

CA-IDMS Server[®]

User Guide

43



Computer Associates™

This documentation and related computer software program (hereinafter referred to as the "Documentation") is for the end user's informational purposes only and is subject to change or withdrawal by Computer Associates International, Inc. ("CA") at any time.

This documentation may not be copied, transferred, reproduced, disclosed or duplicated, in whole or in part, without the prior written consent of CA. This documentation is proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, the user may print a reasonable number of copies of this documentation for its own internal use, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the confidentiality provisions of the license for the software of the user will have access to such copies.

This right to print copies is limited to the period during which the license for the product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to return to CA the reproduced copies or to certify to CA that same have been destroyed.

To the extent permitted by applicable law, CA provides this documentation "as is" without warranty of any kind, including without limitation, any implied warranties of merchantability, fitness for a particular purpose or noninfringement. In no event will CA be liable to the end user or any third party for any loss or damage, direct or indirect, from the use of this documentation, including without limitation, lost profits, business interruption, goodwill, or lost data, even if CA is expressly advised of such loss or damage.

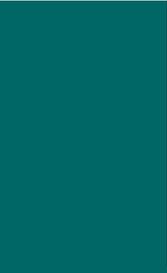
The use of any product referenced in this documentation and this documentation is governed by the end user's applicable license agreement.

The manufacturer of this documentation is Computer Associates International, Inc.

Provided with "Restricted Rights" as set forth in 48 C.F.R. Section 12.212, 48 C.F.R. Sections 52.227-19(c)(1) and (2) or DFARS Section 252.227.7013(c)(1)(ii) or applicable successor provisions.

© 2001 Computer Associates International, Inc., One Computer Associates Plaza, Islandia, New York 11749. All rights reserved.

All trademarks, trade names, service marks, or logos referenced herein belong to their respective companies.



Contents

Chapter 1: Introducing CA-IDMS Server

Who Should Use this Document	1-1
Components of CA-IDMS Server	1-2
The ODBC and JDBC Drivers	1-2
CA-IDMS Server Architecture	1-4

Chapter 2: Preparing to Install CA-IDMS Server

Windows Software Prerequisites	2-1
About CAICCI/PC	2-1
About the ODBC Driver Manager	2-2
About the Java Virtual Machine	2-3
Delivery of Components	2-4
Mainframe Software Prerequisites	2-5
OS/390 Software Prerequisites	2-5

Chapter 3: Setting Up Your CA-IDMS System

Installing the Host Component	3-1
Setting Up CA-IDMS Server	3-2
Defining the CA-IDMS System	3-2
Defining a CAICCI Line	3-3
Creating the CASERVER Task	3-4

Setting Up Database Access	3-5
Setting Up SQL Access	3-5
Utilizing Page Groups	3-6
Setting Up SQL Access to Non-SQL Databases	3-6
Setting up Catalog Views	3-10
Handling Invalid Numeric Data	3-11
Pseudo-conversational Processing	3-12
Tuning Pseudo Conversational Processing	3-13

Chapter 4: Installing the Client on Windows

Preparing to Install CA-IDMS Server	4-1
Uninstalling Previous Versions	4-1
Installing CA-IDMS Server on Windows	4-2

Chapter 5: Configuring the Client on Windows

Configuring CAICCI/PC	5-1
Defining Data Sources	5-2
Data Source Types	5-2
Adding a New Data Source	5-2
Saving the Data Source Definition	5-5
Testing the Data Source Definition	5-5
Editing the Data Source Definition	5-5
Setting Up a Server	5-6
Setting ODBC Options	5-8
Performance Considerations for ODBC Options	5-11
Specifying CA-IDMS Administrator Defaults	5-11
Logging Errors and Trace Information	5-13
Setting Language Options	5-14
Using the International Tab	5-14
Using a Custom Conversion DLL	5-21
Enabling a Custom Conversion DLL	5-21
Developing a Custom Conversion DLL	5-22
Configuring the JDBC Server	5-28

Chapter 6: Using the ODBC Driver on Windows

Connecting to a Predefined Data Source	6-1
Connecting Dynamically to a Data Source Not Previously Defined	6-2

Chapter 7: Using the CA-IDMS JDBC Server on Windows

Installing the JDBC Server	7-1
Configuring the Web Server for Applets	7-2
Using the JDBC Server on Windows NT	7-2
Using the JDBC Server on Windows 98	7-2

Chapter 8: Installing the Client on OS/390

Installing the Client Components for UNIX System Services	8-2
Step 1: Load the Installation Files	8-2
Step 2: Customize the Installation Files	8-2
Step 3: Upload OS/390 Datasets	8-4
Step 4: Customize the Installation JCL	8-4
Step 5: Set Up a New OMVS Group and User	8-6
Step 6: Allocate the HFS	8-7
Step 7: Create the Installation Directory in the HFS	8-7
Step 8: Create Subdirectories, Allocate Datasets, and Prelink Object Modules	8-8
Step 9: Install the Executable Modules Using SMP/E	8-8
Step 10: Set File Access Bits	8-9
Step 11: Delete Unnecessary Files	8-9

Chapter 9: Configuring the Client on OS/390

Configuring CA-IDMS	9-1
Specifying Environment Variables	9-2
Editing the Configuration File	9-3

Chapter 10: Using the Client on OS/390

Configuring Applications to Use CA-IDMS Server	10-1
Configuring the Web Server to Use CA-IDMS Server	10-2
Controlling the JDBC Server	10-3
Monitoring the JDBC Server	10-4

Chapter 11: Using the Client on Other Platforms

Using the JDBC Driver on Other Platforms	11-1
Using the JDBC Server on Other Platforms	11-2

Appendix A: ODBC Programmer Reference

Debugging User Sessions	A-1
Error Messages	A-1
ODBC Conformance Levels	A-2
API Conformance Levels	A-2
SQL Conformance Levels	A-5
Database Type Mapping Between ODBC and CA-IDMS	A-8
CA-IDMS to ODBC Data Type Mapping	A-9
ODBC to CA-IDMS Data Type Mapping	A-10
Driver-Specific Data Types	A-11
SQLDriverConnect Connection String Format	A-11
Supported Attribute Keywords and Attribute Values	A-12
Driver-Specific Connect Options	A-13
Supported Isolation and Lock Levels	A-13
Bulk Insert Support	A-13
Retrieving Network Set Information	A-14

Appendix B: JDBC Programmer Reference

JDBC Conformance	B-1
API Conformance	B-2
SQL Conformance	B-3
Database Type Mapping Between JDBC and CA-IDMS	B-4
CA-IDMS to JDBC Data Type Mapping	B-4
Connection Parameters	B-6
IDMS URL Format	B-6
DriverPropertyInfo	B-7
Dynamic Positioned Updates	B-8
Sample Programs	B-8
IdmsJcf	B-8
IdmsExample	B-9

Appendix C: Windows Registry Information

Registry Information	C-1
HKEY_LOCAL_MACHINE\Software\ODBC\ODBCINST.INI	C-2
HKEY_LOCAL_MACHINE\Software\ODBC\ODBC.INI	C-3
HKEY_CURRENT_USER\Software\ODBC\ODBC.INI	C-4
HKEY_LOCAL_MACHINE\Software\ComputerAssociates\CA-IDMS	C-4
Values	C-13

Appendix D: Configuration File Information

Configuration File Data	D-1
Environment Variables	D-1
Sections	D-2

Appendix E: Passing Accounting Information to CA-IDMS

Supplying Accounting Information	E-1
Using Accounting Information	E-2

Appendix F: Configuring CAICCI for TCP/IP

Installation Notes	F-1
Specifying Protocol Parameters for TCP/IP	F-2
Tracing a Communications Problem	F-4
Testing the Configuration	F-5
Exiting the CAICCI-PC Properties Dialog	F-6

Index

Introducing CA-IDMS Server

CA-IDMS Server provides open access to data stored in CA-IDMS databases, allowing you to maintain existing corporate databases and make your data available to new client-server and web-based applications. CA-IDMS Server provides support for dynamic Structured Query Language (SQL) using both the Open Data Base Connectivity (ODBC) and Java Data Base Connectivity (JDBC) application program interfaces.

CA-IDMS Server 4.3 supports ODBC on all 32-bit versions of Windows, and supports JDBC on all platforms supporting Java 1.1 or later, including Windows and OS/390.

Included on the CD is CA-IDMS Server 3.0, which runs on Windows 3.1 and provides support for ODBC applications as well as compiled SQL embedded in CA-Visual Realia COBOL applications. CA-IDMS Server 3.0 must be used in the Windows 3.1 environment.

Who Should Use this Document

This guide assumes you are a new user of CA-IDMS Server with experience using either Microsoft Windows or OS/390, and have familiarity with CA-IDMS and database access.

Use this document if you are a CA-IDMS 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 ODBC and JDBC Application Program Interface (API) requests are implemented in CA-IDMS Server.

Components of CA-IDMS Server

CA-IDMS Server consists of both host and client components.

The host component is installed on your CA-IDMS system and enables the Common Communications Interface (CAICCI) line driver to communicate with the client components. CAICCI services are installed on the same operating system image as your CA-IDMS system. CAICCI is a component of Unicenter TNG Framework for OS/390, CA-Common Infrastructure Services (CA-CIS), or CA90s Services, depending on your operating system.

The client components include the CA-IDMS ODBC Driver, the CA-IDMS JDBC Driver, the CA-IDMS JDBC Server, the native CA-IDMS 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.

The 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 CA-ADS. It communicates with CA-IDMS using CAICCI. Supported protocols include TCP/IP and LU2 for Windows clients and Cross Memory Services for OS/390 clients.

The CA-IDMS ODBC Driver

The CA-IDMS ODBC Driver can be used on all Windows platforms (for Windows 3.1 use CA-IDMS Server 3.0, included on the CD). The ODBC driver implements the ODBC 2.5 specification and works with ODBC 3.X applications when the ODBC 3.0 (or later) Driver Manager is installed.

The CA-IDMS ODBC Driver always calls the native interface directly. Applications using the ODBC Driver run on the machine on which both CA-IDMS Server and CAICCI/PC are installed. Web-based applications running in browsers use ODBC indirectly through technologies, such as Active Server Pages, which appear as the application to the ODBC Driver.

The CA-IDMS JDBC Driver

The CA-IDMS JDBC Driver can be used on Java 1.1 (or later) platforms, including Windows and OS/390. The JDBC Driver implements the JDBC 1.2 specification, but it also can be used with JDBC 2.0 applications that do not call JDBC 2.0 specific methods.

The CA-IDMS JDBC Driver calls the native interface directly when both are installed on the same platform. Applications using the JDBC Driver can run on the same machine on which both CA-IDMS Server and CAICCI are installed. Web based applications running in browsers use JDBC indirectly, through technologies such as Java Servlets or Java Server Pages, which appear as the application to the JDBC Driver.

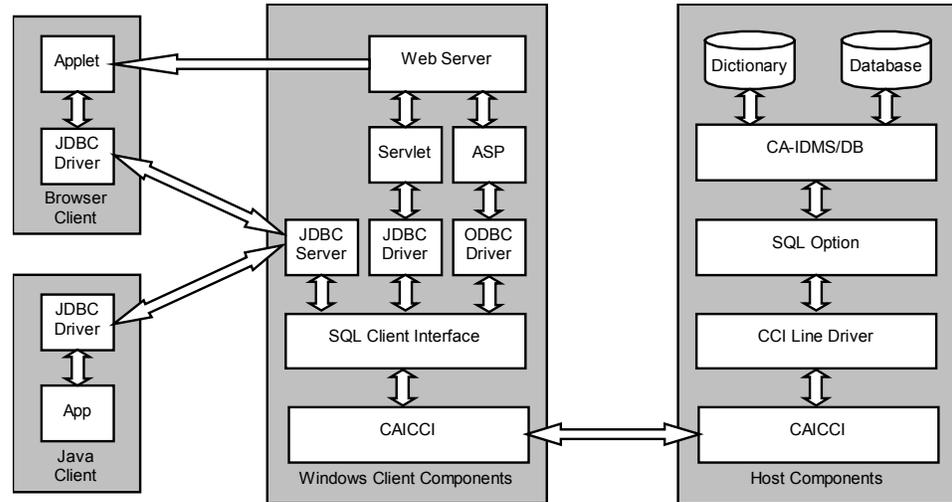
The CA-IDMS JDBC Server

The CA-IDMS JDBC Server acts as an intermediate server between the JDBC Driver and the CA-IDMS system, and allows applications to use JDBC directly when running on platforms on which the native SQL client interface is not installed. The JDBC Driver uses TCP/IP to communicate with the JDBC Server, and the JDBC Server invokes the native interface on behalf of the JDBC Driver. The JDBC Server is often used with a web server to support JDBC applets, but can also be used to run standalone JDBC applications.

The JDBC Server is not used when the application and the native interface run on the same platform. It is only needed when the application or applet runs on a platform on which the native interface is not installed.

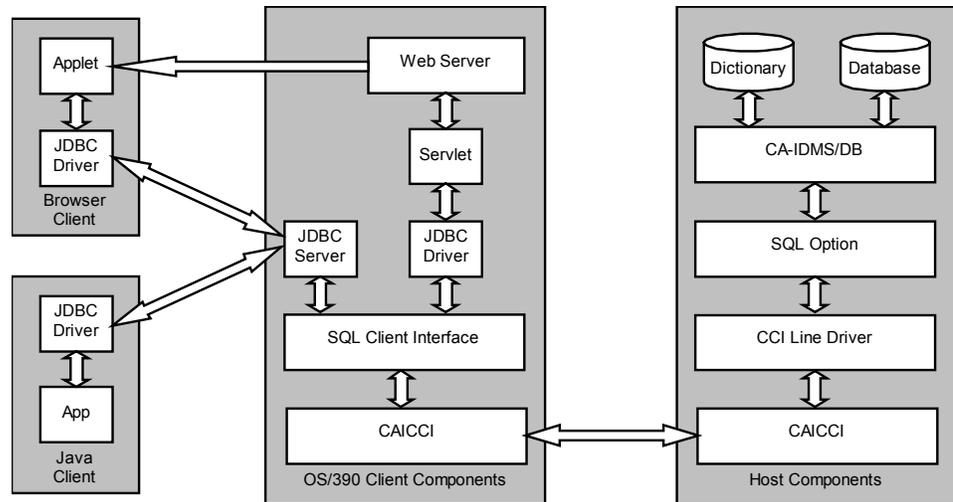
CA-IDMS Server Architecture

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 either TCP/IP or LU2. 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, such as CA-Visual Express.

The following diagram illustrates how the CA-IDMS Server software components fit together when the client platform is OS/390:



The CA-IDMS JDBC Driver is supported for OS/390 applications, represented here by the web server. The OS/390 client and CA-IDMS communicate using Cross Memory Services.

The host components are the same whether the native client is installed on Windows or OS/390.

For More Information

You may find it useful to refer to the following documents when setting up your CA-IDMS system to work with CA-IDMS Server:

- *CA-IDMS Installation and Maintenance – OS/390*
- *CA-IDMS System Generation*
- *CA-IDMS System Operations*
- *CA-IDMS Database Administration*
- *CA-IDMS SQL Reference*
- *CA-IDMS SQL Programming Guide*
- *CA-IDMS DML Reference — COBOL*
- Unicenter TNG Framework for OS/390, CA-CIS, or CA90s Services documentation

Information about ODBC is available at www.microsoft.com. Information about JDBC is available at www.java.sun.com.

Preparing to Install CA-IDMS Server

This chapter summarizes the software prerequisites for CA-IDMS Server on Windows and the mainframe.

Windows Software Prerequisites

CA-IDMS Server requires the following software to be installed on the PC:

- Microsoft Windows 95 or 98, Windows NT 4.0 (Service Pack Release 3 or higher), Windows Millennium, or Windows 2000.
- TCP/IP or LU2 communications protocols
- CAICCI/PC
- ODBC Driver Manager 2.5, or later
- Java 1.1 or later virtual machine for JDBC support

About CAICCI/PC

CAICCI/PC provides a common interface between the CA-IDMS ODBC Driver and various communications protocols. CAICCI/PC is distributed on the Unicenter TNG Framework for OS/390, CA-CIS, or CA90s Services tape for your mainframe operating system. The CAICCI/PC runtime interface for TCP/IP can be installed as part of the CA-IDMS Server installation.

CAICCI/PC must be installed to use ODBC, the CA-IDMS JDBC Server, and JDBC applications that connect directly to CA-IDMS. It is not necessary to install CAICCI/PC on client workstations that will use JDBC only through applets running in a Web browser. Additionally, it is not necessary to reinstall CAICCI/PC when upgrading from CA-IDMS Server Versions 4.0 or later.

The multi-threaded version of CAICCI/PC for TCP/IP can be installed as part of the CA-IDMS Server installation. Only the run-time DLL is installed. For your convenience, the complete CAICCI/PC installation is included in the CCI directory on the CA-IDMS Server CD. Copy the file **ccipcw32.exe** to a directory on your hard disk, and run it to extract the installation files. Then run the extracted **install.exe**.

For additional information about CAICCI/PC for TCP/IP, see the appendix “[Configuring CAICCI for TCP/IP](#)” in this guide. For information about downloading and configuring CAICCI/PC and supported network transport products, refer to Unicenter TNG Framework for OS/390, CA-CIS, or CA90s Services documentation, depending on your operating system.

About the ODBC Driver Manager

The ODBC Driver Manager provides the link between ODBC enabled applications and ODBC drivers, and is installed automatically by newer releases of many Microsoft products, including Windows NT, Office, and Internet Explorer. CA-IDMS Server conforms to the ODBC 2.5 specification and will work with the ODBC 2.5, or later, Driver Manager. To verify the version of ODBC installed on your machine, click the ODBC 32 icon in the Windows Control Panel and select the About tab. For detailed information about CA-IDMS Server conformance to the ODBC standard, refer to the “[ODBC Programmer Reference](#)” appendix in this guide.

The ODBC Driver Manager must be installed before installing CA-IDMS Server to use ODBC, and is recommended for use with the CA-IDMS JDBC Server and JDBC applications that connect directly to CA-IDMS. It is not necessary to install it on client workstations that will only use JDBC through applets running in a Web browser. Additionally, it is not necessary to reinstall the ODBC Driver Manager when upgrading from CA-IDMS Server Versions 4.0 or later.

With the release of ODBC 3.5, Microsoft has embedded the ODBC Driver Manager into 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. Since the ODBC Driver Manager components can no longer be redistributed separately, they are not installed as part of the CA-IDMS Server installation. Additional information about ODBC and MDAC is available from the Microsoft web site.

The latest version of MDAC can be freely downloaded directly from the Microsoft web site, www.microsoft.com. For your convenience, the typical MDAC installation, **mdac_typ.exe**, is included in the Microsoft directory on the CA-IDMS Server CD. This file can be run directly from the CD.

CA-IDMS Server supports 16-bit applications, using the 16-bit ODBC Driver Manager to route the API requests to the 32-bit ODBC driver. This Driver Manager can be installed using CA-IDMS Server 3.0, which has an option to install the Driver Manager only. For your convenience, the CA-IDMS Server 3.0 installation disk image, **idmsrv30.zip**, is included in the Serv30 directory on the CA-IDMS Server CD. This file must be unzipped onto a writable disk before the installation is run.

About the Java Virtual Machine

A Java Virtual Machine (JVM) is an interpreter that executes Java programs, which are stored on disk as class files. CA-IDMS Server conforms to the JDBC 1.2 specification and requires a Java 1.1 (or later) compliant Virtual Machine. For detailed information about CA-IDMS Server conformance to the JDBC standard, refer to the “[JDBC Programmer Reference](#)” appendix in this guide. Additional information about JDBC and Java is available on the World Wide Web at www.java.sun.com.

Recommended JVMs for CA-IDMS Server are the JVM installed as part of the Microsoft Internet Explorer 4.01, Service Pack 1 (or later), and the Sun Microsystems Java Run Time Environment (JRE) 1.1.8 or later. You can verify the version of the Microsoft JVM, if installed, by entering “jview” at the command prompt. The commands used to verify the version of the Sun JVM vary depending on the version, and on whether the JRE or JDK is installed, but are generally either “jre” or “java version”.

The latest version of the Microsoft JVM can be freely downloaded directly from the Microsoft web site, www.microsoft.com. For your convenience, the JVM 5.0 installation, **msjavx86.exe**, is included in the Microsoft directory on the CA-IDMS Server CD, and can be run directly from the CD.

The latest versions of the Sun Microsystems JRE can be freely downloaded directly from the JavaSoft web site, www.java.sun.com. For your convenience, the JRE 1.1.8 and 1.3 installation files are included in the Sun directory on the CA-IDMS Server CD. These files can be run directly from the CD.

The CA-IDMS Java Command Facility (JCF) demo applet uses the Swing classes from Sun Microsystems. They are not included with the Microsoft JVM or the Sun JRE 1.1, but can be downloaded from Sun's web site. They are installed as part of the Sun JRE 1.3. When the JCF demo is installed, the **swingall.jar** file containing these classes is installed in the CA-IDMS Server\Java\classes directory.

Delivery of Components

Delivery of the Computer Associates software components utilized by CA-IDMS Server is shown in the chart below:

Software Component	Base Product	Installation Medium
CA-IDMS Server ODBC and JDBC Drivers	CA-IDMS Server	CA-IDMS Server CD
CAICCI/PC	Unicenter TNG Framework for OS/390, CA-CIS, or CA90s Services	Base product tape or CA-IDMS Server CD.
CAICCI and CAIENF	Unicenter TNG Framework for OS/390, CA-CIS, or CA90s Services	Base product tape
CA-IDMS	CA-IDMS	CA-IDMS
CA-IDMS Server (host component)	CA-IDMS Server	CA-IDMS tape

Mainframe Software Prerequisites

CA-IDMS Server requires the installation of CA-IDMS Release 12.0 (or later) on the mainframe. You will also need the SQL option. The host component of CA-IDMS Server is delivered on the CA-IDMS installation tape for Release 14.0 and higher, and on the CA-Host Server tape for prior releases. APAR LO88882 should be applied to Release 14.0 and 14.1 on MVS to update the host component. The updated host component is also included in the Service Pack 4 tape for these releases, and the Release 15.0 tape. Refer to the *CA-IDMS Installation and Maintenance Guide* for more information about installing CA-IDMS Server on the mainframe.

APAR LO09385 should be applied to CA-IDMS 12.01 systems with gen level 9506 or later. This corrects an infinite loop than can occur in the CV when a command with a "piggybacked suspend" (CA-IDMS Server automatically pseudo converses) is rolled back due an error. This problem was fixed in CA-IDMS Server 14.0.

CA-IDMS Server requires the installation of the Unicenter TNG Framework for OS/390, CA-CIS, or CA90s Services components CAIENF (Event Notification Facility) and CAICCI (Common Communications Interface) on the mainframe. For information about installing CAIENF and CAICCI on the mainframe, refer to Unicenter TNG Framework for OS/390, CA-CIS, or CA90s Services documentation.

OS/390 Software Prerequisites

The CA-IDMS JDBC Driver and JDBC Server run in the UNIX System Services (USS) environment on OS/390. They must be installed and set up on the mainframe. The JVM must also be installed and available in the USS environment. TCP/IP must be installed and configured correctly to use the JDBC Server. For additional information about installing USS, TCP/IP, and the JVM, refer to IBM documentation.

Setting Up Your CA-IDMS 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. The chapter also includes a description of how network records appear to SQL, as well as information about Numeric data, and pseudo conversational processing.

Installing the Host Component

CA-IDMS Server includes a host component, installed on all mainframe operating systems, that allows encrypted passwords to be sent from the client to the CA-IDMS System. This host component is compatible with CA-IDMS 12.0 and all later releases, as well as all releases of CA-IDMS Server. The host component must be installed on each secured CA-IDMS system.

Attempts to connect to a secured CA-IDMS system with the earlier version of the host component installed will fail when password encryption is enabled.

The host component consists of an object module, RHDCD0LB, linked with the CAICCI line driver, RHDCD0LV. The object module is delivered on the CA-IDMS tape for Release 15.0, and is available on service packs for Release 14.0 and 14.1.

Setting Up CA-IDMS Server

To set up CA-IDMS Server access on the mainframe, each CA-IDMS system to be accessed as a CA-IDMS data source server must be generated with the following definitions:

- A CAICCI line
- A PTERM/LTERM pair for each concurrent session with the CA-IDMS system
- A CA-IDMS Server task
- The SQL definitions in the catalog area of the dictionary associated with the CA-IDMS system

Defining the CA-IDMS System

The following section describes the procedure to define the CA-IDMS system-generation parameters to support communications using CA-IDMS Server. Include these definitions in each CA-IDMS system to be accessed from a CA-IDMS data source.

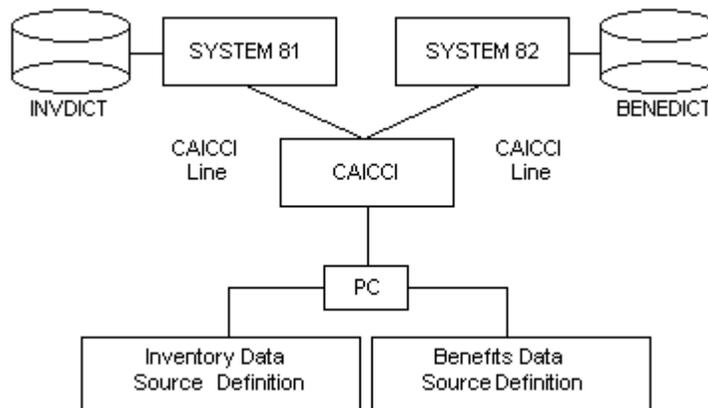
For detailed information about these system-generation statements and how to use the system-generation compiler, refer to *CA-IDMS System Generation*. Additionally, *CA-IDMS System Operations* provides information about configuring and maintaining CA-IDMS systems.

Examples

The examples in this chapter assume that CA-IDMS Server will be 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.

The following diagram illustrates the sample CA-IDMS system network:



Defining a CAICCI Line

CAICCI provides communication between mainframes, or between a mainframe and PC, independent of any particular communications protocol or access method. A CAICCI line connects a CA-IDMS system with the CAICCI network. Define a CAICCI line in each CA-IDMS system to be accessed as a CA-IDMS data source.

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

Each client process uses at least one PTERM/LTERM pair when connected to a CA-IDMS system. Define a minimum of one PTERM/LTERM pair for each PC to be used concurrently to access the CA-IDMS system. If more than one application will be used on the same PC to access the CA-IDMS system, you must define additional PTERM/LTERM pairs for each additional application. For example, if 5 PC users each run 2 ODBC applications concurrently, define 10 PTERM/LTERM pairs.

When multi-threading is enabled, 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 wish 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. Define a CASERVER task code for each CA-IDMS system to be accessed by CA-IDMS Server. To do this, perform the following steps:

1. From the system-generation compiler, enter the 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.

Setting Up Database Access

CA-IDMS Server uses 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 CA-IDMS system. 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 Data Base Administrator (DBA) for assistance.
- To limit the size of the list of tables returned to an ODBC application, create an Accessible Tables View that returns a subset of the default view, and set it as an option for specific data source. Using such a view of accessible tables can generate a more meaningful list of tables for each user and improve performance.

Note: Since registry settings are generally not available to Java applets, the JDBC driver always uses the default view, SYSCA.ACCESSIBLE_TABLES.

- 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 14.1 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 by CA-IDMS Release 14.1, 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.

Refer to the *CA-IDMS Database Administration — Volume 1* or the *CA-IDMS 14.1 Features Guide* for more information about using page groups.

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";
```

For more information about defining SQL schemas, see the *CA-IDMS SQL Reference* for syntax and information about accessing non-SQL databases and *CA-IDMS Database Administration* 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. To access 123-ORD-NUM, for example, 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. However, 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.

Tip: 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.

In addition, 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.

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 and SQLSpecialColumns 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 and getBestRowIdentifier methods. This table procedure is required for JDBC support.

Gen level 9506 or later of CA-IDMS 12.01, or later, is required to use this feature, and includes the DDL to define the table procedure (contained in VIEWDDL) and the table procedure load module (IDMSOINX), which is installed in the same library as the updated CA-IDMS nucleus modules. In most cases, the table procedure is already defined. If not, the DDL to define it is contained in IDMSOINX.DDL, which is installed in the CA-IDMS Server directory.

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 CA-IDMS Release 15.0 and later. For CA-IDMS Release 14.1 and earlier, the DDL contained in ACCVIEWS.DDL, installed in the CA-IDMS Server\Java directory, should be run against every catalog that will be accessed using JDBC.

Refer to the *CA-IDMS SQL Reference* for more information about accessing non-SQL-defined databases using SQL.

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, and may 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:** The default option. 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 an `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 will 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. It does this by issuing a SUSPEND command whenever a transaction ends. When autocommit is enabled, a transaction ends after the execution of every statement updating the database. When autocommit is disabled, the application explicitly ends a transaction using the ODBC SQLTransact function or the JDBC Connection.commit or Connection.rollback method. To optimize performance, the COMMIT and SUSPEND commands are “piggybacked” onto other requests to avoid additional network traffic whenever possible.

To minimize contention with other transactions, CA-IDMS Server also attempts to commit (and suspend) when a cursor is closed. It can do this only if the connection has no uncommitted updates (only possible when autocommit is disabled), no other open cursors, and no prepared statements. The COMMIT, SUSPEND, and CLOSE commands are actually piggybacked onto the FETCH command to minimize network traffic. The piggybacked CLOSE and COMMIT commands are executed only if the cursor reaches the end, that is, the FETCH returns SQLCODE = 100. The piggybacked SUSPEND command is always executed. The drivers also attempt to commit and suspend when the application explicitly closes an open cursor.

The SUSPEND command ends the CA-IDMS task unless a cursor is open to process a catalog function. To minimize contention with application tables, both the ODBC and JDBC drivers use a separate session to access data from the SQL catalog. The task can only end when a SUSPEND has been issued for both the “main” and “catalog” sessions. The drivers suspend the catalog session automatically when all cursors used to process catalog requests reach the end or are explicitly closed.

Tuning Pseudo Conversational Processing

The default options attempt to balance network traffic, CA-IDMS resource use, and contention with other transactions. You can set options in the Windows registry or OS/390 configuration file to tune pseudo-conversational processing, if appropriate for your application.

Since CA-IDMS Server suspends the session whenever it commits a transaction, you can disable autocommit to reduce the number of pseudo-converses. Your application must then explicitly commit the transaction at appropriate intervals. Use the ODBC SetConnectOption function or JDBC Connection.setAutoCommit method to control autocommit..

CA-IDMS Server attempts to commit and suspend when it closes a cursor, even when autocommit is disabled. You can use the CloseCommit option to prevent this. If your application uses a SELECT command to invoke a table procedure that updates the database, you must make sure that the table procedure commits any updates.

You can use the FetchSuspend option to prevent CA-IDMS Server from suspending the session after each FETCH. The default is disabled when the CA-IDMS System is 14.0 or earlier. The session is suspended when the cursor is closed unless both autocommit and CloseCommit are disabled. Note that this can cause slightly more network traffic when the result set is small, since the SUSPEND must sent as a separate command rather than piggybacked onto a FETCH.

These options are described in more detail in the “[Windows Registry Information](#)” and “[Configuration File Information](#)” appendices of this manual.

Using International Character Sets with JDBC

All character data is represented as Unicode in Java. Each JVM provides a set of converter classes that convert Unicode to and from specific character sets. The character set is identified by an **encoding** name and the converter class names are derived from the encoding. For example, Cp037 is the encoding that corresponds to the IBM EBCDIC code page 037, CharToByteCp037 is the name of the class that converts from Unicode to this variant of EBCDIC, and ByteToCharCp037 is the class that converts from EBCDIC to Unicode.

A JVM typically includes a small set of basic converter classes with the standard class libraries. These support conversion to the encodings used on the platform where the JVM runs. The Sun and IBM implementations also include a more extensive set of converter classes in the i18n.jar file.

The CA-IDMS JDBC Driver uses these built in converter classes to convert character data from the Unicode representation used by the Java VM to the EBCDIC encoding used by CA-IDMS. How this actually happens depends on the platform where the JDBC Driver runs and how it calls the native CA-IDMS interface.

The native SQL Client Interface, commonly called "QCLI", runs on mainframes, Windows, and OS/390 USS. All CA-IDMS applications use a version of it, including ADS dialogs, COBOL programs, OCF, IDMSBCF, and the CA-IDMS ODBC Driver. The CA-IDMS JDBC Driver can call QCLI directly on Windows and OS/390 USS, and via the CA-IDMS JDBC Server from all Java 1.1 (or later) platforms.

When the CA-IDMS JDBC Driver calls QCLI directly it converts Unicode data to the encoding used by QCLI, usually the "default" platform encoding for the machine. On OS/390 USS this is typically the same EBCDIC encoding used by CA-IDMS. On Windows, however, this is ASCII, which the CA-IDMS Server communications layer converts to EBCDIC, just as it does for the CA-IDMS ODBC Driver.

The default platform encoding is given by the `file.encoding` system property, and is usually determined by the JVM when it is installed or started. The default encoding is used whenever a Java program reads or writes character data in native format without explicitly specifying an encoding.

You can specify the encoding that the CA-IDMS JDBC Driver uses to convert data passed to QCLI by adding the `ca.jdbc.file.encoding` system property on the java command line, for example:

```
-Dca.jdbc.file.encoding=Cp037
```

This overrides the default encoding for the CA-IDMS JDBC Driver only and does not affect the encoding used by other classes.

When the CA-IDMS JDBC Driver connects to CA-IDMS via the CA-IDMS JDBC Server, QCLI is actually called by the JDBC Server. The JDBC Server is responsible for ensuring that Unicode data is converted to the native encoding. On OS/390 this is EBCDIC, while on Windows it is ASCII, which is converted by the communications layer. The `ca.jdbc.file.encoding` system property can be used to override the default encoding for the JDBC Server, although it is usually more convenient to specify the Encoding setting in the Proxy section of the OS/390 configuration file or the Windows registry.

To minimize mainframe resource usage the JDBC Server requests that the JDBC Driver convert data into the native CA-IDMS encoding. When the connection is first established, the JDBC Server passes the desired encoding name to the JDBC Driver. If the corresponding converter class is accessible to the JDBC Driver, it is used for all data exchanged with the JDBC Server, offloading most of the conversion. If the converter class is not accessible (as is often the case with the Microsoft JVM) data is exchanged with the JDBC Server in either UTF-8 or Unicode, as specified by the Unicode option in the Proxy section. The JDBC Server then converts the data to the specified encoding.

The Proxy options are described in more detail in the “[Windows Registry Information](#)” and “[Configuration File Information](#)” appendices of this manual. A list of supported character encodings is available at <http://java.sun.com/j2se/1.3/docs/guide/intl/encoding.doc.html>.

Installing the Client on Windows

This chapter describes the installation of CA-IDMS Server in Windows, and identifies the steps you must take before installing the product.

Preparing to Install CA-IDMS Server

Before you install CA-IDMS Server, the following must be installed:

- Network transport software, either LU2 or TCP/IP.
- CAICCI/PC
- ODBC Driver Manager
- Java VM 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 want to access.
- The node name of the system on which the dictionary resides.
- The task code defined for the CA-IDMS Server. The default task code is CASERVER.

On Windows NT and Windows 2000, you must use a user ID with administrative privileges when installing CA-IDMS Server and CAICCI/PC.

Uninstalling Previous Versions

Because there have been changes to the locations where certain files are stored, we recommend that you uninstall any previous versions of CA-IDMS Server to remove old items from the Start menu. For the most up-to-date information about uninstalling earlier versions of CA-IDMS Server, refer to the **readme.txt** file included on the CD. Do not remove your data source definitions.

Installing CA-IDMS Server on Windows

1. 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.
2. Insert the CA-IDMS Server CD into your CD-ROM drive. The CA-IDMS Server installation begins automatically. If it does not, right-click on the CD icon in the My Computer window and select Auto Start.
3. 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.
4. Choose an installation option from among the following:
 - Select Typical to install all components of CA-IDMS Server, including the ODBC driver, JDBC driver, JCF demo, and javadoc
 - Select Compact to install the ODBC driver only
 - Select Custom to choose which components to install

Choose Custom to copy the CAICCI/PC Dynamic Link Library (DLL) for TCP/IP into the Windows/SYSTEM directory along with the CA-IDMS Server DLLs. This ensures the use of the correct version of CAICCI for multi-threaded applications.

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

5. The CA-IDMS Server Installer updates the Windows registry with information required by the ODBC Driver Manager, and invokes a dialog that allows you to add data sources, defaults, or options defined for CA-IDMS Server 3.0 16-bit data sources to the registry.

You can add data sources either as User data sources, available only to the user currently signed on, or as System data sources, available to all users and services of the machine. Server definitions, defaults, and global options are always available to all users and services. Data sources used by Windows NT services or the CA-IDMS JDBC Server must be defined as system data sources. See the chapter "[Using the CA-IDMS JDBC Server on Windows](#)" in this guide for information about the JDBC Server.

6. The CA-IDMS Server Installer displays a dialog allowing you to run the ODBC Administrator to modify or add additional data source, server, and option definitions.
7. The CA-IDMS Server menu is added to the Start Menu.

For more information about defining data sources, refer to the chapter "[Configuring the Client on Windows](#)" in this guide.

Configuring the Client on Windows

A data source is the description of a database you want to access from an ODBC or JDBC application. The CA-IDMS JDBC driver can use ODBC data source definitions to access CA-IDMS databases. The JDBC driver does not use the ODBC driver at run time.

This chapter discusses the elements of a data source definition and how to use the CA-IDMS ODBC Administrator dialog to define and administer a data source definition.

CA-IDMS Server provides a context-sensitive online Help facility. Click the Help button displayed in any dialog to obtain Help about that dialog.

Configuring CAICCI/PC

You must configure CAICCI/PC to use either LU2 or TCP/IP. For TCP/IP, specify the host where the CCITCP job (the mainframe job supporting TCP connections for CCI) runs. You can use the CAICCI/PC Properties dialog to configure CAICCI with the same properties for all data sources. When CAICCI/PC is configured for TCP/IP, you can also use the CA-IDMS ODBC Administrator to configure CAICCI/PC with different properties for each data source.

If you install only the CAICCI/PC DLL as part of the CA-IDMS Server installation, you must use the CA-IDMS ODBC Administrator to configure CAICCI/PC, unless you have previously installed the complete CAICCI/PC product.

For information about setting CAICCI/PC options, refer to the appendix [“Configuring CAICCI for TCP/IP”](#) in this guide.

Defining Data Sources

The CA-IDMS ODBC Administrator, accessed from the Control Panel or CA-IDMS Server menu, allows you to define a connection between the Windows application and CA-IDMS. You can also use the CA-IDMS ODBC Administrator to define a Server, establish Administrator default settings, set language options, specify ODBC options, or configure the CA-IDMS JDBC Server.

Data Source Types

CA-IDMS allows you to define new data sources as either User data sources 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. Server definitions, defaults and global options are always available to all users and services. Typically, data sources meant to be used by Windows NT services must be defined as System data sources. For example, data sources must be defined as System data sources to be used by the CA-IDMS JDBC Server.

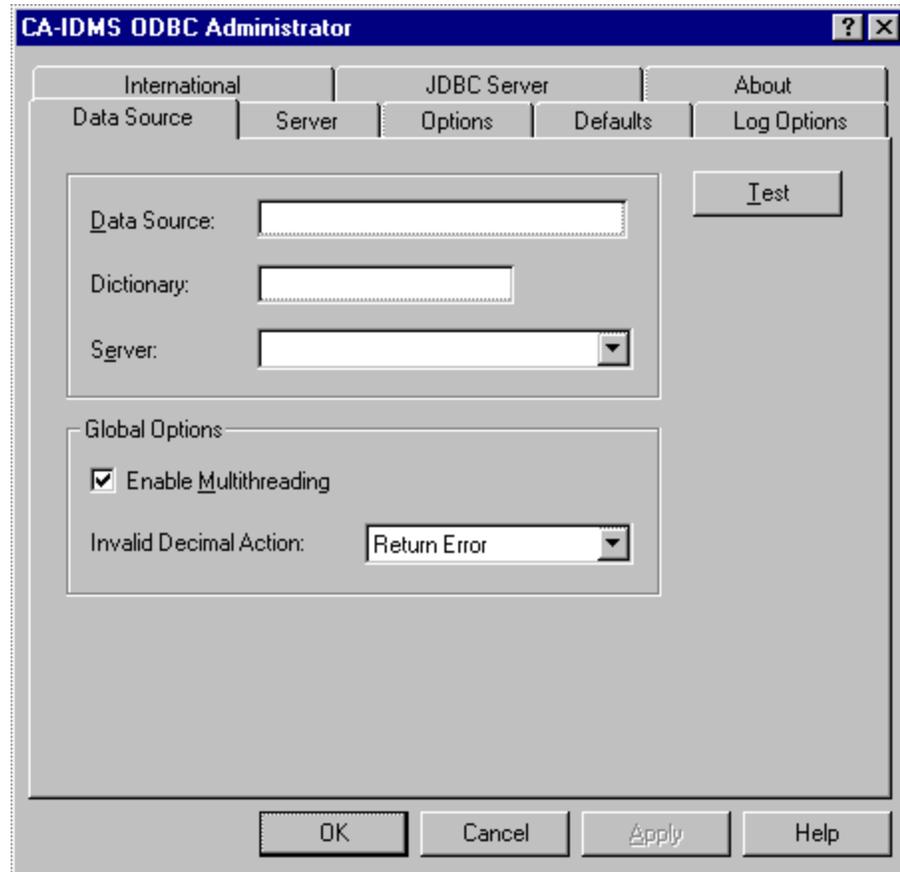
Adding a New Data Source

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 name of the CA-IDMS system 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 dialog from the Windows Control Panel double-click the Microsoft ODBC Administrator icon. To access this dialog from the CA-IDMS Server menu, select Start, Programs, CA-IDMS Server, 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, select either the User DSN or System DSN tab and click the Add button 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 ODBC Administrator dialog appears:



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

Data Source: Specify the name of the data source. 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.

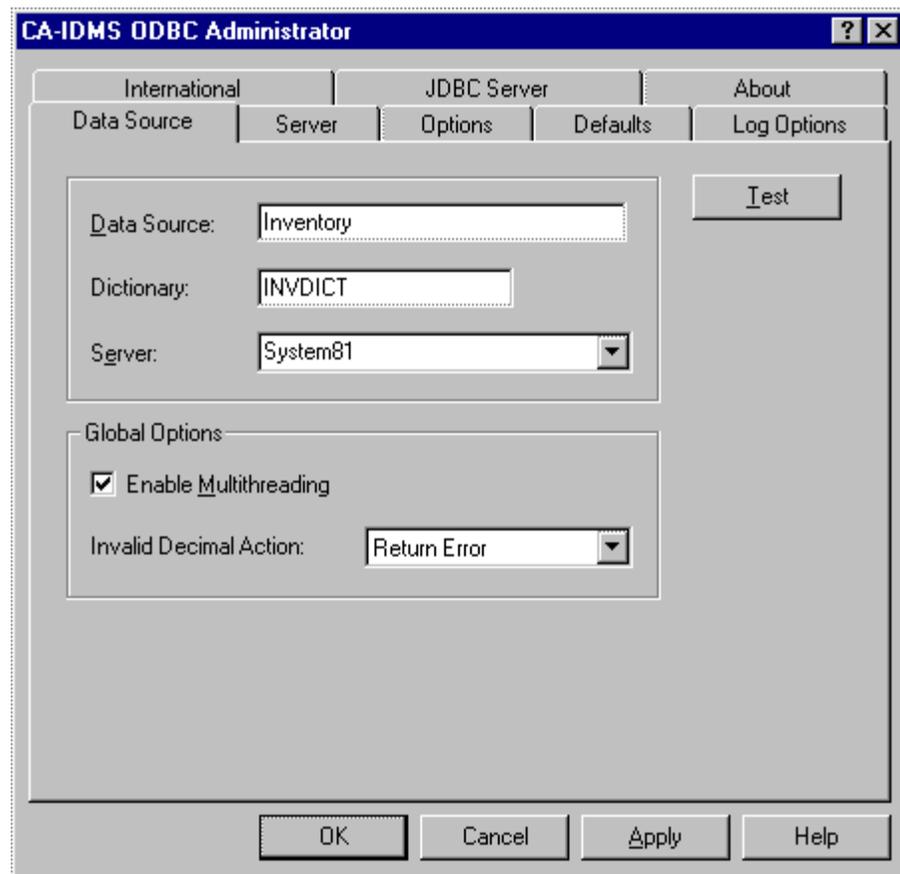
Dictionary: (Optional) Specify the DBNAME or segment name of the dictionary containing the definitions of the tables you want to access. This name must be defined in the DBNAME table on the CA-IDMS system identified by the server name. The default is the first eight characters of the DSN.

Server: Specify the name of the Server identifying the CA-IDMS node containing the dictionary that defines the tables you want to access. Enter a new name or select an existing name from the pull-down list. This field is required, and can be either the Node Name of the host CA-IDMS system or a 1- to 32-character logical name of a Server defining the Node Name and options.

Under Global Options, specify the following:

Enable Multithreading: Allows concurrent access to the CA-IDMS system by all connections in the same application. Note that this option affects all data source descriptions. This option is enabled by default.

Invalid Decimal Action: Specify how the ODBC and JDBC drivers handle invalid Decimal and Numeric data. Specify the return of Null, 0, or an error to the application. See the section [“Handling Invalid Numeric Data”](#) in the chapter [“Setting Up Your CA-IDMS System”](#) in this guide for more information regarding the available options.



Note: The examples in this chapter contain information based on the sample CA-IDMS system network illustrated in the chapter, [“Setting Up Your CA-IDMS System,”](#) in this guide. The values displayed are based on the system-generation statements defined for System 81 and System 82.

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 ODBC Administrator dialog, and return to the Data Sources dialog. To save the definition without closing the CA-IDMS 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 the Test button, to invoke the CA-IDMS Test Connect dialog:



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 associated with the data source named. 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 CA-IDMS ODBC driver.

Editing the Data Source Definition

Once you have saved your data source definition, you may 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 the Configure button to invoke the CA-IDMS ODBC Administrator dialog. Edit the information and click Apply or OK.

Setting Up a Server

To define a Server, which specifies the CA-IDMS system containing the databases, use the Server tab on the CA-IDMS ODBC Administrator dialog. You can also use this dialog to add or change a Server definition, or to override default CCI options for this Server.

The screenshot shows the 'CA-IDMS ODBC Administrator' dialog box with the 'Server' tab selected. The dialog has a title bar with a question mark and close button. Below the title bar are tabs for 'International', 'JDBC Server', and 'About'. Underneath are sub-tabs for 'Data Source', 'Server', 'Options', 'Defaults', and 'Log Options'. The 'Server' sub-tab is active, showing a 'Delete' button. The 'IDMS Options' section contains:

- Name:** System81
- Node Name:** SYST0081
- Via Node:** (empty)
- Task Code:** CASERVER
- MF Version:** Server 4.2 or earlier (dropdown menu)

 The 'CCI Options' section contains:

- Wait Timeout:** 0
- Server Name:** (empty)
- Server Port:** 0

 At the bottom are buttons for 'OK', 'Cancel', 'Apply', and 'Help'.

Under IDMS Options, specify the following:

Name: Displays the selected Server name, from the Data Source page. The name cannot be changed here.

Node Name: Specify 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.

Via Node: (Optional) Specify the node with which CAICCI establishes a connection. The system identified here must contain a RESOURCE table entry for the system identified by Node Name. Use this option when the system containing your tables does not communicate directly with CAICCI.

Task Code: Specify 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.

MF Version: Specify the version of the mainframe component installed on the CA-IDMS system. The default is 4.2 or Earlier. When the MF Version option is set to 4.3 or Later, all signons will be rejected unless the mainframe component included with CA-IDMS Server Version 4.3 is installed on that system.

To use the following CCI Options, the multi-threaded version of CAICCI must be installed, although multi-threading need not be enabled for the ODBC driver.

Wait Timeout: Specify 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 multi-threading 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
- Enter -1 to specify an indefinite wait (this is interpreted as the largest positive integer)
- Enter a specific time, in seconds

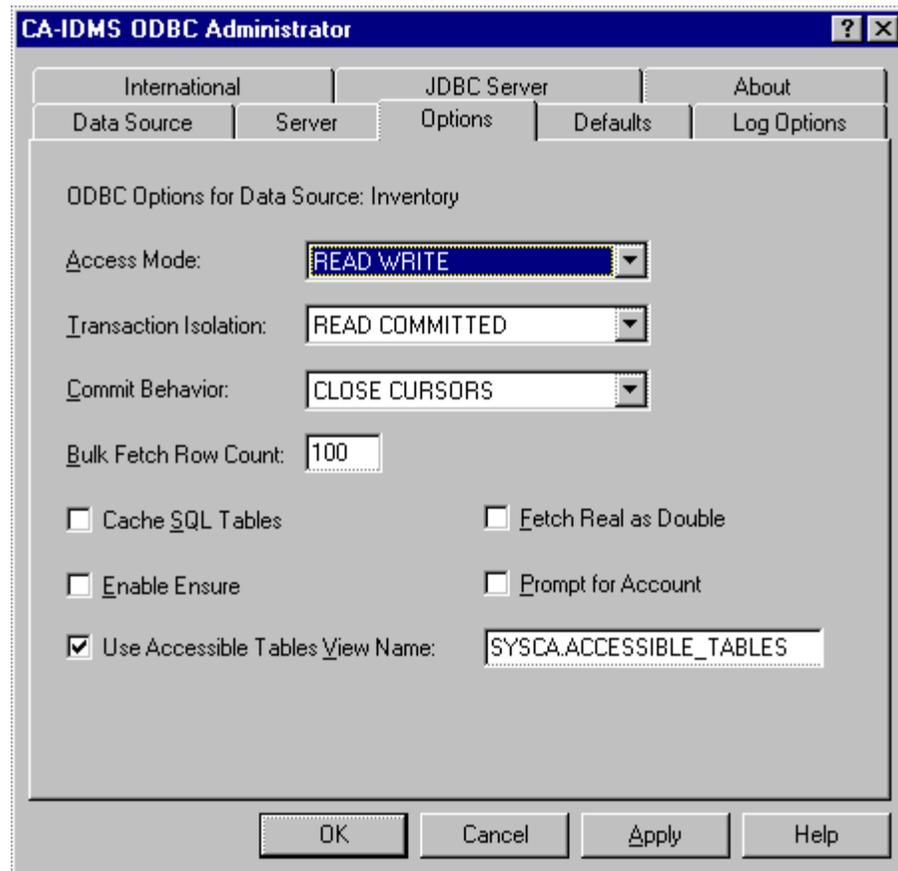
Server Name: Specify the name or TCP/IP address of the CAICCI host server. This overrides the default CAICCI server name specified using the CAICCI/PC Properties dialog **for this server only**, and allows concurrent access to multiple CAICCI servers.

Server Port: Specify the TCP/IP port of the CAICCI host server. This overrides the default CAICCI server port, specified using the CAICCI/PC Properties dialog, **for this Server only**. Enter 0 to use the default value set by CAICCI, typically 1202.

Delete: Delete the server definition and close the dialog.

Setting ODBC Options

Select the Options tab to set ODBC options for a specified data source. Only the ODBC Driver uses these options.



The DSN appears at the top of the dialog. Modify the displayed options for database access using this data source:

Access Mode: Specify the level of access to the database to allow. Choose either of the following:

- **Read Write:** Allows applications to read and update the database. This is the default setting.
- **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.

Transaction Isolation: Specify the degree to which your transactions impact, and are impacted by, other users accessing the same data. Choose either:

- **Read Committed:** Prevents access to data updated by another user, before it has been committed. This corresponds to the CURSOR STABILITY option of the CA-IDMS 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.

This option is defined by the ODBC specification and can be set and queried by the application using the SQLSetConnectOption and SQLGetConnectOption functions. For more information about the SET TRANSACTION statement, see the *CA-IDMS SQL Reference Guide*.

Commit Behavior: Specify the way in which COMMIT operations affect cursors in CA-IDMS. 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.

This is a CA-IDMS Server extension allowing you to optimize ODBC usage by different applications. The ODBC application can use the SQLGetInfo function to query this setting.

Bulk Fetch Row Count: Specify the number of database rows to be fetched in a single database request, to improve performance. The default is 100. Valid values are 1 to 30000. Choose the default unless experience suggests another value for performance reasons.

Cache SQL Tables: Specify that the CA-IDMS Server caches table lists returned by the SQLTables function. Although the default setting is Off, enabling this is recommended. 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, CA-IDMS Server uses the cached result to process repeated SQLTables requests. CA-IDMS Server 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: Select this option to enable the ENSURE parameter of the SQLStatistics function.

The ENSURE parameter of the SQLStatistics function call normally results in 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: This option forces the CA-IDMS Server 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. See the *CA-IDMS Server Administration* manual for more information.

Use Accessible Tables View Name: Check the Use Accessible Tables View Name field to enable you to enter the name of a view to use for the SQLTables function, 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))

For information about the SYSCA.ACCESSIBLE_TABLES view, see the *CA-IDMS SQL Reference Guide*.

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

Performance Considerations for ODBC Options

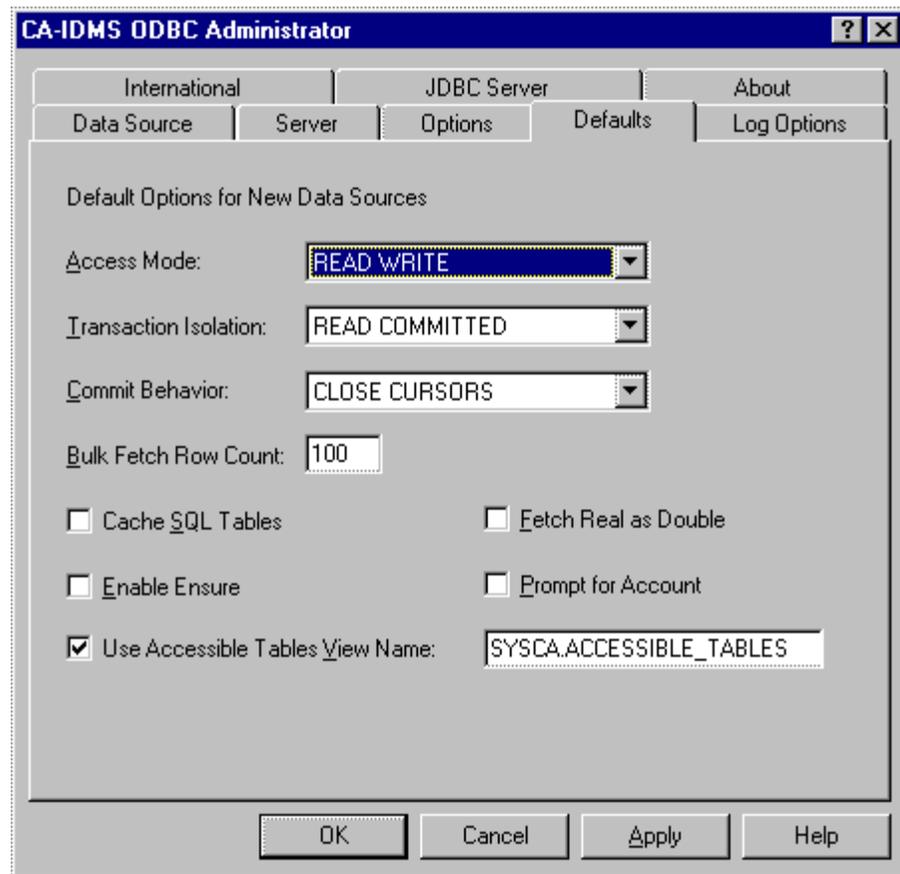
The following ODBC options can affect the performance of the CA-IDMS Server ODBC driver:

- **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:** When disabled, this prevents the SQLStatistics function from issuing commands to update table statistics.
- **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.

Refer to Microsoft documentation about ODBC software for more information about the various ODBC functions mentioned in the previous descriptions.

Specifying CA-IDMS Administrator Defaults

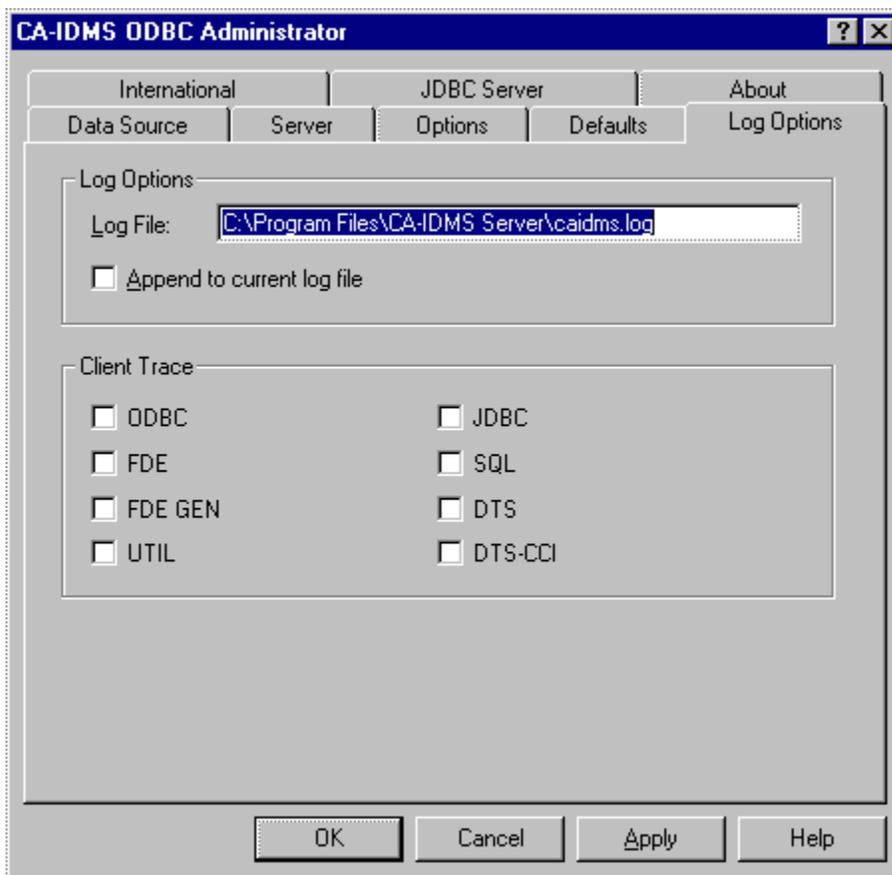
Administrator defaults establish settings for new data sources, applied by the CA-IDMS Administrator each time you define a new data source. Existing data source definitions are not affected. Use the Defaults tab to change default ODBC option settings.



The fields on the Defaults tab are identical to those on the Options tab. For a description of each option, see the previous section “[Setting ODBC Options](#)” 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.



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.

The default log file name is **CAIDMS.LOG**. CA-IDMS Server writes messages to the log file about the status of the database connection. Client trace options control tracing on the PC.

Under Log Options, specify the following options:

Log File: Specify the name of the log file into which the CA-IDMS Server enters messages indicating the status of the database connection. If you omit path information in the file name, CA-IDMS Server creates the file in the directory specified during installation to contain the CA-IDMS Server files. 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 run time.

Append To Current Log File: This option causes the log file to be appended with new information every time an ODBC session is started. 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 Computer Associates Technical Support. Select the check boxes under Client Trace Options as requested by Computer Associates Technical Support to collect trace information:

- **ODBC** - enables tracing of calls to the ODBC driver
- **FDE** - enables tracing of Format Descriptor Element (FDE) conversion calls
- **FDE GEN** – no longer used
- **UTIL** - enables tracing of internal utility calls
- **JDBC** - enables tracing of calls to the JDBC driver
- **SQL** - enables tracing of calls to the native SQL client interface
- **DTS** - enables tracing of calls to the Data Transport Services (DTS) interface
- **DTS-CCI** –enables tracing of calls from DTS to CAICCI

Setting Language Options

When CA-IDMS 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.

Note: The language setting is a global option, affecting all data sources. You cannot establish different language options for different data sources. These conversions are performed for both the ODBC and JDBC drivers.

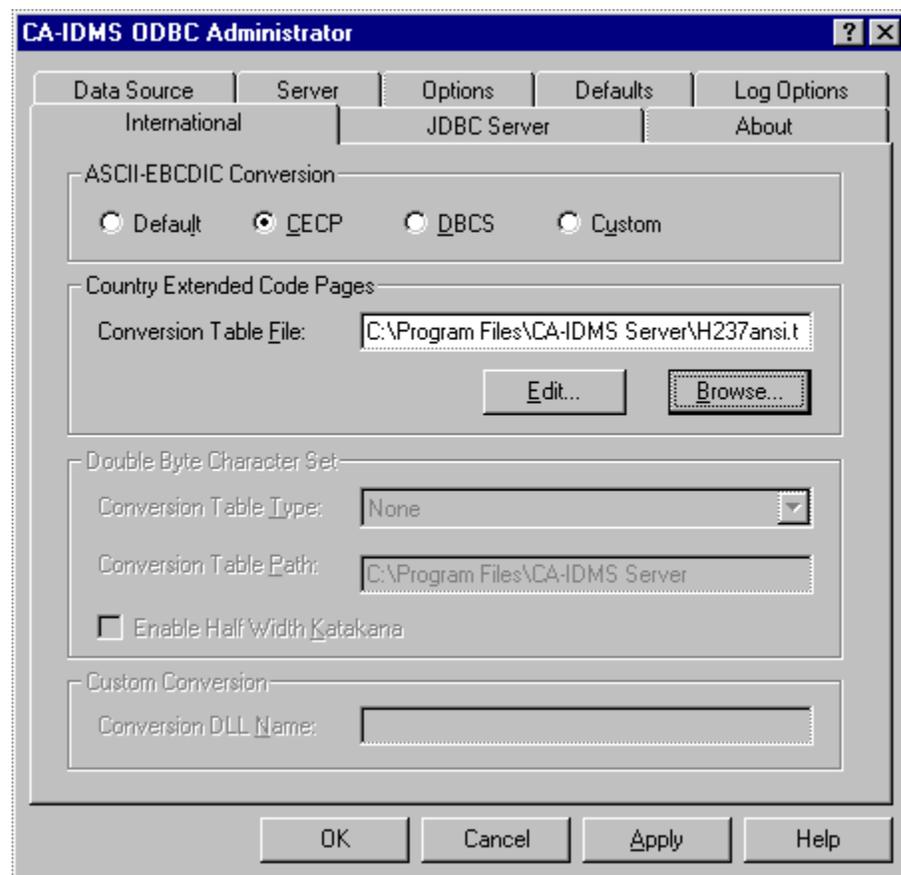
Using the International Tab

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:** Specify the use of the default conversion tables.
- **CECP:** Enable the CECP options in the Country Extended Code Pages box
- **DBCS:** Enable the DBCS options in the Double Byte Character Set box
- **Custom:** Enable 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 ODBC application. Under Country Extended Code Pages, select the file containing the conversion tables.

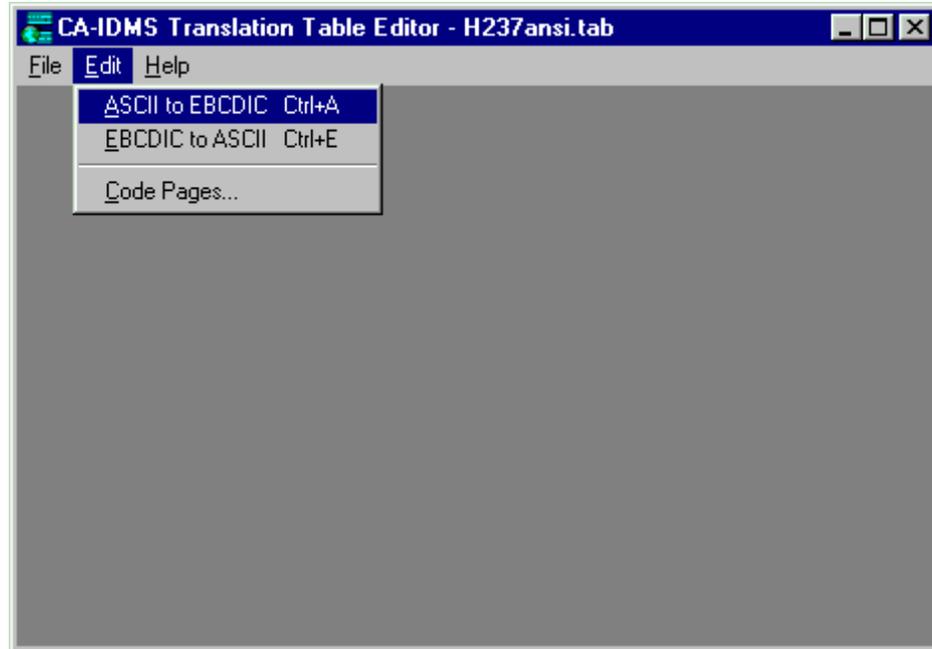


To select a translation table, enter the name of the table file in the Conversion Table File field or click the Browse button to select from the list of available files.

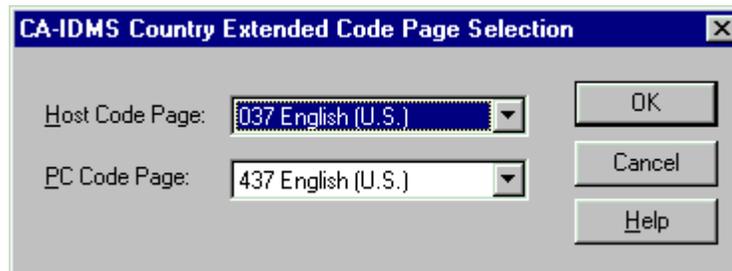
Click the Edit button to activate the Translation Editor to create or edit a translation table.

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 allows you to select the code pages to use for your translation table.



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

Code Page	Representative Language
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:

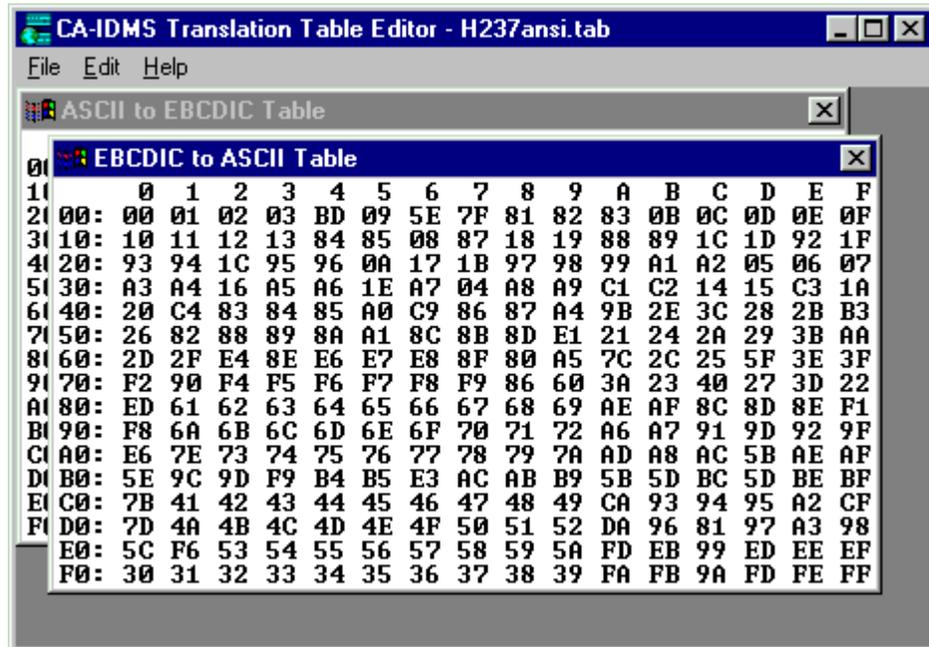
Code Page	Representative Language
437 English (U.S.)	English and most other European languages
850 Multilingual (Latin I)	Most languages using the Latin alphabet
852 Slavic (Latin II)	Slavic languages using the Latin alphabet
860 Portuguese	English and Portuguese
863 Canadian-French	English and French 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.



The generated tables convert control codes between their ASCII and EBCDIC equivalents, where possible. (Releases prior to CA-IDMS Server 4.0 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 through 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**

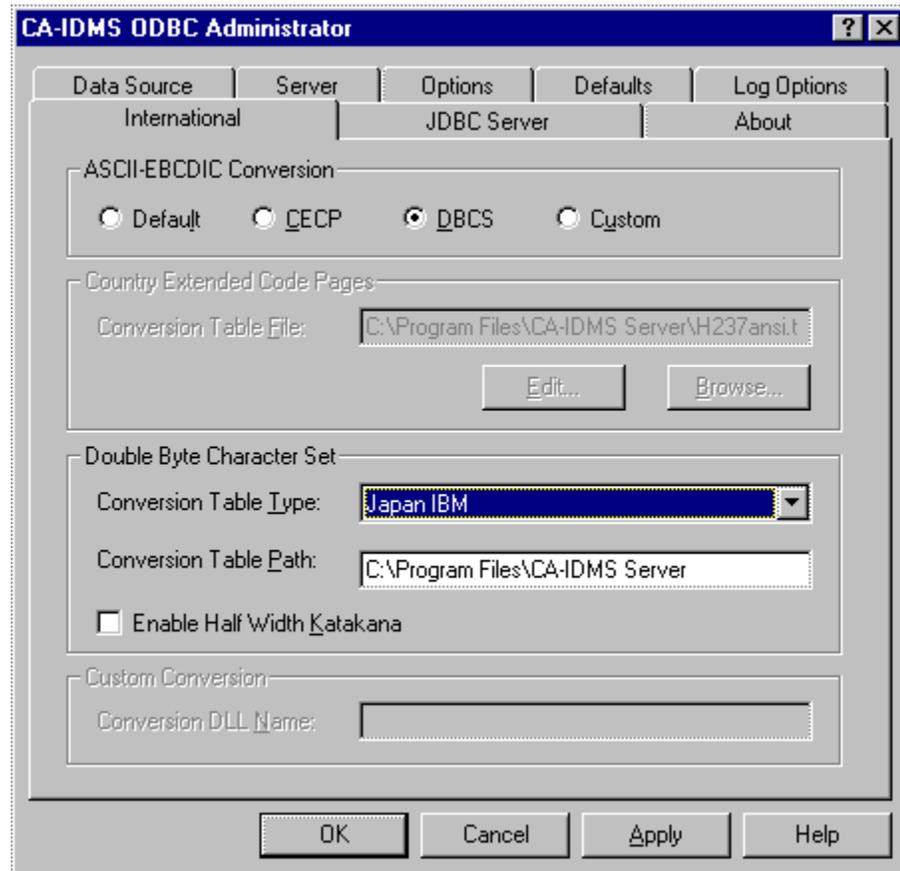
Included Tables

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 run time to convert between ASCII and EBCDIC. These tables are:

Table	Converts
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'.
danish.tab	Host code page 037 to pc code page 850. Control codes are converted
swedish.tab	Host code page 037 to pc code page 850. Control codes are converted

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.



Under Double Byte Character Set, set the following options:

Conversion Table Type: Use the drop-down list to select the types of DBCS used by your CA-IDMS system.

Conversion Table Path: Specify the path of the subdirectory containing the DBCS conversion tables. Typically, the default is accepted.

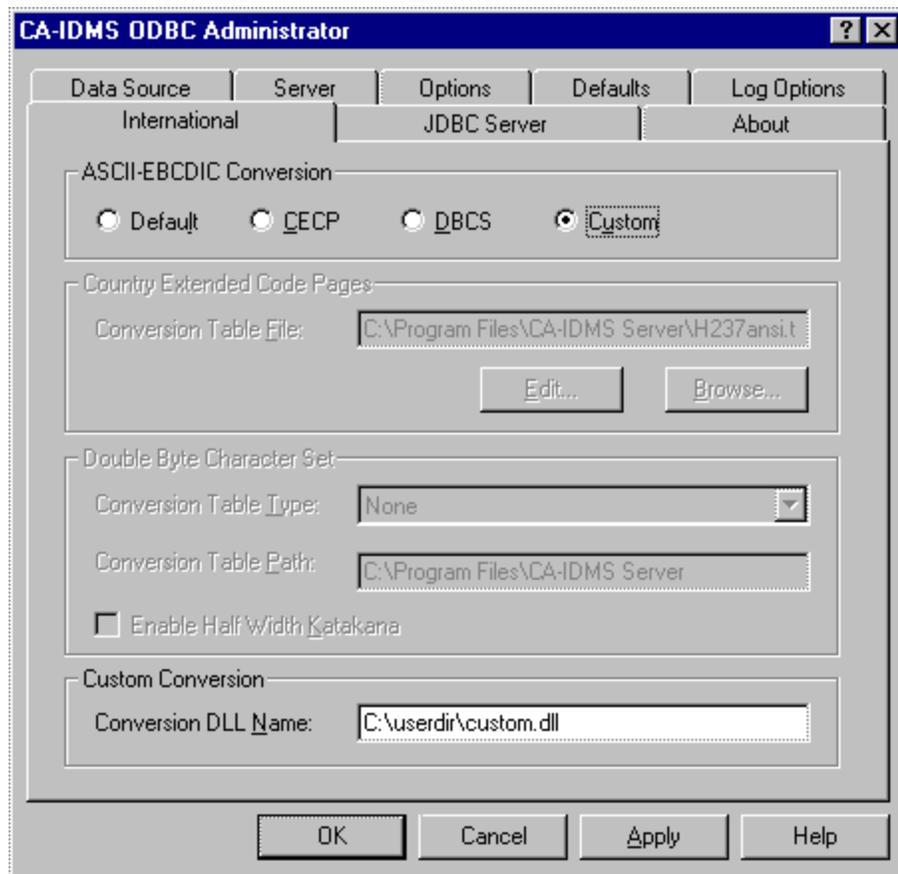
Enable Half Width Katakana: Check this box to enable half width Katakana support when DBCS is enabled. All lowercase characters in CHAR and VARCHAR data are treated as half width Katakana. This does not affect GRAPHIC, VARGRAPHIC, and mixed data within SO and SI in CHAR and VARCHAR types. Only uppercase 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 will be searched automatically by Windows, such as the SYSTEM32 subdirectory or a directory specified in the PATH.

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. Since CA-IDMS Server does not use them, they are not documented here.

DBCAlloc

```
UINT DBCAlloc(HANDLE * hDBCS)
```

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

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

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

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

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

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

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

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

Converts the input buffer from EBCDIC to ASCII. The caller must allocate the output buffer and provide an output field for the converted length. Since 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: DBCCHAR (includes VARCHAR) DBCGRAPHIC (includes VARGRAPHIC)
nLength:	Buffer for converted length

Returns:

DBC_SUCCESS:	Completed successfully
DBC_NO_MEMORY:	Unable to allocate memory
DBC_INVALID_HANDLE:	hDBCS is Null
DBC_ERR_PARM:	Invalid parameter passed
DBC_TRUNCATION:	Converted data was truncated

DBCStoMF

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

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

```
UINT DBCSEnd(HANDLE hDBCS)
```

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

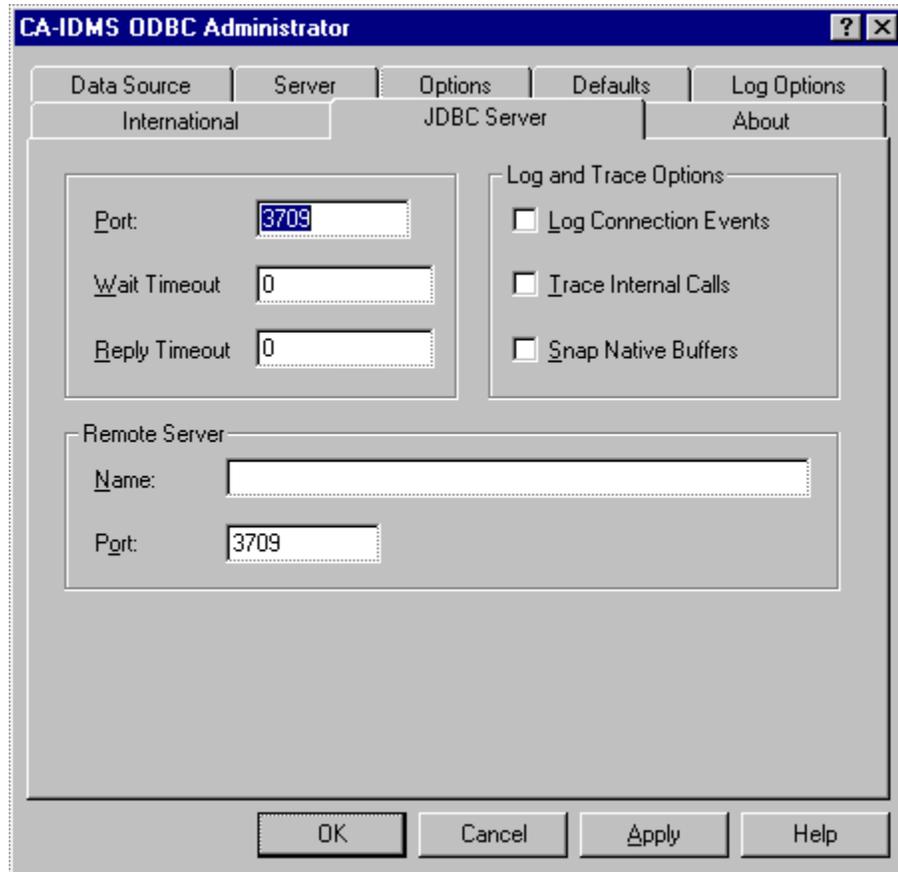
How CA-IDMS Server Uses the API

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

DBCAlloc	Called before any other processing is done
DBCInit	Called after DBCAlloc 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 CA-IDMS 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.
DBCStoMF	Called for each field converted from ASCII to EBCDIC.
DBCSEnd	Called before unloading the DLL

Configuring the JDBC Server

The CA-IDMS ODBC Administrator is used to configure the JDBC Server to allow access to CA-IDMS from browser applets. Note that neither the CA-IDMS JDBC driver nor the JDBC Server actually uses ODBC at runtime. To configure the JDBC Server, select any CA-IDMS data source from the ODBC Administrator, and then select the JDBC Server tab.



Port: Specify the TCP/IP port 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. For information about the URL recognized by the CA-IDMS JDBC Driver, see the "[JDBC Programmer Reference](#)" appendix. The default is 3709.

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: Specify 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.

Under Log and Trace Options, select the following options:

Log Connection Events: Enables logging of connection requests and terminations by the JDBC Server to the Windows NT Application Event Log. By default, only startup, shutdown, and error events are logged.

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.

It is possible to route JDBC connections to another JDBC Server before communicating with CA-IDMS. This can be useful when security requirements prevent the machine on which the web server is running from directly connecting to the mainframe. Under Remote Server, specify the following to use a remote server:

Name: The DNS name or IP address of the remote JDBC Server machine.

Port: The listener port of the remote JDBC Server. The default is 3709.

See the chapter "[Using the CA-IDMS JDBC Server on Windows](#)" in this guide for more information.

Using the ODBC Driver on Windows

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

This chapter discusses the elements of data source connection, and how to use the two CA-IDMS DriverConnect dialogs to connect to data sources. These dialogs are implemented in the CA-IDMS ODBC driver.

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. See the appendix “[JDBC Programmers Reference](#)” in this guide for more information about connecting to a data source using JDBC.

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:	Inventory	OK
User Id:	invuser	Cancel
Password:	*****	Help
Account:		

Enter your user ID and password, and an optional 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 allow you to 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.

The screenshot shows the CA-IDMS DriverConnect dialog box with the following fields and values:

- Required:**
 - Dictionary: INVICT
 - Node Name: SYST0081
 - User Id: invuser
 - Password: [masked]
- Optional:**
 - Task Code: CASERVER
 - Account: [empty]
- CCI Options:**
 - Wait Timeout: 0
 - Server Name: CCISERVER
 - Server Port: 1202

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.

Under Required, enter the following information:

Dictionary: Specify 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: Specify 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: Enter a valid user ID for the CA-IDMS system.

Password: If required for the system, enter a password in this field

Under Optional, specify the following:

Task Code: Specify 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: Enter your account, if your site requires it.

Under CCI Options, specify the following options:

Wait Timeout: Specify 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 multi-threading 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

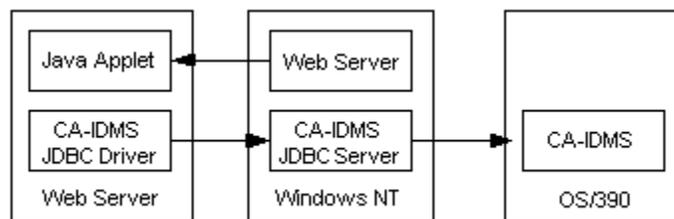
Server Name: Specify the name or TCP/IP address of the CAICCI host Server, overriding the default CAICCI Server name **for this connection only**.

Server Port: Specify the TCP/IP port of the CAICCI host Server, and override the default CAICCI Server port **for this connection only**. Enter 0 to use the default value set by CAICCI, typically 1202.

Note: This data source exists only for the duration of the connection.

Using the CA-IDMS JDBC Server on Windows

The CA-IDMS JDBC Server is a Java application that acts as a proxy server for the CA-IDMS JDBC driver, allowing a JDBC applet running in a web browser to access a CA-IDMS System.



The JDBC Server is implemented as a Windows NT service and supports web servers running on Windows NT. A command line version is also provided to support web servers running on other Java 1.1 (or later) platforms. The Java security model generally prevents an applet from opening a socket connection with a machine other than the web server from which it was loaded.

Installing the JDBC Server

The CA-IDMS JDBC Server is installed automatically when the Typical option is selected during the installation of CA-IDMS Server. If you select the Custom installation option, the CA-IDMS JDBC Server is selected and installed by default. CA-IDMS Server can be reinstalled to add the JDBC Server, if needed. The CA-IDMS JDBC Server must be installed on the same machine as the web server.

If the Custom installation option is used and the ODBC Driver is not installed, the registry settings for the JDBC Server must be set manually. The JDBC Server can be used only as an intermediate server and must access CA-IDMS via another JDBC Server running on either OS/390 UNIX System Services or a Windows machine with the complete product installed.

Configuring the Web Server for Applets

The CA-IDMS JDBC Driver classes must be installed in a directory accessible to web pages accessed from the web server. These classes are distributed as both standard Java archive (JAR) and as Microsoft cabinet files. These files, **idmsjdbc.jar** and **idmsjdbc.cab**, are installed into the Java\classes subdirectory of the CA-IDMS Server directory. Define a virtual directory for the web server pointing to this directory, or copy the class files to an existing accessible directory.

Include an ARCHIVE parameter naming **idmsjdbc.jar** in the Applet tag of the HTML page. A PARAM cabinets tag can also be used with Internet Explorer to name the **idmsjdbc.cab** archive. You can refer to the **IdmsJcf** applet for an example of how to do this. Refer to the "[IdmsJcf](#)" section of the "JDBC Programmer Reference" appendix for more information. See **idmsjcf.html** in the CA-IDMS Server Java directory.

Using the JDBC Server on Windows NT

Use the CA-IDMS Server menu or the Windows NT Control Panel Services applet to start the CA-IDMS JDBC Server service. By default, it is installed as a manually started Windows NT service. You can use the Control Panel to set it to start automatically.

The URL used by the applet 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. See the appendix "[JDBC Programmer Reference](#)" in this guide for a description of the URL recognized by the CA-IDMS JDBC driver.

Using the JDBC Server on Windows 98

The CA-IDMS JDBC Server can be used on Windows 98, 95, and Millennium Edition. A version of the JDBC Server service wrapper, **jsrv.exe**, is installed in the CA-IDMS Server\Java directory. This version invokes the JVM using the Java command (as provided by the JRE or Java Development Kit (JDK) from Sun Microsystems). It can also be used on Windows NT and 2000, if desired. Like the Windows NT service version, configuration settings are maintained in the registry, and can be updated using the CA-IDMS ODBC Administrator.

This version of the JDBC Server is controlled like the OS/390 version (refer to the [“Controlling the JDBC Server”](#) section of the chapter “Using the Client on OS/390” for more information). The following commands can be executed in a command window or as part of a batch file:

Command	Description
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 PATH environment variable must include the directory to the Java executable, installed as part of the JRE or JDK. The CLASSPATH environment variable must include the complete path to the CA-IDMS JDBC Server archive, **idmsjsrv.jar**.

The JDBC Server sends status messages to the Windows NT Event Log. See the [“Monitoring the JDBC Server”](#) section in the “Using the Client on OS/390” chapter of this guide for more information. See the section [“Using the JDBC Server on Other Platforms”](#) of the chapter “Using the Client on Other Platforms” in this guide for more detailed command information.

Installing the Client on OS/390

CA-IDMS Server for OS/390 is distributed on the CA-IDMS Server CD. The installation process consists of three parts:

1. Copy the distribution files from the CA-IDMS Server CD to any directory on your hard disk, customize them, and upload them to OS/390 data sets using FTP.

This part of the installation process is done in the Windows environment, using a text editor such as Notepad or Word and a command prompt window. You may find it helpful to set the Command Prompt window or buffer size larger in order to see messages produced by the batch files used.

2. Set up the OS/390 UNIX System Services Hierarchical File System (HFS) where the run-time files will be installed.

This part of the installation process is performed with a mixture of batch jobs and commands entered into the OMVS shell. The user ID may need to be authorized to allocate datasets on SMS packs, or to set up a new OMVS group and owner, depending on the optional steps used.

3. Link and install the run-time software into the HFS using SMP/E. This is done using batch jobs.

CA-IDMS Server for OS/390 calls CAICCI directly, and does not use the OMVS interface, libcci.so. Installation of Unicenter TNG Framework for OS/390 is not a prerequisite to use CA-IDMS Server on OS/390.

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 OS/390.

Step 1: Load the Installation Files

Insert the CA-IDMS Server CD into a CD drive. When the CA-IDMS Server setup window is displayed, select the Copy the OS/390 Installation Files option and follow the prompts to load the installation files onto your hard disk.

If the setup window does not appear automatically when you insert the CD, open a My Computer window, right click on the CD icon, and select AutoPlay, or simply copy the files from the \OS390\DATA directory on the CA-IDMS Server CD into a directory of your choice. You must turn off the Read Only attribute of the files before editing them.

Step 2: Customize the Installation Files

Edit **pdsedit.bat**, using any text editor in Windows, and run it to customize the installation files:

pdsload.txt: Contains FTP commands to upload the distribution files to OS/390 data sets and run the reload batch job.

pdsload.jcl: OS/390 JCL for the batch job that reloads the JCL, object, and export partitioned data from the uploaded distribution files.

In Windows 95, you may have to increase the environment variable space for the MS-DOS windows used to run **pdsedit.bat**.

If you must rerun **pdsedit.bat**, the original versions of these files are backed up as **pdsback.txt** and **pdsback.jcl**. The following variables are customized in both files:

Variable	Description
\$USER\$	User ID on this OS/390 system
\$HLQ\$	High-level qualifiers of the data set names used during installation
\$VOLSER\$	Name of the disk pack for the data sets

The following variables are customized in **pdsload.txt**:

Variable	Description
\$HOST\$	The DNS name or TCP/IP address of the OS/390 system where the files will be uploaded
\$PASS\$	Password on this OS/390 system

The following variables are customized in **pdsload.jcl**:

Variable	Description
\$UNIT\$	Disk unit associated with the VOLSER, typically 3390 or SYSDA
\$CLASS\$	Job class, typically A
\$MSGCLASS\$	Message class, typically X to hold output

Note that the only variables modified in **pdsload.jcl** are on the JOB card.

Step 3: Upload OS/390 Datasets

Run the **pdsload.bat** batch file in a command prompt window. This invokes FTP with the commands in **pdsload.txt** to upload the files and submit the JCL contained in **pdsload.jcl**. Check the status of the batch job on OS/390. This job allocates and populates four OS/390 datasets:

Dataset	Description
\$HLQ\$.JCL	Jobs to complete the remainder of the installation
\$HLQ\$.OBJ	Modules to be pre-linked and linked into HFS
\$HLQ\$.EXP	Export modules used in the pre-link process
\$HLQ\$.TAR	Archive containing HFS files

This is the end of the work done in Windows. All remaining steps are done in OS/390.

Step 4: Customize the Installation JCL

Edit the JOBCARD, JOBCARD2, and SRVINS00 members in the JCL library, allocated in the previous step, to customize them for your environment. The SERVEMAC member is an edit macro used to make the necessary changes.

JOBCARD contains the JOB card to be used in all the required installation jobs. Some jobs are optional because the environmental changes they make, such as allocating more space on an SMS pack or creating a new OMVS group and user, may be unnecessary at your site. These jobs, which require increased authority, use JOBCARD2.

SRVINS00 contains the JCL for all the installation jobs, in the form of a series of IEBGENER steps to create customized members of the JCL library for each job step. SRVINS00 contains 21 literal strings to be modified as appropriate for your environment.

You can change each value in these files manually, or use the SERVEMAC edit macro to customize the three files as described in the following sections. All of the values have defaults.

Using the SERVEMAC Edit Macro

SERVEMAC contains 21 numbered boxes, each containing a literal string, a description of the item, and a value to be used for the global change. To use SERVEMAC, change the values as needed, save the file to a library that can be accessed by your TSO session (as described below), edit each of the three members, and invoke the macro by entering it on the command line.

An edit macro is a collection of ISPF editor commands that can be invoked when the file is open in the ISPF editor. Note that you cannot run an edit macro like SERVEMAC from ISPF Option 6 as if it were a CLIST.

An edit macro must be available to your TSO session. The easiest way to make an edit macro available is to copy it into a CLIST library already concatenated to SYSPROC in your TSO logon proc. If you do not have authority to update any CLIST libraries, you can temporarily concatenate this library to SYSPROC until you log off, by issuing the following two TSO commands:

```
ALLOD DD(TEMP) DATASET('lib.containing.servemac') SHR REU
CONCAT DD(SYSPROC TEMP)
```

Scroll through the resulting SRVINS00 member. To change the values, cancel out of the SRVINS00 member, make additional changes to SERVEMAC, and repeat. You may wish to make a copy of SRVINS00 before beginning.

When you have used SERVEMAC to customize SRVINS00, go on to the 'Set Up a New OMVS Group and User' step.

Editing the JCL Manually

If you choose not to use the SERVEMAC edit macro, edit JOBCARD, JOBCARD2, and SRVINS00 and change all occurrences of the literals. The data is case-sensitive, and must be in upper case. Turn CAPS ON and press Caps Lock before making changes.

Value	Description
%USER%	For JOB cards, user ID
%ACCT%	For JOB cards, account number
%CLASS%	For JOB cards, job class for short jobs
%MSG%	For JOB cards, message class
%HLQ%	High level qualifiers of data set names used during installation.
%VOLSER%	Disk pack for permanent datasets
%UNIT%	Disk unit associated with above VOLSER
%WORK%	Disk unit for temporary datasets
%JCLLIB%	Name of this library
SYSOUT=*	SYSOUT class
SYS1.MACLIB	OS/390 system macro library
CEE.SCEEMSGP	Message dataset for LE370 C linker

Value	Description
CEE.SCEE OBJ CEE.SCEELKEX	Resolution object libraries for LE370
CEE.SCEELKED SYS1.CSSLIB	Libraries containing callable BPX routines

If you must allocate an SMS dataset to create space in the HFS, the following global changes are also required.

%HFSDSN%: Data set name for HFS allocation on an SMS volume

%SMSSTORC%: Disk unit to ensure HFS allocation on an SMS volume

The following global changes are optional and are based on site-specific security requirements. These global changes can be omitted if the files can be installed into the HFS using an existing user and group.

%TUSER%: User ID of an authorized security administrator able to use RACF, CA-ACF2, or CA-TOP SECRET commands to create a new OMVS owner and administrator for the CA-IDMS Server HFS files.

%PW%: Password for this user

Make the following global change in the SRVINS00 member. This data is case-sensitive. Turn CAPS OFF and ensure that the Caps Lock key is not pressed.

%IDMSDIR%: The path of the install directory in the HFS.

Review the customized SRVINS00 job and submit it to create customized install jobs in members SRVINS01 through SRVINS10. This job contains COMPRESS steps and may be rerun.

Step 5: Set Up a New OMVS Group and User

Submit SRVINS01 to issue security administrator commands to set up a new OMVS group and user. This new user will be the owner and administrator of the CA-IDMS Server HFS files.

This job is optional. If the run-time files can be installed into the HFS using an existing user and group, then this job is not required.

Before running this job, delete the portions of the job pertaining to security software not installed at your site.

Step 6: Allocate the HFS

Submit SRVINS02 to allocate the HFS. CA-IDMS Server requires approximately 1.3 MB in the HFS during the installation. Log files created at run time may require additional space, and can be allocated in a separate HFS devoted to temporary files, if desired.

This job is optional. You can install CA-IDMS Server into an existing HFS if space is available.

Step 7: Create the Installation Directory in the HFS

Create the CA-IDMS Server installation directory in the HFS. In all files where %IDMSDIR% was referenced and changed to the case-sensitive path name, the path name must point to this directory.

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, reflecting the customized values from the previous step "Customize the JOBCARD" under the OMVS shell, invoked by entering the TSO OMVS command:

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

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)
```

Where *idmsdir* is the installation directory specified by %IDMSDIR% and *HFSDSN* is the name of the dataset specified by %HFSDSN%.

Step 8: Create Subdirectories, Allocate Datasets, and Prelink Object Modules

Submit SRVINS03 to copy the tar file to the HFS and extract it to create the CA-IDMS Server subdirectories and files (other than executables, which are installed using SMP/E, as described in the following section). The tar file is not used after this step and can be deleted from the HFS after it is extracted to save space, if desired. This job uses BPXBATCH to execute OMVS commands. Output is written to **srvins03.out** and **srvins03.err**, which contains error messages, if any.

Submit SRVINS04 to allocate datasets used by the CA-IDMS Server installation process.

Submit SRVINS05 to prelink the object modules into the indirect object (INDOBJ) library.

Step 9: Install the Executable Modules Using SMP/E

Submit SRVINS06 to create a new SMP/E environment. Omit this job if you are installing CA-IDMS Server into an existing SMP/E environment, such as the CA-IDMS environment.

The FMID for the portion of CA-IDMS Server delivered on the CA-IDMS tape is CXS2000. The FMID for this portion of CA-IDMS Server is CXS430E (Open Edition). CXS2000 and CXS430E can both be installed into the same SMP/E environment (the CA-IDMS SMP/E environment), but you are not required to do so.

Submit SRVINS07 to customize the SMP/E CSI for CA-IDMS Server. If you are installing into an existing SMP/E environment, ensure that the global zone contains DDDEFs and that the environment contains an SMPLTS dataset. If the global zone does not contain DDDEFs, add them using SRVINS06 as a guide, or add additional DD statements to the SRVINS08 job. If the SMPLTS library does not exist, you must allocate it, using the SRVINS07 job as a guide.

Submit SRVINS08 to run SMP/E to receive, apply, and accept the executable modules. This job links the prelinked object modules from INDOBJ into the */idmsdir/bin* directory in the HFS.

Step 10: Set File Access Bits

Submit SRVINS09 to set the appropriate mode bits for the executable files in the */idmsdir/bin* directory. The user ID associated with this job must be the owner of the files or have superuser authority. BPXBATCH output is written to **srvins09.out** and **srvins09.err**.

Submit SRVINS10 to set the appropriate mode bits for the JSRV service wrapper to allow it to run under the user ID of the owner instead of the user ID that starts the service. The user ID associated with this job must be the owner of the file or have superuser authority. BPXBATCH output is written to **srvins10.out** and **srvins10.err**.

Step 11: Delete Unnecessary Files

You can delete the following files from the installation directory for space reasons, when the installation is complete:

- hfs.tar
- instpax
- *.out
- *.err

Configuring the Client on OS/390

This chapter describes how to configure CA-IDMS Server in the OS/390 UNIX System Services (USS) environment for use by JDBC enabled applications, as well as the environment variable settings and configuration file information needed to access a CA-IDMS database. It assumes that you are familiar with the OS/390 USS shell and HFS.

Configuring CA-IDMS

CA-IDMS Server is installed into a subdirectory in the HFS. For more information about installation, see the chapter [“Installing the Client on OS/390”](#) in this guide. This subdirectory, specified when the product is installed, is referred to in this document as */idmsdir*. Its structure is as follows:

- */idmsdir*: Configuration file and JDBC server shell scripts
- */idmsdir/bin*: Executable files, including the JDBC Server service wrapper and native SQL client interface DLLs (shared object libraries)
- */idmsdir/classes*: Java archive (jar) files including the CA-IDMS JDBC Driver, JDBC Server, and JCF demo.
- */idmsdir/log*: Default directory for the log (trace) file.

See the [“Using the JDBC Server on Other Platforms”](#) section of the chapter [“Using the Client on Other Platforms”](#) for information about configuring CA-IDMS Server for use with applets and JDBC enabled applications running on other platforms.

Specifying Environment Variables

You must specify the locations of the executables, DLLs, and Java class files for an OS/390 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 CA-IDMS JDBC Driver, these variables could be set as follows:

```
set PATH $JAVA_HOME/bin:$PATH
export PATH
set LIBPATH /idmsdir/bin:$LIBPATH
export LIBPATH
set CLASSPATH /idmsdir/classes/idmsjdbc.jar:$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 these 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** directory.

If you do not use the supplied shell scripts to run the CA-IDMS JDBC Server, the environment variables might be set as follows:

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

Optional environment variables, specific to CA-IDMS, include:

- **IDMS_CFG_PATH:** Specifies the configuration file name or path.
- **IDMS_CFG_RELOAD:** Forces reloading of the configuration file.

The use of these is described in the appendix "[Configuration File Information](#)" in this guide.

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 OS/390.

This file, by default, is named **caidms.cfg** and is found in the installation directory, but you can use the **IDMS_CFG_PATH** environment variable to give it a different name, or to locate it elsewhere.

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

```
set IDMS_CFG_PATH /usr/appdir/
export IDMS_CFG_PATH
```

Data Source Definitions

A JDBC-enabled application connects to a database by specifying a URL. Each JDBC driver specifies the format of the URL it can recognize. The URL recognized by the CA-IDMS JDBC Driver includes a data source name similar to an ODBC style data source. The 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. For a complete description of the URL, see the appendix "[JDBC Programmer's Reference](#)" in this guide.

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 allows meaningful names to be used for the data source and server names. Using an explicit server section allows optional information to be specified for a CA-IDMS system. When using all default values, this is equivalent to the following minimal data source definition:

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

Since the characters [or] may be difficult to use on 3270 terminals or emulators, you can substitute \$ characters; for example, \$APPLDICT\$.

Configuring the JDBC Server

The JDBC Server can be customized with settings in the [Proxy] section of the configuration file, or you can accept the default settings for any or all of the following options:

Host: (Optional) Specify the DNS name or TCP/IP address of the host. This can be used to force 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: Specify the TCP/IP that the server uses to listen for connection requests. If the port is changed from the default, 3709, client applications must specify the correct port in the URL.

Encoding: Specify the way character data is converted. The JVM represents all character data 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.

The default platform encoding on OS/390 OMVS is Cp1047. The corresponding character conversion classes are generally not available on other 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. Cp037 is a standard IBM encoding supported by the Sun and IBM Java implementations.

In the absence of documentation, it may be possible to determine the encodings supported by converted classes supplied with the Java implementation. These are generally named **ByteToCharXXXXXX.class** and **CharToByteXXXXXX.class**, where XXXXXX is the encoding name. On the Java 2 Platform these classes are included in a separate archive file, **il8n.rt**.

WaitTimeout: Specify how long the JDBC Server waits for the next request from a connected client. The default, 0, specifies an indefinite wait. However, 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.

LogLevel: Control the types of messages sent to the system log or operator console. The default, 8, sends start up, shut down, error, and warning messages.

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. See the appendix "[Configuration File Information](#)," for detailed information about all options and settings in the configuration file.

Using the Client on OS/390

This chapter describes how to use CA-IDMS Server in the USS environment on OS/390. CA-IDMS Server supports JDBC-enabled applications running in the USS environment, as well as client applications running on other platforms.

The CA-IDMS JDBC Driver always runs on the same platform as the client application. Applications running on OS/390 use the CA-IDMS JDBC Driver on OS/390, which, in turn, uses the CA-IDMS native client interface to access the CA-IDMS system.

Remote client applications use a local (from the application's point of view) copy of the CA-IDMS JDBC Driver, which uses TCP/IP to communicate with the CA-IDMS JDBC Server on OS/390. The CA-IDMS JDBC Server acts as a proxy server, calling the native client interface on behalf of the remote JDBC driver.

Note: Applications running on OS/390 do not need the JDBC Server to communicate with a CA-IDMS system.

Configuring Applications to Use CA-IDMS Server

JDBC-enabled applications running on OS/390 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 CA-IDMS 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. See the chapter "[Using the Client on Other Platforms](#)" in this guide for more information.

It is usually convenient to define a data source name for each CA-IDMS database to be accessed using JDBC. A DSN can be included in the URL recognized by the CA-IDMS JDBC Driver, and can reference a data source definition in the CA-IDMS Server configuration file, **caidms.cfg**. A data source defines all connection information needed to access the database, including the database name, Node and Task Code. It is also possible to connect to a CA-IDMS database without a DSN, using DriverProperties.

See the chapter “[Configuring the Client on OS/390](#)” in this guide for more information about setting the required environment variables and defining data sources. See the appendix “[JDBC Programmer’s Reference](#)” in this guide, for information on the URL format and DriverPropertyInfo objects used by the CA-IDMS JDBC Driver.

Configuring the Web Server to Use CA-IDMS Server

For an applet to use the CA-IDMS 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/classes
```

The APPLET tag in the web page should include an ARCHIVE tag referencing the CA-IDMS JDBC Driver JAR file. For example, an HTML page that runs the CA-IDMS JCF demo might look like the following:

```
<APPLET CODE="IdmsJcf.class">  
CODEBASE="idmsdir"  
ARCHIVE="idmsjcf.jar,idmsjdbc.jar"  
</APPLET>
```

Controlling the JDBC Server

The CA-IDMS JDBC Server can be controlled using the OMVS shell or with batch jobs. Four batch jobs are included in the installed sample JCL Partitioned Data Set:

JCL Member	Description
JSRVSTRT	Starts the JDBC server
JSRVSTOP	Stops the JDBC Server
JSRVSUSP	Suspends the JDBC Server
JSRVRESU	Resumes the JDBC Server

These jobs use BPXBATCH to run the corresponding shell scripts. See the chapter [“Installing the Client on OS/390”](#) for more information on customizing the sample JCL.

The following table describes the scripts used to control the JDBC Server from the OMVS shell:

Shell Script	Description
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 shell scripts set environment variables, described in the chapter [“Configuring the Client on OS/390”](#), and run the CA-IDMS JDBC Server Service wrapper, which starts the JVM and passes control to the JDBC Server entry point.

When the JDBC Server starts, it typically 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. Tracing output can be redirected to the standard output. Messages to the system log can also be echoed on the standard output. Press Enter to shut down the JDBC Server.

Typically, the JDBC Server gets all run-time options from the configuration file. When debugging, it may be convenient to specify some options on the command line. The shell scripts pass up to 10 options to the **jsrv** executable. When more options are required, **jsrv** can be executed directly, if the necessary environment variables have been set. For detailed information on the command line options see the “[Using the JDBC Server on Other Platforms](#)” section of the chapter “Using the Client on Other Platforms” in this guide.

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 Unicenter TNG Framework and other system management products. These messages are identified by message number, which conforms to the standard OS/390 message format, PPPNNNNS, where:

PPP: Product specific prefix: ‘UJS’

NNNN: Message number: ‘0000-9999’

S: Severity level:

- **E**: Error
- **W**: Warning
- **I**: Information
- **D**: Debugging

The destination and level of messages written are controlled by settings in the configuration file. See the appendix “[Configuration File Information](#)” in this guide for more information.

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

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

Using the Client on Other Platforms

Installable versions of CA-IDMS Server are available for OS/390, Windows 2000, Windows NT, Windows 98, and Windows 95. The CA-IDMS JDBC Driver and JDBC Server can also be used on other platforms that support Java 1.1 (or later) and TCP/IP.

Note: The CA-IDMS JCF demo also requires support for the Swing classes on the client machine.

“Installing” CA-IDMS Server on Other Platforms

CA-IDMS Server can be “installed” on these platforms 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 **/classes** directory:

- **idmsjdbc.tar** compiled class files, archived in jar files
- **idmssamp.tar** sample program source files

Use either the **tar** or **pax** utility, whichever is available, to extract the needed files on variants of UNIX. On Windows, the files can simply be copied directly from the **/classes** directory on the CA-IDMS Server CD.

Using the JDBC Driver on Other Platforms

Applications, application servers, and servlets running on platforms other than Windows or OS/390 can use the CA-IDMS 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 CA-IDMS JDBC Server running on Windows or OS/390. The CA-IDMS JDBC Server does not need to run on the application platform.

To use the CA-IDMS 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 archive file. For example, on Windows:

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

On UNIX:

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

3. Specify the system where the JDBC Server is running as part of the URL used to connect to the database. For example:

```
jdbc:idms://hostname/datasource
```

See the “[Connection Parameters](#)” section of the appendix “JDBC Programmer’s Reference” of this guide, for more information about the URL format.

Using the JDBC Server on Other Platforms

The CA-IDMS JDBC Server can be used as a command line application to support web servers running on platforms other than Windows and OS/390. The JDBC Server application is provided as a Java archive file, and is actually the same file used by the JDBC Server service on OS/390 and Windows 98 or 95. 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. Specify options on the command line.
- The log file is not supported. Redirect **stderr** for messages and **stdout** for trace information.
- The native SQL client is not supported. Connections are routed to CA-IDMS via a JDBC Server running on Windows or OS/390, which is treated as a remote server.

To use the JDBC Server as a command line application:

1. Extract the JDBC Server 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 archive file. For example:

```
set CLASSPATH=/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 &
```

where *host* specifies the DNS name or TCP/IP address of the Windows or OS/390 machine where the native JDBC Server is running, *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 OS/390 or using the ODBC Administrator on Windows are specified on the command line:

Options	Description
-?	Print this information
-h <i>host</i>	Host listener name or IP address
-p <i>port</i>	Host listener IP port
-q <i>count</i>	Host listener queue length
-r <i>host</i>	Remote host name or IP address
-s <i>port</i>	Remote IP port
-c	Enable control by remote client
-e <i>encoding</i>	Override platform encoding
-u	Specify Unicode fallback encoding
-w <i>seconds</i>	Client wait timeout interval
-t <i>seconds</i>	Server reply timeout interval
-b <i>seconds</i>	Socket blocking timeout interval
-v [<i>level</i>]	Syslog message level (level = 10 if not specified)
-l <i>level</i>	Trace log message level

Options	Description
-d <i>option</i> [<i>option</i>]	Enable debugging with the following trace options, where <i>option</i> can be:
trace	enable debug tracing
native	enable native trace
snap	enable object display
buffer	enable native buffer display
object	enable native object display
-i <i>class</i> [<i>class</i>]	Include <i>class</i> in trace
-x <i>class</i> [<i>class</i>]	Exclude <i>class</i> from trace

For detailed information about these options, see the appendix “[Configuration File Information](#)” in this guide.

ODBC Programmer Reference

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

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

Debugging User Sessions

CA-IDMS Server writes messages to the default PC log file, CAIDMS.LOG, specified on the Log and Trace Options tab of the CA-IDMS ODBC Administrator dialog, described in the “[Logging Errors and Trace Information](#)” section of the “Configuring the Client on Windows” chapter of this guide. 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 CA-IDMS system is down.

Error Messages

Error messages returned by CA-IDMS Server have one of the following formats, depending on the component in which the error is detected:

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

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

The CA-IDMS 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, CA-IDMS, and the network components.

ODBC Conformance Levels

CA-IDMS Server conforms to the ODBC 2.5 standard. Unless otherwise noted, all descriptions of ODBC in this document refer to ODBC 2.5.

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, as well as specific API function, data type, and scalar function support.

API Conformance Levels

The ODBC 2.5 API includes three conformance levels:

- **Core API** supports a set of Core functions in the X/Open and SQL Access Group (SAG) CLI specification.
- **Level 1** supports Core functionality plus an extended set of functions.
- **Level 2** supports Core API and Level 1 functionality, as well as an extended set of functions.

The three conformance levels are described in the following sections.

Core API

The ODBC 2.5 Core API corresponds to the functions in the X/Open and SAG CLI specification, with a few minor differences. 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 CA-IDMS ODBC driver supports all Core API functions. The Core API functions are as follows:

- `ConfigDSN`: For use by configuration programs
- `SQLAllocConnect`
- `SQLAllocEnv`
- `SQLAllocStmt`
- `SQLBindCol`

- SQLBindCol
- SQLCancel
- SQLColAttributes
- SQLConnect
- SQLDescribeCol
- SQLDisconnect
- SQLError
- SQLExecute
- SQLExecDirect
- SQLFetch
- SQLFreeConnect
- SQLFreeEnv
- SQLFreeStmt
- SQLGetCursorName: Used by ODBC Cursor Library
- SQLNumResultCols
- SQLPrepare
- SQLRowCount
- SQLSetCursorName
- SQLSetParam: Supported by ODBC Driver Manager for ODBC 1.0 applications only. Replaced by SQLBindParameter (see the following section)
- SQLTransact

Level 1 API

The ODBC 2.5 Level 1 extension to the API supports the Core functionality and also allows the application to set connection and statement options, retrieve information about driver and data source capabilities, and retrieve catalog information.

The CA-IDMS ODBC driver supports all Level 1 API functions. The Level 1 API functions are as follows:

- SQLBindParameter: Replaces SQLSetParam
- SQLColumns
- SQLDriverConnect
- SQLGetConnectOption
- SQLGetFunctions: Implemented by ODBC driver manager
- SQLGetInfo
- SQLGetStmtOption
- SQLGetTypeInfo
- SQLParamData
- SQLPutData
- SQLSetConnectOption
- SQLSetStmtOption: No asynchronous operation; can set timeout, but it is ignored
- SQLSpecialColumns: CA-IDMS does not support auto update columns
- SQLStatistics
- SQLGetData
- SQLTables

Level 2 API

The ODBC 2.5 Level 2 extension to the API supports the Core and Level 1 functionality and also allows scrollable bulk cursors, retrieval of input parameter descriptions, and retrieval of additional catalog information.

The CA-IDMS ODBC driver supports the following Level 2 API functions:

- SQLParamOptions
- SQLMoreResults
- SQLNumParams
- SQLNativeSQL

SQL Conformance Levels

The ODBC 2.5 defines three conformance levels for SQL grammar:

- Minimum grammar supports minimal DDL, simple SELECT, INSERT, searched UPDATE and DELETE, simple expressions, and the CHAR data type.
- Core grammar supports the SQL statements and functionality proposed by the X/Open and SQL Access Group CAE specification (1992).
- Extended grammar supports common SQL extension supported by different database management systems.

The following sections describe the three conformance levels.

Minimum SQL Grammar

The Minimum SQL Grammar is defined by ODBC 2.5 to provide a basic level of conformance. It supports minimal DDL, simple SELECT, INSERT, searched UPDATE and DELETE, simple expressions, and the CHAR and VARCHAR data types.

CA-IDMS supports the Minimum SQL Grammar.

Integrity Enhancement Facility (IEF)

CA-IDMS Server supports referential integrity. Although CA-IDMS Server supports Integrity Enhancement Facility (IEF) functionality, it does not support the syntax defined in the ODBC specification, as noted in the following list. Implementation of IEF is optional for the ODBC driver and does not affect the SQL Conformance Level supported by the driver.

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. Does not support the RESTRICT keyword. The absence of CASCADE implies RESTRICT.
REVOKE	
CASCADE/RESTRICT	CA-IDMS does not support the CASCADE and RESTRICT options on REVOKE.

Core SQL Grammar

The Core SQL Grammar corresponds to the X/Open and SQL Access Group CAE draft specification (1991). It is similar to, but not precisely the same as, the ANSI SQL2 standard. The Core SQL Grammar supports the Minimum SQL Grammar and adds more DDL, full SELECT, positioned UPDATE and DELETE, subquery, set expressions, and additional data types.

CA-IDMS supports the complete Core SQL Grammar.

The CA-IDMS driver converts the following ODBC SQL statements to CA-IDMS SQL:

SQL Statement	Comments
CREATE INDEX <i>user.index</i> DROP INDEX <i>user.index</i>	The CA-IDMS ODBC driver removes the qualifier from the index name if present.
GRANT UPDATE (<i>column-list</i>) REFERENCES (<i>column-list</i>)	CA-IDMS does not support column level security. CA-IDMS driver removes the column list and grants UPDATE to all columns of the table.

Extended SQL Grammar

The Extended SQL Grammar supports the Minimum and Core SQL Grammar and adds outer joins, scalar functions, date/time literals, batch SQL statements, procedure calls, and additional data types.

CA-IDMS does not support the Extended SQL Grammar. Although CA-IDMS does support outer joins, it does not support the outer join syntax. CA-IDMS does not support procedure calls.

The ODBC driver supplied with CA-IDMS Server does not support outer joins and procedure calls. Scalar functions in escape sequences are passed to the data source unchanged except that the escape sequence itself is converted to spaces. SQL statements submitted in batch jobs are also not supported.

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

The ODBC driver supplied with CA-IDMS Server Release 4.0 does not support the following statements specified in the ODBC Extended SQL syntax:

Statement	Comments
ODBC procedure extension	CA-IDMS does not support procedures.
<i>outer join</i> LEFT OUTER JOIN	CA-IDMS supports outer join using the PRESERVE keyword, but the semantics are not identical.
<i>statement-list</i>	CA-IDMS and the CA-IDMS ODBC driver do not support this statement.
ODBC <i>scalar function extension</i>	CA-IDMS supports scalar functions that map to a subset of the ODBC scalar functions, but does not support the ODBC escape sequences for translation of scalar functions.
DELETE WHERE CURRENT OF <i>cursor</i>	Supported by using the ODBC Cursor Library.
SELECT FOR UPDATE	Supported by using the ODBC Cursor Library.
UPDATE WHERE CURRENT OF <i>cursor</i>	Supported by using the ODBC Cursor Library.

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_DATE
DECIMAL	SQL_DECIMAL
DOUBLE PRECISION – See below*	SQL_DOUBLE
FLOAT – See below*	SQL_FLOAT
GRAPHIC (DBCS Disabled)	SQL_BINARY
GRAPHIC (DBCS Enabled)	CAID_GRAPHIC
INTEGER	SQL_INTEGER
LONGINT	SQL_BIGINT
NUMERIC	SQL_NUMERIC
REAL – See below*	SQL_REAL
SMALLINT	SQL_SMALLINT
TIME	SQL_TIME
TIMESTAMP	SQL_TIMESTAMP
UNSIGNED DECIMAL	SQL_DECIMAL
UNSIGNED NUMERIC	SQL_NUMERIC
VARCHAR	SQL_VARCHAR
VARGRAPHIC (DBCS Enabled)	CAID_VARGRAPHIC
VARGRAPHIC (DBCS Disabled)	SQL_BINARY

- *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_LONGVARIABLE	BINARY
SQL_CHAR	CHAR
SQL_DATE	DATE
SQL_DECIMAL	DECIMAL
SQL_DOUBLE	DOUBLE PRECISION
SQL_FLOAT – See below*	DOUBLE PRECISION
SQL_REAL – See below*	REAL
SQL_INTEGER	INTEGER
SQL_BIGINT	LONGINT
SQL_NUMERIC	NUMERIC
SQL_BIT	SMALLINT
SQL_SMALLINT	SMALLINT
SQL_TINYINT	SMALLINT
SQL_TIME	TIME
SQL_TIMESTAMP	TIMESTAMP
SQL_LONGVARCHAR	VARCHAR
SQL_VARCHAR	VARCHAR

- *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 2.5 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 that on CA-IDMS the length is specified in DBCS characters, which is half the precision specified using the CA-IDMS 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 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	Computer Associates	Dictionary name (use with DRIVER only)
NODE	Computer Associates	CA-IDMS System ID (use with DRIVER only)
TASK	Computer Associates	Alternate task code (use with DRIVER only)
UID	Microsoft	User ID
PWD	Microsoft	Password
ACCT	Computer Associates	Account information, if used
CCINAME	Computer Associates	CAICCI host server name or IP address (optional, use with DRIVER only)
CCIPORT	Computer Associates	CAICCI host server port (optional, use with DRIVER only)
WAIT	Computer Associates	CAICCI reply wait timeout (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
```

Refer to the Microsoft *ODBC Programmer's Reference* for more information about calling the SQLDriverConnect function. See the online help and the chapter "[Using the ODBC Driver on Windows](#)," in this guide for information about the DriverConnect dialog. See the chapter "[Configuring the Client on Windows](#)," for more information about attribute values.

Driver-Specific Connect Options

The ODBC options that can be specified for a data source using the ODBC Administrator can also be specified during program execution using `SQLSetConnectOption` and `SQLSetStmtOption`. These options and their values are defined in `CAIDOPT.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 CA-IDMS ODBC Driver supports the following two transaction isolation levels:

SQL_READ_COMMITTED: The 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 2.5 Core and Level 1 API functions listed in the section "[API Conformance Levels](#)," earlier in this appendix. To facilitate Bulk Inserts, the CA-IDMS Server also supports the Level 2 functions `SQLParamOptions` and `SQLMoreResults`.

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 marks 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
```

where:

owner is 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 is 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)`:

Column Name	Description
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)

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 CA-IDMS JDBC driver source code contains additional information about the CA-IDMS implementation of JDBC. This HTML format documentation is installed, optionally, in the CA-IDMS Server directory and can be accessed from the CA-IDMS Server menu.

JDBC Conformance

CA-IDMS Server conforms to the JDBC 1.2 specification, which is included in the Java 1.1 (or later). Unless otherwise noted, all descriptions of JDBC in this document refer to JDBC 1.2.

API Conformance

JDBC does not define conformance levels in the same sense that ODBC does. A JDBC driver must implement all methods defined in the specification. However, the driver can return an exception, or a 0 or null value, to indicate that it cannot do what the method requires. The CA-IDMS JDBC driver implements the following methods only to satisfy the JDBC specification:

- `CallableStatement.execute`
- `CallableStatement.executeQuery`
- `CallableStatement.executeUpdate`
- `CallableStatement.getBigDecimal`
- `CallableStatement.getBoolean`
- `CallableStatement.getByte`
- `CallableStatement.getBytes`
- `CallableStatement.getDate`
- `CallableStatement.getDouble`
- `CallableStatement.getFloat`
- `CallableStatement.getInt`
- `CallableStatement.getLong`
- `CallableStatement.getObject`
- `CallableStatement.getShort`
- `CallableStatement.getString`
- `CallableStatement.getTime`
- `CallableStatement.getTimestamp`
- `CallableStatement.registerOutParameter`
- `CallableStatement.isNull`
- `Connection.setCatalog`: Catalog name not used to qualify tables
- `DatabaseMetaData.getCatalogs`

- DatabaseMetaData.getColumnPrivileges: Information not available in catalog
- DatabaseMetaData.getCrossReference
- DatabaseMetaData.getExportedKeys
- DatabaseMetaData.getImportedKeys
- DatabaseMetaData.getPrimaryKeys
- DatabaseMetaData.getProcedures
- DatabaseMetaData.getProcedureColumns
- DatabaseMetaData.getTablePrivileges
- DatabaseMetaData.getVersionColumns
- ResultSetMetaData.getCatalog: Information not available in SQLDA
- ResultSetMetaData.getSchemaName
- ResultSetMetaData.getTableName
- Statement.cancel: Client cannot cancel mainframe task
- Statement.getMoreResults: Batched statements not supported

Refer to the javadoc for more detailed information, including the specific values and exceptions returned by the CA-IDMS JDBC Driver methods.

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 CA-IDMS ODBC and JDBC drivers pass most SQL statements to CA-IDMS essentially unchanged, other than converting escape sequences into CA-IDMS equivalents. For more information about SQL conformance, refer to the appendix "[ODBC Programmer Reference](#)" in this guide.

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)
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
TIMESTAMP	TIMESTAMP

Connection Parameters

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

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

- **jdbc:idms:database:** - This format is used when the JDBC driver runs on the same machine as the native CA-IDMS SQL client interface. This is typically the case when a Java application is running on the machine where CA-IDMS Server is installed. The JDBC driver calls the native interface directly.
- **jdbc:idms://hostname:port/database:** - This format is used when the JDBC driver is running on a different machine than the native CA-IDMS SQL client interface. This is typically the case when a Java applet is running in a web browser and CA-IDMS Server is installed on the machine where the web server is running. The JDBC driver communicates with the CA-IDMS JDBC Server, which calls the native interface directly. ***hostname*** is the DNS name or IP address of the web server, and ***port*** is the IP port that was specified as the CA-IDMS JDBC Server listener.

In either case, ***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 CA-IDMS Server native 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.

DriverPropertyInfo

JDBC DriverPropertyInfo objects are analogous to the connection attributes used by the ODBC SQLDriverConnect and SQLBrowseConnect functions. For the CA-IDMS 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 data source. CA-IDMS supports the following driver properties:

Name	Description
user	User ID to sign onto CA-IDMS. Always required.
password	Password associated with the user ID. Required to connect to a secured CA-IDMS system.
account	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 appendix " Passing Accounting Information to CA-IDMS " for more information.
node	DC NODE name, identifying the CA-IDMS system containing the database, and allowing a connection to be established without defining an ODBC data source. Use of this property implies that the subname contains a DICTNAME.
via	NODE name of an intermediate CA-IDMS DC system, used to route requests to the target system. Used when a physical connection cannot be established directly to the CA-IDMS DC system containing the SQL database. Only valid if node is specified.
task	DC TASK code that invokes the internal CA-IDMS SQL server interfaces. Only valid if node is specified.
host	DNS name or IP address of the CAICCI host server, for use by the native CA-IDMS Server SQL client interface. Only valid if node is specified. Typically, the default is used.
port	IP port of the CAICCI host server. Valid only if host is specified. Typically, the default is used.

Dynamic Positioned Updates

The CA-IDMS JDBC driver supports positioned updates and deletes in dynamic SQL, when connected to a CA-IDMS Release 14.0, 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.

In order to use positioned updates and deletes, the FOR UPDATE clause must be specified in the SQL query statement. According to the JDBC specification, the SQL query must have the form:

```
SELECT FOR UPDATE ... FROM ... WHERE ...
```

CA-IDMS expects the FOR UPDATE clause at the end of the syntax:

```
SELECT ... FROM ... WHERE ... FOR UPDATE
```

The CA-IDMS JDBC driver will recognize the clause in either location, and move it to the end before passing the syntax to CA-IDMS, if necessary.

To optimize performance, the CA-IDMS JDBC driver attempts to fetch up to 100 rows at a time. Since row currency is at the last row, issuing a positioned update or delete would not have the expected effect. Specifying the FOR UPDATE clause causes the driver to fetch one row at a time.

Sample Programs

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

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 either as an application or as 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.

IdmsJcf uses the Swing classes to implement its user interface. These classes must be accessible to the application or applet, and, for best performance, should be installed on the client machine. Simply copy the **swingall.jar** file, containing the classes, to the client machine and set the CLASSPATH environment variable to point to it. For example, in Windows, set an environment variable as:

```
CLASSPATH=C:\Program Files\CA-IDMS Server\Java\classes\swingall.jar
```

When the JCF demo is installed on Windows, the **swingall.jar** file containing the swing classes is copied into the CA-IDMS Server\Java\classes directory. This allows the JCF demo to run on the machine where CA-IDMS Server is installed.

To run to the CA-IDMS JCF applet demo in a web browser, the CA-IDMS JDBC Server must be running on the web server. Since **JdbcTest** is the default data source, it may be convenient to define a data source called **JdbcTest**.

On OMVS these samples are installed in the CA-IDMS installation directory:

- */idmsdir/jcf/IdmsJcf.java*: source code, entry point and UI
- */idmsdir/jcf/JdbcTable.java*: source code, JDBC calls
- */idmsdir/jcf/idmsjcf.html*: sample web page to invoke as applet
- */idmsdir/classes/idmsjcf.jar*: compiled IdmsJcf classes

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 OS/390. 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.

On OMVS the following samples are installed in the CA-IDMS installation directory:

- */idmsdir/example/IdmsExample.java*: source code
- */idmsdir/example/example.sql*: sample SQL input file
- */idmsdir/example/example*: shell script to run IdmsExample.class
- */idmsdir/classes/IdmsExample.class*: compiled sample program

On Windows, both sample programs are optionally installed in the corresponding subdirectories of the CA-IDMS Server\Java directory. For other platforms, the Java class files are provided in **idmsjdbc.tar** and the source files in **idmssamp.tar**. Both are found in the /classes directory on the CA-IDMS Server CD.

Windows Registry Information

The registry is a database used by Windows to store system and application information. It replaces the ini files used in Windows 3.1.

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 32-bit 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, subkeys, and values. At the top level are four or five keys, two of which are used by the ODBC and CA-IDMS Servers. 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 subkey is analogous to a directory path and is specified in a similar fashion. The following are the subkeys used by ODBC and CA-IDMS:

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

Under each of these keys are subkeys 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 subkeys.

HKEY_LOCAL_MACHINE\Software\ODBC\ODBCINST.INI

This section contains information about the ODBC drivers installed on the machine, and corresponds to the ODBCINST.INI file used in Windows 3.1. The CA-IDMS Server installer program adds the information for the CA-IDMS ODBC driver using the ODBC installer DLL when the product is installed. This key contains the following information (note that the Value Name column assumes that CA-IDMS is the only installed ODBC Driver). These values are documented in more detail in the *ODBC SDK Reference*.

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 subkey, whose name is the name of the driver.
	APILevel	Driver ODBC API conformance level. For CA-IDMS, value is 1.
	ConnectFunctions	Connect functions supported by driver. For CA-IDMS, value is YYN .
	Driver	Driver DLL name and path.
	DriverODBCVer	Version of ODBC supported by driver. For CA-IDMS, value is 02.50 .
	FileExtns	Not used for CA-IDMS.
	FileUsage	Not used for CA-IDMS.
	Setup	Driver setup (configurator) DLL name and path.
	SQLLevel	Driver SQL conformance level. For CA-IDMS, value is 1 .
	UsageCount	Driver usage count
Default	Driver	Name of ODBC driver for 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. It corresponds to the ODBC.INI file used in Windows 3.1. The CA-IDMS Server configurator maintains this information using the ODBC installer DLL when the ODBC administrator is used from the Control Panel. The ODBC.INI key contains the following subkeys and values:

Subkey	Value Name	Description
ODBC Data Sources	<i>DSN</i>	Each data source has an entry. Value Name is the data source name. Value Data is the driver name. For CA-IDMS, this is CA-IDMS .
	<i>DSN</i>	Each data source has a subkey 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 CA-IDMS DBNAME table on the target CA-IDMS system. Value comes from the CA-IDMS ODBC Administrator dialog Dictionary field. Default is the first eight characters of data source name.
	Server	Specifies the user-defined server name.
Default		Default data source can contain the same values as other data source definitions.

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, and corresponds to the ODBC.INI file used in Windows 3.1.

The CA-IDMS Server configurator maintains this information using the ODBC installer DLL. The structure of the information under this key is the same as the ODBC.INI subkey of HKEY_LOCAL_MACHINE.

HKEY_LOCAL_MACHINE\Software\ComputerAssociates\CA-IDMS

This section contains information about global options, data source default options, and server definitions for all users and system services, and corresponds to the CAIDDSI.INI file used in Windows 3.1.

Data source information, previously maintained in the CAIDDSI.INI file, is now stored under the ODBC.INI keys, to support user and system data sources. The CA-IDMS Server configurator maintains this information directly.

The CA-IDMS key contains information that CA-IDMS uses to establish a connection to a CA-IDMS database anywhere in a network of CA-IDMS systems.

The CA-IDMS key can contain these subkeys:

- **Servers** subkey, to associate a server name with an ODBC driver name
- **Server** *server_name* subkey, for each server to define its database access path information
- **Options** subkey, for log settings, language settings, and path
- **Defaults** subkey, for default database settings

Servers

The Servers subkey lists all servers defined using the CA-IDMS ODBC Administrator dialog.

The Servers subkey contains the parameter **server_name**, which is always CA-IDMS, the name of the ODBC driver.

Server *Server_name*

The Server *server_name* subkey contains information describing a CA-IDMS system. It contains the following values:

Value Name	Value
Resource	<i>node_name</i>
AlternateTask	<i>task_code</i>
Node	<i>via_node_name</i>
CCIServerName	<i>cci_name</i>
CCIServerPort	<i>port</i>
WaitTimeOut	<i>wait</i>
Version	0 1

Values

server_name: Specifies a server name listed in the Servers section and referenced by a data source definition.

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.

AlternateTask=*task_code*: (Optional) Identifies an alternate task defining the resource limits and timeout values for a session. The default is CASERVER. Any task identified, whether the default or another task you define, 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 ODBC Administrator dialog. For more information on resource limits for external user sessions, see *CA-IDMS System Generation* and *CA-IDMS System Operations*.

Node=*via_node_name*: (Optional) Specifies the node with which CAICCI will establish a connection. 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 your tables does not directly communicate with CAICCI.

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 normally 1202, and typically should not be specified here.

WaitTimeOut=wait: Specifies the number of seconds CAICCI will wait 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.

Version=0 | 1: Specifies the version of the CA-IDMS Server mainframe component installed on the CA-IDMS system. The default, 0, indicates Version 4.2 or Earlier. Set this to 1 to specify Version 4.3 or Later.

Passwords can be sent to the CA-IDMS system in encrypted form, which requires the installation of an updated CA-IDMS Server mainframe component to enable the CAICCI line driver to recognize the password. Use this option to enable password encryption when connecting to a secured CA-IDMS system with the new version of the mainframe component installed.

Note: Specifying 4.3 or Later when the new version of the mainframe component has not been installed will cause connection attempts to the secured system to fail. Consult your Data Base Administrator for more information.

Options

The Options subkey contains global options. This key is updated with the values you select in the Options, Defaults, Log Options, and International tabs of the CA-IDMS ODBC Administrator dialog.

The Options subkey contains the following parameters for global options:

Value Name	Value
CloseCommit	0 1
FetchSize	<i>integer_value</i>
FetchSuspend	0 1
FetchSuspendClose	0 1
InvalidDecimal	<i>integer_value</i>
DbcsPath	<i>dbcs_path</i>
DbcsType	<i>dbcs_type</i>
cadcdc32.dll	<i>dll_name</i>
AsciiEbcDicTables	<i>translation_table_name</i>
LogFile	<i>log_file_name</i>
LogOptions	<i>log_option_values</i>
MultiThread	0 1
JdbcTrace	<i>integer_value</i>

Value Name	Value
OdbcTrace	<i>integer_value</i>
SQLTrace	<i>integer_value</i>
DtsTrace	<i>integer_value</i>
DnsTrace	<i>integer_value</i>
CmTrace	<i>integer_value</i>
FdeTrace	<i>integer_value</i>
UtilTrace	<i>integer_value</i>

Values

CloseCommit=0|1: When enabled, the CA-IDMS Server sends a COMMIT following a CLOSE when autocommit is off. The default value is 1, enabled. This option is also considered enabled when autocommit is on. The COMMIT (or COMMIT CONTINUE) is normally 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.

FetchSize=*integer_value*: Specifies the maximum size that the JDBC driver will attempt to use for a FETCH buffer. The default is 128,000. Depending on the platform and implementation of the CAICCI interface, a smaller buffer may be used. Typically, this should be left at the default setting. Specifying too large a value may cause the JVM to run out of memory.

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 1, enabled, if the CA-IDMS system is Release 14.01 or later. If the CA-IDMS system is Release 14.00 or earlier, the default is 0, because enabling this option would cause an extra network request for otherwise. This can also be specified in a specific Data Source section.

FetchSuspendClose=0|1: Causes a conditional SUSPEND to be piggybacked onto each FETCH. The default is 0, disabled. The SUSPEND is to be done only if the cursor reaches the end. This is not currently supported by CA-IDMS. This can also be specified in a specific Data Source section.

InvalidDecimal=*integer_value*: Specifies how the CA-IDMS ODBC and JDBC drivers will handle invalid packed or zoned decimal data returned in a result set column. Options are:

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

DbcPath=*dbcs_path*: Specifies the path to the DBCS translation tables, typically the direction specified when CA-IDMS Server is installed.

DbcType=*dbcs_type*: Specifies the integer value identifying the DBCS Language, as defined by the DBCS Types subkey.

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. **DbcType** must be set to a non-zero value, typically 1, to enable the use of the specified DLL.

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 ODBC Administrator dialog.

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 ODBC Administrator dialog.

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 multi-threaded access, a setting of 0 disables it. The default is 1.

XxxxTrace=*integer_value*: Specifies the flag bits used to control tracing. Computer Associates 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. Descriptions of the individual bit flags are as follows:

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

OdbcTrace (CAIDOD32.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 (CAIDQC32.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

DtsTrace (CAIDTD32.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

DnsTrace (CAIDTD32.DLL):

0x0010 // snap unconverted send data
0x0020 // snap converted send data
0x0040 // snap received data
0x0080 // snap converted received

CmTrace (CAIDTD32.DLL):

0x0001 // trace CCI and internal function calls
0x0002 // elapsed CCI call timings

FdeTrace (CAIDFD32.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

UtilTrace (CAIDUT32.DLL):

0x0001 // Trace external calls
0x0002 // Trace internal calls
0x0004 // Trace DllEntry calls

The ODBC Configurator is normally used to enable and disable tracing. Since tracing can add overhead and affect performance, it should be disabled under normal circumstances.

Defaults

The Defaults subkey contains the default ODBC options, used when adding a new data source. This key is updated from the options you select in the Defaults tab of the CA-IDMS ODBC Administrator dialog.

The Defaults subkey contains the following values for default options for new data sources. Data Source subkeys in the ODBC.INI key also contain these values.

Value Name	Value
AccessibleTables	0 1
AccountPrompt	0 1
CacheSQLTables	0 1
CatalogTable	<i>view_name</i>
CommitBehavior	0 1 2
EnableEnsure	0 1
FetchDouble	0 1
FetchRows	<i>integer_value</i>
Path	<i>path_name</i>
ReadOnly	0 1
TxnIsolation	1 2

Values

AccessibleTables=0|1: When set to 1, the ODBC driver uses the SYSCA.ACCESSIBLE_TABLES view, or another view defined by you, for the SQLTables function. A setting of 0 disables this option. This value comes 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. See the appendix "[Passing Accounting Information to CA-IDMS](#)," for more information.

CacheSQLTables=0|1: When set to 1, CA-IDMS Server caches the table list returned from an SQLTables call. A value of 0 disables this option. This value comes from the Cache SQL Tables option.

CatalogTable=view_name: Specifies the name of the view to use for the SQLTables function, if other than the default view name. This value comes from the Accessible Tables View Name field.

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. Values are:

0 specifies SQL_CB_DELETE. 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. 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. All cursors remain open, and their position is preserved. Prepared statements are not deleted. Specified by selecting **Preserve Cursors** in the Commit Behavior field.

EnableEnsure=0|1: When set to 1, CA-IDMS Server honors the ENSURE parameter of the SQLStatistics function call. A setting of 0 disables this option. This value comes from the Enable Ensure field.

FetchDouble=0|1: When set to 1, CA-IDMS converts four-byte floating point numbers to eight-byte floating point before returning them to CA-IDMS Server. This value comes from the Fetch Real as Double field.

FetchRows=integer_value: Specifies the number of database rows that CA-IDMS Server fetches at a time. The default for the ODBC driver is 100. The default for the JDBC driver is the number of rows that fit in a 30K buffer. This value comes from the Bulk Fetch Row Count field.

Path=path_name: Specifies the directory where files used by CA-IDMS Server are installed. The default is x: \Program Files\CA-IDMS Server, where x: identifies the disk drive. Note that the ODBC and CA-IDMS Server DLLs are installed in the Windows\System(32) directory.

ReadOnly=0|1: Specifies the access mode for the Windows application. A setting of 0 specifies Read Write. A setting of 1 specifies Read Only. This value comes from the Access Mode field.

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 comes from the Transaction Isolation field.

Specify these options on the Defaults tab in the CA-IDMS ODBC Administrator dialog. For an individual data source, these values can also be specified using the Options tab. In this case, the values are stored under the data source name subkey in the registry.

DBCS Types

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

Proxy

The Proxy subkey contains information used to the configure the CA-IDMS JDBC Server, and contains the following values:

Value Name	Value
Host	<i>host_name</i>
Port	<i>integer</i>
RemoteHost	<i>host_name</i>
RemotePort	<i>integer</i>
Encoding	<i>integer_value</i>
Unicode	011
Backlog	<i>integer_value</i>
TimeOut	<i>integer</i>
WaitTimeOut	<i>integer</i>
SocketTimeOut	<i>integer_value</i>
LogLevel	<i>integer_value</i>
LogTrace	<i>integer_value</i>
Trace	011
Snap	011

Values

Host=*host_name*: (Optional) 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.

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

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.

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.

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.

LogLevel=*integer_value*: Specifies the level of messages sent to the Windows NT 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 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.

TimeOut=integer_value: Specifies the number of seconds the JDBC Server will wait for a response from the CA-IDMS server. The default, 0, causes the JDBC Server to wait indefinitely.

WaitTimeOut=integer_value: Specifies the number of seconds the JDBC Server will wait for a request from the JDBC client. The default, 0, causes the JDBC Server to wait indefinitely.

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

Snap=0|1: Enables display of data buffers sent and received in the CA-IDMS log file.

Configuration File Information

On OS/390 and other UNIX platforms, CA-IDMS Server uses a configuration file to store information about database definitions, server definitions, global options, and JDBC Server options. This file is similar in format to the ini file used in Windows.

Configuration File Data

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 semi-colon.

Since many 3270 devices and emulators do not support square brackets, dollar signs or percent symbols can be used 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 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 OS/390, the configuration file is copied into a memory file when the **libutil.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 **libutil.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.
- **[Server *server_name*]**: Defines optional access information for each CA-IDMS system.
- **[Options]**: Contains other global options.
- **[Proxy]**: Contains information used by the CA-IDMS JDBC Server.

Datasource

The [*datasource_name*] section identifies CA-IDMS databases, and is specified in the JDBC URL. One is required for each database to be accessed using CA-IDMS Server. A *datasource_name* section may contain the following parameters:

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

Server=*server_name*: Specify the CA-IDMS system used to access the data. This name can be either a NODE Name or a user-defined Server name, referring to a Server *server_name* section containing additional connection information. This parameter is required.

Server

The [Server *server_name*] section contains information describing a CA-IDMS System. The Server *server_name* section may contain the following parameters:

Resource=*node_name*: (Optional) Identify 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.

AlternateTask=*task_code*: (Optional) Identify 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: (Optional) Specify 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.

WaitTimeOut=integer_value: Specify 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. A setting of 0 causes CAICCI to wait indefinitely.

Version=0/1: Identify the version of the CA-IDMS Server mainframe component installed on the CA-IDMS system.

- 0 – indicates Version 4.2 or Earlier. This is the default setting
- 1 - indicates Version 4.3 or Later

Passwords can be sent to the CA-IDMS system in encrypted form, which requires the installation of an updated CA-IDMS Server mainframe component to enable the CAICCI line driver to recognize the password. Use this option to enable password encryption when connecting to a secured CA-IDMS system with the Version 4.3 or later mainframe component installed.

Note: If you specify 4.3 or Later and the new version of the mainframe component has not been installed, connection attempts to the secured system to will fail. Consult your Data Base Administrator for more information.

Options

The [Options] section contains global options, including path information, logging options, and debugging flags. The Options section may 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.

CloseCommit=0/1: Causes a COMMIT to be sent following a CLOSE when autocommit is off. The default value is 1, enabled. This is also considered enabled when autocommit is on. The COMMIT (or COMMIT CONTINUE) is normally 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.

FetchRows=integer_value: Specify the number of database rows that CA-IDMS Server fetches at a time. The default is the number of rows that fit in a 30K buffer. This is generally the amount of data that can be transferred in a single CAICCI request.

FetchSize=*integer_value*: Specify the maximum size that the JDBC driver will attempt to use for a FETCH buffer. The default is 128,000. Depending on the platform and implementation of the CAICCI interface, a smaller buffer may be used. This should normally be left at the default setting. Specifying too large a value may cause the Java Virtual machine to run out of memory.

FetchSuspend=*0/1*: Causes a SUSPEND to be piggybacked onto each BULK FETCH, ending the IDMS-DC task. The default is 1, enabled, if the CA-IDMS system is Release 14.01 or later. Since this would cause an extra network request for Release 14.00 and earlier, the default is 0 otherwise. This can also be specified in a specific Data Source section.

FetchSuspendClose=*0/1*: Causes a conditional SUSPEND to be piggybacked onto each FETCH. The default is 0, disabled. The SUSPEND is to be done only if the cursor reaches the end. This is not currently supported by CA-IDMS. This can also be specified in a specific Data Source section.

InvalidDecimal=*integer_value*: Specify how the CA-IDMS JDBC driver handles invalid packed or zoned decimal data returned in a result set column. Choose one of the following options:

- 0 - Return error. This is the default setting
- 1 - Return NULL
- 2 - Return 0

LogFile=*log_file_path*: Specify the log file destination. The default is **/idmsdir/log/caidms.log**. When this value ends in a '/' the default name is appended to the specified path.

LogOptions=*integer_value*: Specify log options as a bitmask. The bit flags are:

- 0x0010 - Display 8-byte thread ID in trace (OS/390)
- 0x0020 - Send messages to the system log (SYSLOG). This is the default
- 0x0040 - Send messages to the system console (OS/390)

XxxdTrace=*integer_value*: Specify the flag bits used to control tracing. Computer Associates Technical 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:

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

```

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

```

DnsTrace (libtd0d.so):

```

0x0010 // Snap unconverted send data
0x0020 // Snap converted send data
0x0040 // Snap received data
0x0080 // Snap converted received

```

CmTrace (libtd0d.so):

```

0x0001 // Trace CAICCI and internal function calls
0x0002 // Elapsed CAICCI call timings
0x0004 // Snap control blocks
0x0008 // Debug #CAICCI calls on OS/390
0x0010 // Trace signon failures

```

UtilTrace (libutil.so):

```

0x0001 // Trace external calls
0x0002 // Trace internal calls

```

Proxy

The [Proxy] section contains information used to configure the CA-IDMS JDBC Server. It may contain the following parameters:

Host=*host_name*: (Optional) Specify the DNS name or IP address the JDBC will bind 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.

Port=*port*: Specify the IP port that the JDBC Server listens on for connection requests. The default is 3709.

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

RemoteHost=*host_name*: (Optional) Specify the DNS name or IP address of an intermediate JDBC Server used to forward packets to the CA-IDMS system. This value is optional.

RemotePort=*port*: Specify the IP port address of the remote host. The default value 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.

Encoding=*character_encoding_name*: Specify 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.

Unicode=*0/1*: Enable 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.

TimeOut=*integer_value*: Specify the number of seconds that the JDBC Server waits for a response from the CA-IDMS server. The default, 0, causes the JDBC Server to wait indefinitely.

WaitTimeOut=*integer_value*: Specify the number of seconds that the JDBC Server waits for a request from the JDBC client before assuming the connection has been terminated. The default, 0, causes the JDBC Server to wait indefinitely.

SocketTimeOut=*integer_value*: Specify 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 will cause the thread to block forever, and is not recommended. The default is 60 seconds.

Trace=*0/1*: Enable tracing of internal function calls. Output is written to the CA-IDMS log file.

Snap=*0/1*: Enable display of data buffers, sent and received, in the CA-IDMS log file.

LogLevel=*integer_value*: Specify 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*: Specify the level of log messages sent to the trace file.
Options are identical to the options for LogLevel.

Passing Accounting Information to CA-IDMS

This appendix describes how to use CA-IDMS Server to pass accounting information from the PC client to the backend CA-IDMS system.

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.

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

For additional information about the ODBC DriverConnect dialog, see the chapter "[Using the ODBC Driver on Windows](#)" in this guide. For more information about the ODBC SQLDriverConnect connection string, see the "[ODBC Programmer Reference](#)" appendix in this guide. For information about the JDBC DriverManager getConnection method, see the "[JDBC Programmer Reference](#)" appendix.

Using Accounting Information

The supplied accounting information is passed to the backend, 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.

Refer to the guide *CA-IDMS System Operations* for more information about programs invoked by user exits.

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 will be binary zeros if no accounting information was passed from the PC or if the frontend is not a PC.

For more information about field names in various IDMS control blocks, see the manual *CA-IDMS DSECT Reference*.

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
    
```

Configuring CAICCI for TCP/IP

The Unicenter TNG Framework for OS/390, CA-CIS, or CA90s version of CAICCI/PC includes multi-threading support when configured for TCP/IP. This version can be downloaded from the latest Unicenter TNG Framework for OS/390, CA-CIS, or CA90s tape. For your convenience, the CCI subdirectory of the CA-IDMS Server CD contains a self-extracting file to install CAICCI/PC. The following brief overview provides information about configuring and using CAICCI/PC with TCP/IP. For further information, refer to Unicenter TNG Framework for OS/390 documentation.

Installation Notes

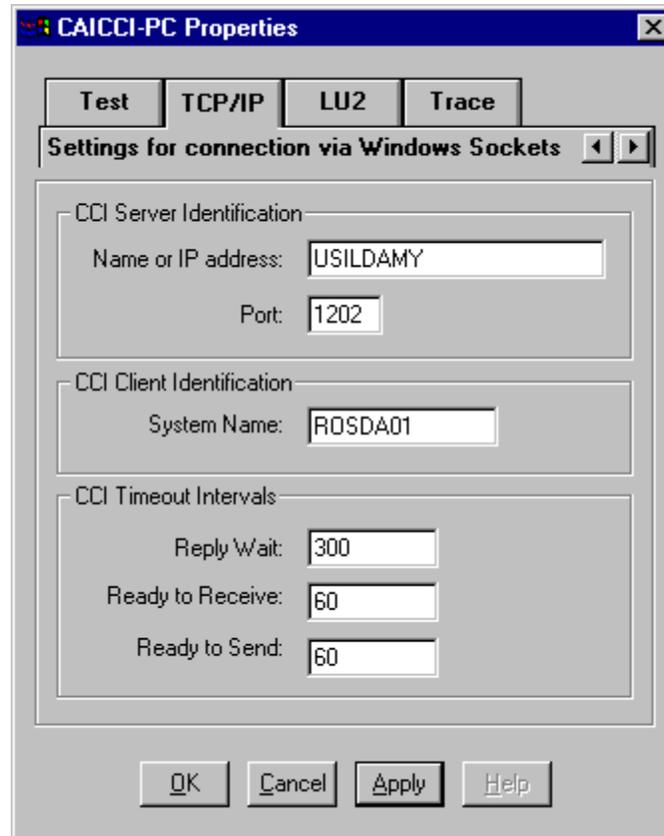
It is not necessary to reinstall CAICCI/PC when upgrading from earlier versions of CA-IDMS Server. However, after installing the Unicenter TNG Framework for OS/390 version of CAICCI/PC, multi-threaded TCP/IP support may not be enabled. When some versions of CAICCI/PC are installed, the older, single-threaded, version of the TCP/IP DLL may replace the multi-threaded version. To correct this, use the CAICCI-PC Properties dialog to modify TCP/IP parameters, and click the OK or Apply button. This restores the multi-threaded version.

Note that this does not occur when TCP/IP DLL is installed in the same directory as the CA-IDMS Server DLLs, as part of the CA-IDMS Server installation.

The CAICCI/PC DLL for TCP/IP can be installed into the Windows\System or Windows\System32 directory along with the CA-IDMS Server DLLs. This updates the run time DLL when CAICCI/PC is already installed for TCP/IP and can be selected using the Custom installation option.

Specifying Protocol Parameters for TCP/IP

Invoke the CAICCI-PC Properties dialog from the Start Menu to specify and configure a communications protocol. From the Start Menu, select CACCI NT Applications, then select CAICCI NT Configuration. To specify protocol parameters for TCP/IP, select the TCP/IP tab in the CAICCI-PC Properties dialog:



Name or IP Address: Specify the DNS name or IP address of the server acting as a gateway for CAICCI requests. Typically, you would enter an Internet address for the CAICCI server. To use a DNS name, you must be running a TCP/IP transport configured to work with a Domain Name Server, and the name you enter must be defined to the name server.

Port: Specify the port number for the CAICCI server to use. The default is 1202. When CAICCI/PC is configured to provide multi-threading support, the application can override the CAICCI server name and port.

System Name: Enter a different name to identify your PC, or to override the System Name defined by TCP/IP. By default, CAICCI uses the first eight characters of the name defined by TCP/IP. When the name is truncated the resulting ID may duplicate an ID on another machine. To avoid this, or to use a preferred ID, enter a new name in the System Name field.

After identifying the CAICCI Server, the Port number, and the CAICCI Client system name, the TCP/IP configuration is set. The CAICCI-PC Properties dialog sets the remaining parameters based on the selected software. Click OK to accept the default settings and exit this dialog.

The remaining fields in this dialog set timeout options when CAICCI/PC is installed to provide multi-threaded support.

Reply Wait: Specify the number of seconds for CAICCI to wait for a response from the other CAICCI application. When this limit is exceeded, the local application may continue to wait or assume an error and terminate the connection. Enter one of the following:

- A specific number of seconds
- -1: CAICCI/PC waits indefinitely
- 0: CAICCI/PC polls indefinitely for a response. This option is supported for compatibility with previous versions of CAICCI/PC. Using a nonzero timeout value is more efficient and results in lower processor use by the application.

Ready to Receive: Enter the time interval that CAICCI/PC will wait for the individual packets:

- Specify -1 for CAICCI to wait indefinitely
- Specify 0 to cause CAICCI to poll indefinitely

A setting of -1 is recommended in most situations.

Ready to Send: Enter the time interval for CAICCI/PC to wait to be able to send data over a connection. Values are entered as they are for the Ready to Receive option. The default is 60.

Tracing a Communications Problem

CAICCI allows you to optionally generate an internal trace file. This file is used by product support to trace communication problems. To enable the creation of this file, select the Trace tab in the CAICCI-PC Properties dialog.



Enabled: Select this option to enable the displayed trace file.

Snap Packets: Select this option to display each buffer as it is sent to, or received from, the underlying socket layer.

CCITRACE.TRC: The default filename for the trace file. This file is already in ASCII format and need not be converted.

Change: Click this button to change the name and location of the default trace file.

Data packets can be traced when CAICCI/TCP is installed to provide multi-threading support.

Important!: To return to the default trace file, repeat the above steps to return to the Diagnostic Trace dialog, and click Cancel, instead of selecting a file.

Testing the Configuration

After selecting and configuring a communications protocol, test the configuration to verify that it has correctly established contact with the host. A series of error messages are returned if the communication fails.

This option is useful because it differentiates between errors related to the communications protocol configuration and errors related to the Computer Associates solution application. This option allows you to locate errors precisely and quickly, and modify your protocol or options until the communication is successful.

To test your configuration, select the Test tab from the CAICCI-PC Properties dialog to display the CAICCI Test log.

Note: You cannot use the Test option if you have not applied your CAICCI settings. The Test tab displays the message “Pending changes have not been applied” if your settings are not in effect. To apply your settings, click the Apply button. If you click OK, you must re-access the CAICCI-PC Properties dialog to test your settings.



Click **Start** to begin the configuration test. Status messages and error messages appear in the Test log.

Exiting the CAICCI-PC Properties Dialog

After establishing and testing your communications protocol, click OK to close the CAICCI-PC Properties dialog. The selected protocol and options will remain in effect until you modify them.

Index

A

Access Mode option, 5-8, C-11

Accessible tables view, 3-6, 5-10, 5-11, C-10, C-11

Account parameter, 5-10, C-10

Accounting information
 passing to CA-IDMS, E-1
 using, E-2

Adding data sources, 5-2

Allocating datasets, 8-8

Alternate task code, defining, 3-4

API conformance levels
 Core API, A-2
 JDBC Driver, B-2
 Level 1, A-4
 Level 2, A-5
 ODBC driver, A-2

API reference, 5-22

ASCII data
 code pages for, 5-17
 translating, 5-14 to 5-19

Attributes
 supported keywords and values, A-12

B

Bit flags, C-8, D-4

Bulk Fetch Row Count, 5-9, C-7, C-11, D-3, D-4

Bulk Insert support, A-13

C

Cache SQL Tables option, 5-9, 5-11, C-11

CAICCI line, defining, 3-3

CAICCI/PC, F-1
 defining to a CA-IDMS system, 3-3
 installing, 2-1
 mainframe requirements, 2-5
 setup for, 5-1

CAICCI-PC Properties dialog, F-2

CA-IDMS
 data type mapping and JDBC, B-4
 data type mapping and ODBC, A-8
 defining, 3-2
 DriverConnect dialog, 6-1, 6-2
 isolation and lock levels, A-13
 mainframe software requirements, 2-5

CA-IDMS Server
 Administrator dialog, 5-6
 language options, 5-14 to 5-19
 Windows installation prerequisites, 4-1

CA-IDMS Server Architecture, 1-4

CA-IDMS Server for OS/390, 8-1

CA-IDMS to JDBC data type mapping, B-4
CA-IDMS to ODBC data type mapping, A-9
CAIENF facility, 2-5
CASERVER task code, 3-4, 4-1
Catalog views, 3-10
CCILINE in system generation, 3-3
CECP translation tables, 5-18
 using, 5-15 to 5-19
Character data, translating, 5-14 to 5-19
Client components for UNIX System Services
 installing, 8-2
CloseCommit option, C-7, D-3
Code page
 for host, 5-17
 for PC, 5-17
Commit Behavior options, 5-9, C-11
Communication problems
 tracing, F-4
Components of CA-IDMS Server, 1-2
Configuration file
 editing, 9-3
 environmental variables, D-1
Configuration file data, D-1
Configuration file sections, D-2
 datasource_name, D-2
 Options, D-3
 Proxy, D-5
 Server *server_name*, D-2
Configuring applications to use CA-IDMS Server, 10-1
Configuring the JDBC Server, 5-28, 9-4
Configuring the web server for applets, 7-2
Configuring the web server to use CA-IDMS Server, 10-2
Conformance levels, A-2, B-1
 API, A-2, B-2
 Core API, A-2
 Core SQL Grammar, A-7
 Extended SQL Grammar, A-7
 Level 1 API, A-4
 Level 2 API, A-5
 Minimum SQL Grammar, A-5
 SQL, A-5

Connecting dynamically to a data source, 6-2
Control-key definition, 3-9
Controlling the JDBC Server, 10-3
Core API conformance level, A-2
 supported functions, A-2
Core SQL Grammar conformance level, A-7
Country Extended Code Page
 for host languages, 5-17
 for PC languages, 5-17
 using, 5-15 to 5-19
Creating subdirectories, 8-8
Creating translation tables, 5-16
Custom Conversion DLL, 5-21
 API reference, 5-22
 developing, 5-22
 enabling, 5-21
Customizing the JDBC Server, 9-4
Customizing translation tables, 5-18

D

Data source
 defaults, 5-11
Data sources
 adding, 4-2
 adding, 5-2
 connecting to, not previously defined, 6-2
 connecting to, predefined, 6-1
 defining, 5-2, 9-3
 editing, 5-5
 maintaining, 5-5
 options, 5-11 to 5-12
 testing connection for, 5-5
 types, 5-2
Data type mapping, A-8 to A-10, B-4
Data types, CA-IDMS, A-9, B-4
Data, converting
 floating point numbers, 5-10, C-11
 language options, 5-14 to 5-19, C-8
Database access
 and page groups, 3-6
 setting up, 3-5
DBCS processing, 5-20

DBCS Types subkey, C-12

Debugging user sessions, 5-29, A-1
error messages, A-1

Default
data source options, 5-11

Defaults subkey, C-10

Defining
CAICCI line, 3-3
CA-IDMS host systems, 3-2
CASERVER task code, 3-4
data sources, using ODBC, 5-2

Double Byte Character Set processing, 5-20

DriverPropertyInfo, B-7

Driver-Specific connect options, A-13

Driver-specific data types, A-11

Dynamic positioned updates, B-8

E

EBCDIC data
code pages for, 5-17
translating, 5-14 to 5-19

Editing the configuration file, 9-3

Editing the data source definition, 5-5

Editing the JCL manually, 8-5

Editing translation tables, 5-16, 5-19

Enable Ensure database option, C-11

Enable Ensure option, 5-10, 5-11

Encoding, 9-4, C-13, D-6

Environmental variables, D-1
specifying, 9-2

Extended SQL Grammar conformance level, A-7

EXTERNAL WAIT parameter, on Task statement, 3-4

F

Fetch Real as Double database option, C-11

Fetch Real as Double option, 5-10

FetchSize option, C-7, D-4

FetchSuspend and FetchSuspendClose options, C-7, D-4

File access bits
setting, 8-9

Fixed OCCURS element definitions, 3-8

Floating point number, conversion, 5-10, C-11

Functions
Core API, A-2
Level 1 API, A-4
Level 2 API, A-5

H

Handling invalid numeric data, 3-11

HFS
allocating, 8-7
creating the installation directory, 8-7

Host code page, 5-17

Host Component, 3-1

Hyphens
in record and set names, 3-7
in record element names, 3-7

I

IDMS URL format, B-6

IEF. *See* Integrity Enhancement Facility

INACTIVE INTERVAL parameter, on Task statement, 3-4

Included tables, 5-19

Installation
Compact option, 4-2
Custom option, 4-2
Custom option, 7-1
customizing files, 8-2
customizing JCL, 8-4
loading files, 8-2
Typical option, 4-2
Typical option, 7-1

Installing CA-IDMS Server
CAICCI/PC, 2-1

- mainframe software prerequisites, 2-5
- OS/390 software prerequisites, 2-5
- Windows prerequisites, 4-1

Installing \ CA-IDMS Server on other platforms, 11-1

Installing CA-IDMS Server on Windows, 4-2

Installing the client components for UNIX System Services, 8-2

Installing the JDBC Server, 7-1

Integrity Enhancement Facility (IEF), A-6

INTERNAL parameter, on Task statement, 3-4

Invalid Decimal option, 3-11, 5-4, C-8, D-4

Isolation level, CA-IDMS, A-13

J

Java Virtual Machine. *JVM*

JCL

- customizing, 8-4
- editing, 8-5

JDBC Driver, 1-3

- conformance levels, B-1
- data type mapping, B-4
- overview of, B-1
- using on other platforms, 11-1

JDBC Driver conformance levels

- API, B-2
- SQL, B-3

JDBC Server, 1-3

- configuring, 5-28, 9-4
- controlling, 10-3
- installing, 7-1
- monitoring, 10-4
- using on Other Platforms, 11-2

JDBC to CA-IDMS data type mapping, B-5

JVM, 2-3

K

Keys and subkeys, C-1

L

Level 1 API conformance level, supported functions, A-4

Level 2 API conformance level, supported functions, A-5

Line, CAICCI, 3-3

Listener queue

- maximum length, C-13, D-6

Lock level, CA-IDMS, A-13

Log file, 5-13, 5-29, C-8, C-13, D-4, D-6

- to debug user sessions, 5-29, A-1, C-13, D-6
- writing trace information to, 5-13

Logging

- options, C-8, D-4

M

Maintaining data sources, 5-5

Messages, in log file, 5-13, 5-29, A-1, C-8, C-13, C-14, D-4, D-6

MF Version option, 5-7, C-6, D-3

Minimum SQL Grammar conformance level, A-5

Monitoring the JDBC Server, 10-4

Multithreading, 5-4, F-1

N

Navigational DML database access

- setting up, 3-5
- using SQL statements, 3-7

Network set information

- retrieving, A-14

Non-SQL database record

- accessing using SQL, 3-7
- element name
 - length, 3-9
 - transforming, 3-7
- name, transforming, 3-7

Non-SQL database set name, transforming, 3-7

O

- ODBC Administrator dialog
 - data source
 - adding, 5-2
 - maintaining, 5-5
 - options, 5-11 to 5-12
 - testing connections, 5-5
 - Data Source tab, 5-3
 - Defaults tab, 5-11
 - International tab, 5-14
 - JDBC Server tab, 5-28
 - Log Options tab, 5-13
 - Options tab, 5-8
 - Server tab, 5-6
- ODBC Driver, 1-2
 - conformance levels, A-2
 - data type mapping, A-8
 - overview of, A-1
 - supported API functions, A-2, A-4, A-5
- ODBC Driver conformance levels
 - API, A-2
 - Core API, A-2
 - Core SQL Grammar, A-7
 - Extended SQL Grammar, A-7
 - Level 1 API, A-4
 - Level 2 API, A-5
 - Minimum SQL Grammar, A-5
 - SQL grammar, A-5
- ODBC Driver Manager, 2-2
- ODBC to CA-IDMS data type mapping, A-10
- OMVS group and user, 8-6
- Options subkey, C-6
- Options, for data sources
 - default, 5-11
 - setting, 5-11 to 5-12
- OS/390
 - datasets, uploading, 8-4
 - installation, 8-1
 - Software Prerequisites, 2-5
- Other platforms, using, 11-1

P

- Page groups, and database access, 3-6
- Performance considerations for ODBC options, 5-11
- Personal computer
 - code pages, 5-17
- Positioned updates and deletes, B-8
- Prelinking object modules, 8-8
- Protocol parameters for TCP/IP, F-2
- Proxy subkey, C-12

R

- Read Committed/Uncommitted option, 5-9, C-11
- Read Only/Write access mode, 5-8, C-11
- Record
 - accessing using SQL, 3-7
 - name, transforming, 3-7
- Record element
 - unavailable, 3-8
- Record element name
 - length, 3-9
 - transforming, 3-7
- Registry information, C-1
- Remote JDBC Server, 5-29, C-14, D-6
- Retrieving network set information, A-14
- RHDCNP3S task code, 3-4
- Row, bulk fetch, 5-9, C-7, C-11, D-3, D-4

S

- Sample Programs
 - IdmsExample, B-9
 - IdmsJcf, B-8
- SERVEMAC edit macro, 8-4
- Server
 - architecture, 1-4
 - setting up, 5-6
- Server *Server_name* subkey, C-5
- Servers subkey, C-4
- Set name, transforming, 3-7
- Set up a new OMVS group and user, 8-6
- Set up for
 - CAICCI/PC, 5-1
 - CA-IDMS Server on the mainframe, 3-2
 - CASERVER task, 4-1
 - servers, 5-6
- Setting default data source options, 5-11
- Setting file access bits, 8-9
- Setting up
 - catalog views, 3-10
 - database access, 3-5
 - SQL access, 3-5
 - SQL to non-SQL database access, 3-6
- Socket
 - reading data from, C-13, D-6
- Software requirements, mainframe
 - CA-IDMS, 2-5
- SQL conformance levels, A-5
- SQL Conformance Levels, B-3
 - Core SQL Grammar, A-7
 - Extended SQL Grammar, A-7
 - Minimum SQL Grammar, A-5
- SQL Data Sources dialog, 6-1
- SQL database access
 - sample CA-Visual Realia COBOL program, E-1
 - setting up, 3-5, 3-6

- SQL synonym, for non-SQL records, 3-8
- SQLDriverConnect connection string format, A-11

- Subkeys
 - DBCS Types, C-12
 - Defaults, C-10
 - Options, C-6
 - Proxy, C-12
 - Server *Server_name*, C-5
 - Servers, C-4

- Supported attribute keywords, A-12

- Supported attribute values, A-12

- SYSCA.ACCESSIBLE_ SCHEMAS, 3-10

- SYSCA.ACCESSIBLE_ TABLES, 3-6, 5-10, 5-11, C-10, C-11

- SYSCA.ODBC_INDEX, 3-10

T

- Tables, accessible, 3-6, C-10

- Task code, 5-7, 6-3, C-5, D-2
 - defining, 3-4
 - for set up, 4-1

- Testing
 - CAICCI settings, F-5
 - connections to data sources, 5-5

- Trace options, 5-13, 5-29, C-8, C-14, D-4, D-7, F-4

- Tracing a communications problem, F-4

- Transaction isolation levels, A-13

- Transaction Isolation option, 5-9, C-11

- Translation editor
 - keystrokes, 5-19

- Translation tables, 5-15 to 5-19, C-8
 - CECP, 5-15 to 5-19, 5-18
 - creating, 5-16
 - customizing, 5-18
 - editing, 5-16
 - for non-character-based languages, 5-15 to 5-19
 - included, 5-19
 - saving, 5-19

U

- Unicode, 9-4, C-13, D-6
- Uninstalling previous versions, 4-1
- URL format, B-6
- Use Accessible Table View option, 3-6, C-10
- Use Accessible Tables View Name option, 5-10, 5-11, C-11
- User session
 - debugging, 5-29, A-1
- Using SMP/E to install executable modules, 8-8
- Using the JDBC Driver
 - on other platforms, 11-1
- Using the JDBC Server
 - on other platforms, 11-2
 - on Windows 95, 7-2
 - on Windows 98, 7-2
 - on Windows Millennium, 7-2
 - on Windows NT, 7-2
- Using the JDBC Server on other platforms, 11-2

V

- Version option, 5-7, C-6, D-3
- View, SYSCA.ACCESSIBLE_TABLES, 5-10, 5-11, C-11
- View, SYSCA_ACCESSIBLE_TABLES, 3-6, C-10

W

- Windows 95
 - using the JDBC Server on, 7-2
- Windows 98
 - using the JDBC Server on, 7-2
- Windows installation, 4-2
- Windows Me
 - using the JDBC Server on, 7-2
- Windows NT
 - Using the JDBC Server on, 7-2
- Windows registry, C-1

