programs using the type 3 JDBC driver. Neither the JDBC driver nor the JDBC server actually uses ODBC at runtime. To configure the JDBC server, select any CA IDMS data source from the CA IDMS Server ODBC Administrator, and then click the JDBC Server tab.

If you are using UAC on Windows Vista, you do not have the authority to update JDBC Server parameters without signing on as an administrator. Alternately, you can remain signed on as a standard user, but right-click the ODBC Administrator executable and select "Run as Administrator."



Server Options
Port: 3709
Wait Timeout: 30
Reply Timeout: 30

Log and Trace Options
[ ] Log Connection Events
[ ] Trace Internal Calls
[ ] Snap Native Buffers

Remote Server
Host Name:
Port: 0

Server Options

Port

Specify the TCP/IP port which the JDBC server uses to listen for connection requests. JDBC applications should specify this value in the Uniform Resource Locator (URL) that identifies the database. The default is 3709.

Note: For information about the URL recognized by the JDBC driver, see the "JDBC Programmer Reference" appendix.

Wait Timeout

The number of seconds the JDBC server waits for a request from the JDBC driver. When this value is exceeded, the JDBC server considers the connection to have failed. The default setting, 0, causes the JDBC server to wait indefinitely.

Reply Timeout

Specifies the number of seconds the JDBC server waits for a response from CA IDMS. When this value is exceeded, the JDBC server considers the connection to have failed. The default setting, 0, causes the JDBC server to wait indefinitely.

Log and Trace Options

Log Connection Events

Enables logging of connection requests and terminations by the JDBC server to the Windows Application Event Log. By default, only startup, shutdown, and error events are logged. This option is deprecated, and kept for compatibility with earlier versions of CA IDMS Server.

Trace Internal Calls

Enables tracing of debugging information to the CA IDMS Server log file. Only internal method calls made by the Java code are traced. Use the Log Options tab to enable tracing of native method calls.

Snap Native Buffers

Enables display of the data buffers sent and received by the JDBC server in the CA IDMS Server log file.

Remote Server

It is possible to route JDBC connections to another JDBC server before communicating with CA IDMS. You can also route connections directly to a CA IDMS r16 SP2 or later system. This can be useful when security requirements prevent the machine on which the web server is running from directly connecting to the mainframe.

Host Name

Specifies the DNS name or IP address of the remote JDBC server machine. For CA IDMS r16 SP2 or later, this could also be the DNS name or TCP/IP address associated with the IDMS system's TCPIP line.

Port

Specifies the listener port of the remote JDBC server. The default is 3709.

Property File Information

You can also specify options for the JDBC driver and JDBC server in the caidms.properties file, which has the same format on all platforms.

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

Chapter 6: Using the Client on Windows

This chapter 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.

This section contains the following topics:

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.

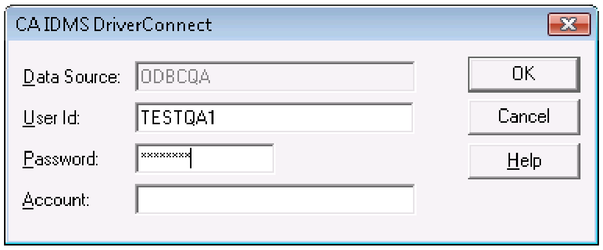
Although the JDBC driver uses the same types of information, it does not display any dialogs, leaving the collection of such information to the JDBC application.

Note: For more information about connecting to a data source using JDBC, see the chapter "JDBC Programmer Reference."

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.

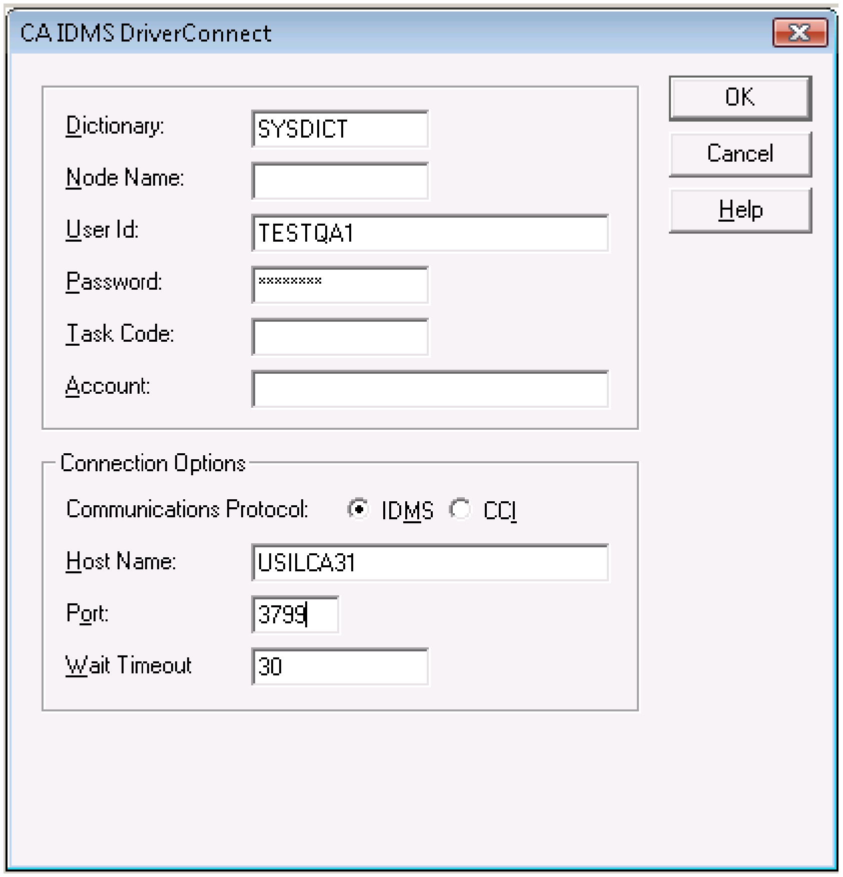


Data Source: ODBCQA
User Id: TESTQA1
Password: ********
Account:

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

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.



Dictionary:	SYSDICT
Node Name:	
User Id:	TESTQA1
Password:	********
Task Code:	
Account:	

Connection Options

Communications Protocol:	[✓ IDMS] CCI
Host Name:	USILCA31
Port:	3799
Wait Timeout	30

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.

Account

Specifies your account, if your site requires it.

Connection Options

Specify the following options in the Connection Options section:

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 chapter "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:

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 chapter "Using the Client on z/OS." For more detailed command information, see Using the JDBC Driver in the chapter "Using the Java Client."

Chapter 7: Installing the Client on z/OS

This chapter and the "Configuring the Client on z/OS" and "Using the Client on z/OS" chapters describe the installation and use of CA IDMS Server in the Unix System Services (USS) environment on z/OS for use with the Type 2 JDBC driver and the JDBC server.

This section contains the following topics:

Installation Process

CA IDMS Server for z/OS is distributed on the CA IDMS Server r16.1 CD. The installation process consists of three parts:

1. Copy the distribution files from the directory on the CA IDMS Server r16.1 CD to any directory on your hard disk
2. Upload the distribution files to z/OS using FTP.
3. Build and populate the CA IDMS Server HFS directory structure.

   This part of the installation process is performed with a series of commands entered into the OMVS shell. The user ID used to perform some of the steps may need to be authorized to allocate data sets on SMS packs, or to set up a new OMVS group and owner, depending on the steps performed.

CA IDMS Server for z/OS calls CAICCI directly, and does not use the OMVS interface, libcci.so.

Installing the Client Components for UNIX System Services

The procedure to install the CA IDMS Server client components for UNIX System Services involves steps performed in Windows and in z/OS.

Step 1: Load the Installation Files

Copy file "hfs.tar", within directory "zos" on the CA IDMS Server r16.1 CD, to a directory on your hard disk.

Step 2: Allocate the HFS

An HFS must be established to store the CA IDMS Database Server Option executables and configuration files. CA IDMS Server requires approximately 2 MB in the HFS during the installation. Log files created at runtime may require additional space, and can be allocated in a separate HFS devoted to temporary files.

You can install CA IDMS Server into an existing HFS if space is available, or you can set up a new HFS specifically for CA IDMS Server. Only users with OMVS superuser authority can allocate and mount a new HFS. If you decide to set up a new HFS, you can also establish a new OMVS UID and GID to serve as the owner and administrator of the CA IDMS Server HFS files. Consult with your site Security Administrator for assistance in performing this step.

Step 3: Create the Installation Directory in the HFS

Under OMVS, create the CA IDMS Server installation directory in the HFS, referred to here as /idmsdir. If you allocated a new HFS, the OMVS superuser must perform the following steps:

1. Create a mount point and mount the new HFS
2. Declare the owning group and user for the mount point
3. Set up initial file permissions
4. Execute the following commands, invoked by entering the TSO OMVS command:

   ```
   mkdir -m 775 /idmsdir
   /samples/mountx /idmsdir IDMSSRV.HFS
   chown ISADMIN:ISGROUP /idmsdir
   ```

   Note: File permission bits are set to 775, indicating that only the owning user or users connected to the owning group, can update this directory. All other users have only read and execute authority.

   To make this mount permanent, update the BPXPRMxx member in SYS1.PARMLIB and add a mount entry. The following is an example of the statement to add:

   ```
   MOUNT FILESYSTEM('HFSDSN')
       MOUNTPOINT('/idmsdir')
       TYPE(HFS) MODE(RDWR)
   ```

   HFSDSN

   Specifies the name of the CA IDMS Server HFS data set.

   idmsdir

   Specifies the name of the CA IDMS Server installation directory.

Step 4: Copy and Extract the TAR File

Perform the following steps to populate the CA IDMS Server HFS:

1. In Windows, use FTP to copy file "hfs.tar" which was downloaded in Step 1 to the CA IDMS Server HFS directory, as defined in Step 2. Ensure that you specify BINARY mode on this transfer.

2. In OMVS, extract the CA IDMS Server subdirectories and files by positioning yourself in the CA IDMS Server directory, and using the following command:

   ```
   pax –rvf hfs.tar
   ```

Note: The hfs.tar file is not used after this step and can be deleted from the HFS to save space.

Step 5: Copy the Sample JCL Files (Optional)

Several MVS jobstreams have been included on the CA IDMS Server HFS under directory "sampjcl" as follows:

jsrvresu

Resumes execution of the IDMS Java Server

jsrvstop

Stops the IDMS Java Server

jsrvstrt

Starts the IDMS Java Server

jsrvsusp

Suspends execution of the IDMS Java Server

- A file called "jobcard" has also been included which should be tailored to your site-specific requirements and copied at the beginning of each jobstream. The jobcard file has a symbolic variable called IDMSDIR that must be set to the USS directory name where you installed your CA IDMS Server software. This directory name must begin with a forward slash (/) and be fully qualified.
- You can copy all of these files into an MVS PDS using whatever means is easiest for you to facilitate the control of the IDMS Java Server from an MVS batch job.

Chapter 8: Configuring the Client on z/OS

This chapter describes how to configure CA IDMS Server in the z/OS UNIX System Services (USS) environment for use by JDBC-enabled applications. The information in this chapter is particularly relevant to the use of the JDBC server. It describes the environment variable settings and configuration file information needed to access a CA IDMS database. It assumes that you are familiar with the z/OS USS shell and HFS.

This section contains the following topics:

Configuring CA IDMS

CA IDMS Server is installed into a subdirectory in the HFS. This subdirectory, specified when the product is installed, is referred to in this document as */idmsdir*. Its structure is as follows:

/idmsdir

Specifies the default location for the configuration file and example SQL script.

***/idmsdir*/bin**

Specifies shell scripts to run the JDBC server and sample Java applications.

***/idmsdir*/bin/mvs**

Specifies compiled executable files, including the JDBC server service wrapper.

***/idmsdir*/classes**

Specifies the caidms.properties file and the IdmsExample sample application. It is also used for classes supplied as part of an APAR and additional helper classes provided by CA.

***/idmsdir*/lib**

Specifies the Java archive files (including the JDBC driver and JDBC server).

***/idmsdir*/lib/mvs**

Specifies the shared object libraries used to implement the native methods and client interfaces.

/idmsdir/sampjcl

Specifies the sample jobstreams for starting and stopping the CA IDMS Server from MVS. These jobs should be copied to an MVS PDS and customized to meet your site requirements. The sample job card member should be customized and included in each of the jobstreams.

***/idmsdir/*src**

Specifies the sample Java source code.

For more information about configuring CA IDMS Server for use with applets and JDBC-enabled applications running on other platforms, see the Using the JDBC Driver topic in the chapter "Using the Java Client on Any Platform."

Specifying Environment Variables

You must specify the locations of the executables, DLLs, and Java class files for a z/OS application to use CA IDMS Server in the USS environment. Set the standard UNIX environment variables to specify these locations:

PATH

Specifies the locations of executable files.

LIBPATH

Specifies the locations of DLL files.

CLASSPATH

Specifies the locations of Java class files.

For example, to run a JDBC application that uses the JDBC driver, these variables could be set as follows:

```
set PATH=$JAVA_HOME/bin:$PATH
export PATH
set LIBPATH=/idmsdir/mvs:$LIBPATH
export LIBPATH
set CLASSPATH=/idmsdir/classes:/idmsdir/lib/idmsjdbc.jar:$CLASSPATH
export CLASSPATH
```

In this case, $JAVA_HOME identifies the directory where Java is installed, and */idmsdir* represents the directory chosen when CA IDMS Server was installed. Note that it is not necessary to include the */idmsdir*/bin directory in the PATH to run a Java application. It is not necessary to set the PATH, LIBPATH, and CLASSPATH environment variables when using the supplied shell scripts to run the JDBC server. These environment variables are automatically set in the shell scripts installed in the */idmsdir*/bin directory, equivalent to the following:

```
set PATH=$JAVA_HOME/bin/idmsdir/bin:$PATH
export PATH
set LIBPATH=/idmsdir/lib/mvs:$LIBPATH
export LIBPATH
set CLASSPATH=/idmsdir/classes:/idmsdir/lib/idmsjsrv.jar:$CLASSPATH
export CLASSPATH
```

Note: The shell scripts assume that JAVA_HOME has been set, typically in a user profile.

Optional environment variables specific to CA IDMS Server include:

IDMS_CFG_PATH

Specifies the configuration file name or path.

IDMS_CFG_RELOAD

Forces reloading of the configuration file.

Important! When invoking Unix scripts and programs using the BPXBATCH utility, it is possible to set environment variables using the STDENV DD statement. This is particularly useful when scheduled batch jobs are used to automate startup and shutdown of the JDBC server.

Specifying Environment Variables for IPv6

Special considerations apply to using IPv6 with the JDBC server:

- Customers who run multiple TCP/IP stacks must set stack affinity to the appropriate IPv6 stack using the _BPXK_SETIBMOPT_TRANSPORT environment variable. In this instance you must also code the appropriate host name or IP address in the Host parameter of the configuration file as detailed in "Configuring the JDBC Server" topic in this chapter.
- It may be necessary to define a new TCPIP.DATA file which provides a special DNS Resolver configuration for IPv6 hosts and addresses. This file must be specified using the RESOLVER_CONFIG environment variable.

For example, prior to starting the JDBC server, the following statements can be run as part of a user signon profile. Alternatively, they can be included at the beginning of the CA-IDMS Server .idms_wrapper script located in the CA IDMS Server bin subdirectory, in which case they must be run when the JDBC server is started with the supplied shell script.

export JAVA_HOME="your-jvm 1.6 location here"
export _BPXK_SETIBMOPT_TRANSPORT="*your-ipv6-proc-name-here*"

```
export RESOLVER_CONFIG="//'your.tcpipv6.data.file.here'"
```

Editing the Configuration File

The configuration file contains data source definitions, CA IDMS system access path information, global option settings, and JDBC server options, corresponding to the information maintained in the registry on the Windows platform. The file is formatted as a text file with sections containing lists of key-value pair parameters, similar to a Windows .ini file. You must edit this file manually for z/OS.

The default file name is caidms.cfg. CA IDMS Server first looks for this file in the current directory. If it is not there, CA IDMS Server looks in the installation directory. You can use the IDMS_CFG_PATH environment variable to specify a different name or directory.

For example, to locate the caidms.cfg file in the application directory */idmsdir*:

```
set IDMS_CFG_PATH=/idmsdir/
export IDMS_CFG_PATH
```

For performance reasons, the configuration file is cached in memory the first time the libidmsutil.so DLL is loaded into a process. Therefore, changes to the configuration file may not take effect for a running process until the process is quiesced and restarted. For the JDBC Server, use the suspend and resume commands documented in the chapter on "Using the Client on z/OS." For testing and other implementations where performance is not critical, the cached configuration file can be refreshed more frequently or caching can be turned off completely using the IDMS_CFG_RELOAD environment variable, or with the CacheConfig option. For more information, see the appendix "Configuration File Information."

Data Source Definitions

A JDBC-enabled application can connect to a CA IDMS database using the DriverManager class with a URL or using JNDI with an IdmsDataSource object. Either technique can reference a data source name similar to an ODBC data source. This DSN is defined in the configuration file, where it is associated with the dictionary name of the catalog defining the SQL schema, a node name identifying the CA IDMS system, and other optional information.

The following sample illustrates a data source definition defined in the configuration file:

```
[APPLDICT]
Dictionary=APPLDICT
Server=SYST0001

[Server SYST0001]
Resource=SYST0001
AlternateTask=CASERVER
```

This syntax lets you use meaningful names for the data source and server names. Using an explicit server section lets you specify optional information for a CA IDMS system. When using all default values, this is equivalent to the following minimal data source definition:

```
[APPLDICT]
Server=SYST0001
```

Bracket characters "[" and "]" may be difficult to use on 3270 terminals or emulators. The dollar character "$" can be used as a substitute as seen in the following example:

```
$APPLDICT$
Server=SYST0001
```

You can also specify this information using DriverPropertyInfo objects or in the IdmsDataSource itself, instead of in the configuration file.

Note: For a complete description of the connection information, see the chapter "JDBC Programmer Reference."

Configuring the JDBC Server

The JDBC server can be customized with settings in the [Proxy] section of the configuration file as described in the "Configuration File Information" appendix. The following options are often specified on z/OS:

Host

Forces the JDBC server to listen for connection requests on a specific TCP/IP stack. This is not needed when the host has only one stack.

Port

Specifies the correct port in the URL by client applications if the port is changed from the default (3709).

Encoding

Specifies the default platform encoding (CP1047) on z/OS OMVS. CP037 is a standard IBM encoding supported by the Sun and IBM Java implementations and is the default set in the configuration file when the product is installed. The JDBC driver includes built-in support for these encodings. Other mainframe character converter classes may not be available on all platforms. Since character conversion can be offloaded to the client when the character conversion classes are available, performance may be improved by specifying an encoding that can be done on the client.

WaitTimeOut

(Recommended) Specifies how long the JDBC server waits for the next request from a connected client.

Other Configuration File Information

You can specify global options, including the location of the CA IDMS log (trace) file, trace flags for debugging, and character set encoding in the [Options] section of the configuration file.

Note: For detailed information about all options and settings in the configuration file, see the appendix "Configuration File Information."

Properties File Information

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.

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

Chapter 9: Using the Client on z/OS

This chapter describes how to use CA IDMS Server in the USS 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.

This section contains the following topics:

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 chapter "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 appendix "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:

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

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

jsrvsusp

Suspends the JDBC server.

jsrvresu

Resumes 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 /sampjcl 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 Guide.*

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 chapter "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 about the command line options, see Using the JDBC Server in the chapter "Using the Java Client." For more information about customizing the sample JCL, see the chapter "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:

- UJS0001I - Server started
- UJS0002I - Server stopped
- UJS0003D - Server stopping
- UJS0004D - Server waiting for connection
- UJS0005I - Server suspended
- UJS0006I - Server resumed
- UJS0101I - Client thread started
- UJS0102I - Client thread stopped
- UJS0103D - Client thread stopping
- UJS0104I - Client thread to remote server
- UJS0105D - Client thread loaded class

- UJS0200E - General error
- UJS0201E - Socket I/O error
- UJS0202E - Packet protocol error

Because the message text can include additional information, only the message number should be used to identify specific events.

Note: For more information, see the appendix "Configuration File Information."

Chapter 10: 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 chapter 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.

This section contains the following topics:

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.

The javadoc files are found in \doc\javadoc_idms.zip, and there is also a link in the HTML Bookshelf.

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
   ```

Note: For more information about the URL format, the Connection Parameters, and IDMS Data Source definition, see the chapter "JDBC Programmer Reference."

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
   ```

2. 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
   ```

3. 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:

Options	Description
-?	Prints this information
-h host	Host listener name or IP address
-p port	Host listener IP port
-q count	Host listener queue length
-r host	Remote host name or IP address
-s port	Remote IP port
-c	Enables control by remote client
-e encoding	Overrides platform encoding
-u	Specifies Unicode fallback encoding
-w seconds	Client wait timeout interval
-t seconds	Server reply timeout interval
-b seconds	Socket blocking timeout interval
-v [level]	Syslog message level (level = 10 if not specified)
-l level	Trace log message level
-d option [option]	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
-k	Enables SSL client support
-a	Requires SSL client certificate
-y	Enables SSL to remote JDBC Server or CA IDMS r16 SP2 or later
-i class [class]	Includes class in trace
-x class [class]	Excludes class from trace

Note: For detailed information about these options, see the appendices "Configuration File Information" and "Properties File Information."

Chapter 11: 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.

This section contains the following topics:

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

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 Guide, 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_DESC_PARAMETER_TYPE

Connection Attributes:

- SQL_ATTR_CONNECTION_TIMEOUT
- SQL_ATTR_LOGIN TIMEOUT

Statement Attributes:

- SQL_ATTR_CONCURRENCY
- SQL_ATTR_ENABLE_AUTO_IPD
- SQL_ATTR_QUERY_TIMEOUT

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 Guide.

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

SQL Statement	Comments
CREATE TABLE clauses:	■ 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.
DROP TABLE	RESTRICT—CA IDMS supports CASCADE, but does not support the RESTRICT keyword. The absence of CASCADE implies RESTRICT.
GRANT UPDATE (column-list) REFERENCES (column-list)	CA IDMS does not support column level security. CA IDMS driver removes the column list and grants UPDATE to all columns of the table.
REVOKE	CASCADE/RESTRICT—CA IDMS does not support the CASCADE and RESTRICT options on REVOKE.

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

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 about 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:

CA IDMS Data Type	ODBC Data Type
BINARY	SQL_BINARY
CHAR	SQL_CHAR
CHARACTER VARYING (VARCHAR synonym)	SQL_VARCHAR
DATE	SQL_TYPE_DATE
DECIMAL	SQL_DECIMAL
DOUBLE PRECISION*	SQL_DOUBLE
FLOAT*	SQL_FLOAT
GRAPHIC (DBCS Disabled)	SQL_BINARY
GRAPHIC (DBCS Enabled)	CAID_GRAPHIC
INTEGER	SQL_INTEGER
LONGINT	SQL_BIGINT
NUMERIC	SQL_NUMERIC
REAL*	SQL_REAL
SMALLINT	SQL_SMALLINT

CA IDMS Data Type	ODBC Data Type
TIME	SQL_TYPE_TIME
TIMESTAMP	SQL_TYPE_TIMESTAMP
UNSIGNED DECIMAL	SQL_DECIMAL
UNSIGNED NUMERIC	SQL_NUMERIC
VARCHAR	SQL_VARCHAR
VARGRAPHIC (DBCS Enabled)	CAID_VARGRAPHIC
VARGRAPHIC (DBCS Disabled)	SQL_BINARY

Note: * Floating point conversion subject to rounding errors due to format differences.

ODBC to CA IDMS Data Type Mapping

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

ODBC Data Type	CA IDMS Data Type
CAID_GRAPHIC - DBCS Enabled	GRAPHIC
CAID_VARGRAPHIC - DBCS Enabled	VARGRAPHIC
SQL_BINARY	BINARY
SQL_LONGVARBINARY	BINARY
SQL_CHAR	CHAR
SQL_TYPE_DATE	DATE
SQL_DECIMAL	DECIMAL
SQL_DOUBLE	DOUBLE PRECISION
SQL_FLOAT*****	DOUBLE PRECISION
SQL_GUID	CHAR
SQL_REAL*****	REAL
SQL_INTEGER	INTEGER
SQL_BIGINT	LONGINT
SQL_NUMERIC	NUMERIC
SQL_BIT	SMALLINT
SQL_SMALLINT	SMALLINT

ODBC Data Type	CA IDMS Data Type
SQL_TINYINT	SMALLINT
SQL_TYPE_TIME	TIME
SQL_TYPE_TIMESTAMP	TIMESTAMP
SQL_LONGVARCHAR	VARCHAR
SQL_VARCHAR	VARCHAR
All Interval Types	CHAR

Note: * Floating point conversion subject to rounding errors due to format differences.

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

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.

Keyword	Defined By	Attribute Value
DSN	Microsoft	Data source name
DRIVER	Microsoft	Driver name (cannot use with DSN)
DICT	CA	Dictionary name (use with DRIVER only)
NODE	CA	Node name (use with DRIVER only)
TASK	CA	Alternate task code (use with DRIVER only)
UID	Microsoft	User ID
PWD	Microsoft	Password
ACCT	CA	Account information, if used
CCINAME	CA	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.
CCIPORT	CA	CAICCI host server port (optional, use with DRIVER only). This option is required when the CCINAME option is specified.
WAIT	CA	CAICCI reply wait timeout (optional, use with DRIVER only)
HOST	CA	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.
PORT	CA	TCPIP port number of the IDMSJSRV listener (optional, use with DRIVER only). This option is required when the HOST option is specified.
SSL	CA	Secure Sockets Layer indicator (optional, use with DRIVER only)
PROGRAM	CA	Program name (optional, use with DRIVER only)

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 about calling the SQLDriverConnect function.
- Online help and the chapter "Using the Client on Windows" for more information about the DriverConnect dialog.
- The chapter "Configuring the Client on Windows" for more information about 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_COMMITTED

(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 appendix. 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 about 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)

Procedures

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 about procedures using the following functions:

SQLGetInfo

Returns information about 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_PT_PROCEDURE, 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":

Name	Number	Type
NUM_INPUT_PARMS	4	SMALLINT
NUM_OUTPUT_PARMS	5	SMALLINT
NUM_RESULT_SETS	6	SMALLINT

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

CA IDMS r17, or later, supports procedures that return result sets. The application uses the following functions to navigate though any returned result sets:

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.

The application uses the normal result set functions to retrieve the column data values.

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 usedfor 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 Guide*.

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

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 SQLDescriberParam. 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

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 updateable 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 Guide* 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 DBCInit(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

```
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

```
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

nBufferLen

Input 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

DBCStoPC

Syntax

```
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

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

```
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 DBCEnd(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:

DLL Function	Description
DBCSAlloc	Called before any other processing is done.
DBCSInit	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.
SetDBCSOption	■ 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.
GetDBCSLength	Called before **DBCStoMF** when the ASCII string is SQL syntax.
DBCStoPC	Called for each field converted from EBCDIC to ASCII.

DLL Function	Description
DBCStoMF	Called for each field converted from ASCII to EBCDIC.
DBCSEnd	Called before unloading the DLL.

Chapter 12: 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 appendix 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 about 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.

This section contains the following topics:

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 about SQL conformance, see the chapter "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 about 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:

CA IDMS Data Type	JDBC Data Type
SMALLINT	SMALLINT
INTEGER	INTEGER
LONGINT	BIGINT
REAL	REAL
FLOAT	REAL (Precision < 25).
FLOAT	FLOAT (Precision > 24).
DOUBLE PRECISION	DOUBLE
DECIMAL	DECIMAL
UNSIGNED DECIMAL	DECIMAL
NUMERIC	NUMERIC
UNSIGNED NUMERIC	NUMERIC
CHAR	CHAR
GRAPHIC	CHAR (DBCS must be enabled)

CA IDMS Data Type	JDBC Data Type
VARCHAR	VARCHAR
VARGRAPHIC	VARCHAR (DBCS must be enabled)
BINARY	BINARY
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP

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.

JDBC Data Type	CA IDMS Data Type
BIT	SMALLINT
TINYINT	SMALLINT
SMALLINT	SMALLINT
INTEGER	INTEGER
BIGINT	LONGINT
REAL	REAL
FLOAT	DOUBLE PRECISION
DOUBLE	DOUBLE PRECISION
DECIMAL	DECIMAL
NUMERIC	NUMERIC
CHAR	CHAR
VARCHAR	VARCHAR
LONGVARCHAR	VARCHAR
BINARY	BINARY
VARBINARY	BINARY
LONGVARBINARY	BINARY
DATE	DATE
TIME	TIME

JDBC Data Type	CA IDMS Data Type
TIMESTAMP	TIMESTAMP

DriverManager

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.

jdbc:idms:ssl://*hostname*:*port/database*

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 chapter "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

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.reflection 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 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 about 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 about 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 manually 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

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.

The JDBC 4.0 Specification and detailed API documentation are available at http://java.sun.com.

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 setTransaction Isolations methods instead of executing these commands.

Batch Updates

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

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:

```
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.

```
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:

```
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 about 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.

Name	**Number**	**Type**
NUM_INPUT_PARMS	4	SMALLINT
NUM_OUTPUT_PARMS	5	SMALLINT
NUM_RESULT_SETS	6	SMALLINT

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
- CLOSE ALL RESULTS

Scrollable Result Sets

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.RowSet 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

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. 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:

Hibernate Property	Setting for IDMS Dialect
dialect	com.ca.idms.hibernate.IDMSDialect
connection.driver_class	ca.idms.jdbc.IdmsJdbcDriver

Sample Programs

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 subdirectories of the installation directory on Windows and z/OS. They are also provided in the file /java/samples.tar on the CA IDMS Server r16.1 CD.

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.

Appendix A: Windows Registry Information

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

This section contains the following topics:

Registry Information

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 keysare used by the ODBC Driver Manager and the CA IDMS Server drivers. HKEY_LOCAL_MACHINE contains information about 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 appendix describes the information in these sub-keys.

HKEY_LOCAL_MACHINE\Software\ODBC\ODBCINST.INI

This section contains information about 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

Subkey	Value Name	Description
ODBC Core	UsageCount	Driver manager usage count
ODBC Drivers	CA IDMS	Each installed ODBC driver has an entry. Value name is the driver name. Value data is installed.
CA IDMS		Each installed driver has a sub-key, whose name is the name of the driver.
	APILevel	Driver ODBC API conformance level.
	ConnectFunctions	Connect functions supported by driver.
	Driver	Driver DLL name and path.
	DriverODBCVer	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 about 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.

Subkey	Value Name	Description
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.
DefaultSchema		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.

Value Name	Value
AccessibleTables	0|1
AccountPrompt	0|1
CacheSQLTables	0|1
CatalogTable	*view_name*
CallSelect	0|1
CloseCommit	0|1 (ODBC Driver)
CloseCommit	0|1 (JDBC Driver)
CommitBehavior	0|1|2
ConnectSuspend	0|1
DefaultParmType	*integer_value*

Value Name	Value
DefaultSchema	*schema_name*
DescribeExtended	0|1
EnableEnsure	0|1
FetchDouble	0|1
FetchRows	*integer_value*
FetchSize	*integer_value*
FetchSuspend	0|1
FetchSuspendClose	0|1
IgnoreDTC	0|1
InvalidDecimal	*integer_value*
LoginTimeout	*integer_value*
PoolSuspendActive	0|1
PreservePrepared	0|1
QueryTimeout	*integer_value*
ReadOnly	0|1
SuspendStrategy	0|1|2|3
TxnIsolation	1|2
WaitTimeOut	*integer_value*

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 appendix "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 to 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 Chapter 3, "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:

Value Name	Value
AccessType	I|C
CciServerName	*cci_name*
CciServerPort	*integer_value*
Host	*host_name*
Port	*integer_value*
Resource	*node_name*
SSL	0|1
WaitTimeOut	*integer_value*

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=*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.

Value Name	**Value**
AlternateTask	*task_code*
AsciiEbcdicTables	*translation_table_name*
BufferLength	*integer_value*
ExternalWait	*integer_value*
Node	*via_node_name*
ResourceInterval	*integer_value*
Version	0|1

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 about 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 about TASK and PTERM System Generation statements see the CA IDMS Server System Generation guide.

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.

Value Name	Value
cadcdc32.dll	*dll_name*
DbcsPath	*dbcs_path*
DbcsType	*dbcs_type*
JcliTraceWs	0|1

Value Name	Value
JdbcTraceID	0|1
LogFile	*log_file_name*
LogFileCount	*integer_value*
LogFileSize	*integer_value*
LoginTimeout	*integer_value*
LogOptions	*log_option_values*
MultiThread	0|1
Path	*path_name*
QueryTimeout	*integer_value*
SSLCertDir	*certificate_directory_name*
SSLClientCert	*client_certificate_file_name*
SSLPassword	*client_certificate_password (encrypted)*
SSLServerCert	*server_certificate_file_name*
XxxxTrace	
CmTrace	*integer_value*
DnsTrace	*integer_value*
DtsTrace	*integer_value*
FdeTrace	*integer_value*
JcliTrace	*integer_value*
JdbcTrace	*integer_value*
OdbcTrace	*integer_value*
SQLTrace	*integer_value*
UtilTrace	*integer_value*
WaitTimeOut	*integer_value*

Values

cadcdc32.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. DbcsType 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.

SSLServer Cert=*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
  - 0x0200 // Snap connection block
  - 0x0400 // Snap statement block
  - 0x0800 // Snap SQLDA
- SqlTrace (IDMSQCLI.DLL):
  - 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
  - 0x0400 // Snap SQL output SQLDA
  - 0x0800 // Snap SQL syntax string
  - 0x4000 // Trace server calls
  - 0x8000 // Snap server interface blocks

- UtilTrace (IDMSUTIL.DLL):
  - 0x0001 // Trace external calls
  - 0x0002 // Trace internal calls
  - 0x0004 // Trace DllEntry calls

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:

Value Name	**Value**
Backlog	*integer_value*
ClientAuth	0|1
Encoding	*character _encoding_name*
Host	*host_name*
LogLevel	*integer_value*
LogTrace	*integer_value*
Port	*integer_value*
RemoteHost	*host_name*
RemotePort	*integer_value*
RemoteSSL	0|1
ReplyTimeOut	*integer_value*
Snap	0|1
SocketTimeOut	*integer_value*
SSL	0|1
Trace	0|1
Unicode	0|1
WaitTimeOut	*integer_value*

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 about 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 about 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.

Appendix B: 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 appendix "Properties File Information."

This section contains the following topics:

Configuration Information

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 nativeclient 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.

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 Chapter 3, "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 about resource limits for external user sessions, see the CA IDMS System Generation and CA IDMS System Operations guides.

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=*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=*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=*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=*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=*integer_value*

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

WaitTimeOut=*integer_value*

Specifies the default system reply timeout for the JDBC driver.

XxxxTrace=*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
  - 0x0400—Snap SQL output SQLDA
  - 0x0800—Snap SQL syntax string
  - 0x4000—Trace server calls
  - 0x8000—Snap server interface blocks
- UtilTrace (libutil.so):
  - 0x0001—Trace external calls
  - 0x0002—Trace internal calls

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*xxxxx*.class and CharToByte*xxxxxx*.class, where *xxxxxx* 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.

Appendix C: 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.

This section contains the following topics:

Setting CA IDMS Server Options as Properties

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:

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

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

```
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 about 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 about 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 about 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 about 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.

Index