CONNX 14.5 User Reference Guide

Table of Contents

Chapter 1 - Preface

Introduction

CONNX is a unique client/server connectivity programming toolset that makes it possible to use computers in real-time interactive operation with many databases. The CONNX data access engine is unique in that it not only provides access to the databases, but it presents them as one enterprise-spanning relational data source. CONNX also offers additional security, metadata management, enhanced SQL capability, views, heterogeneous joins, bidirectional data conversion, and enables read/write access to the data.

Such technology can be used in data warehousing, data integration, application integration, e-commerce, data migration, and for reporting purposes. The technology also has a place within companies seeking to make use of disparate data sources, that need to web-enable their data, or that have older applications storing mission-critical information.

CONNX includes the following components:

- CONNX Data Dictionary (CDD)
- CONNX ODBC Driver
- CONNX Unix ODBC Driver
- CONNX OLE RPC Server (Not implemented for CONNX and VSAM)
- CONNX Host Data Server (RMS, VSAM (Implemented as CICS/C++ TCP/IP Listener/Server), C-ISAM, DISAM, Micro Focus, Rdb, and DBMS)
- CONNX JDBC Driver (Thin Client)
- CONNX JDBC Server
- CONNX JDBC Router

CONNX supports the following host database platforms:

Databases	Operating Systems
Digital RMS, Oracle Rdb, Codasyl DBMS	OpenVMS/VAX, OpenVMS/Alpha, OpenVMS/Itanium (RMS only)
Oracle Databases	Unix, Windows, OpenVMS/VAX, Linux
DataFlex	Windows, Novell NetWare, Linux
POWER*flex*	Windows, Novell NetWare, Linux
IBM DB2	See System Requirements table for detailed information.
SQL Server	Windows
Sybase	Windows, Unix
Informix	Windows, Unix
OLE DB/ODBC-compliant databases	Platforms supported by data source
JDBC-compliant databases	Platforms supported by data source
Adabas, as a product also known as the Adabas SQL Gateway	IBM z/OS, Windows, Unix platforms, including HP-UX, AIX, Solaris, and Linux
VSAM	See System Requirements table for detailed

	information.
C-ISAM	HPUX, SCO, Solaris, AIX, Linux 7.2 (Unix) Windows
DISAM	HPUX, SCO, Solaris, AIX, Linux 7.2 (Unix) Windows
Micro Focus	HPUX, Solaris, AIX, Linux 7.2 (Unix) Windows
IMS	See System Requirements table for detailed information.
Redshift	Solaris, Linux 7.2 (Unix), Windows

*References to VAX in this manual also apply to the Compaq Alpha.

Related Topics

- CONNX Compatibility
- CONNX Components
- CONNX System Requirements
- The CONNX Data Dictionary
- The CONNX ODBC Driver
- ODBC Driver Definition
- ODBC Driver Architecture
- Features of the ODBC Driver
- CONNX OLE RPC Server
- CONNX.Connect
- JDBC Driver Definition
- JDBC Driver Architecture

The CONNX Data Dictionary (CDD)

The CONNX Data Dictionary (CDD) is a repository of information describing the data tables and fields in the accessed databases, including security. The CDD contains the metadata about the source information and provides easy maintenance of the metadata, views, and integrated security. The CDD:

- Can reside on the client computer, a shared server disk, or in a Pathworks area on an OpenVMS system.
- Describes the structures of the tables or files being accessed.
- Enables multiple views of the same data.

The types of CDD objects that can be accessed include the following:

- Adabas data sources (z/OS, VSE, Windows, Unix [AIX, Linux, HP-UX, Solaris]platforms)
- Disparate data sources, including databases, flat files, and other types, including both Unix and Windows
- Tables that reside within each data source
- System tables for relational data sources
- Native Rdb, Oracle, DB2, or OLE DB views (A view is an SQL statement that defines the relationships between one or more tables or that specifies the criteria for the returned resultset.)
- CONNX views that combine tables from one or more data sources and which are created in the CDD
- Native Oracle or DB2 stored procedures
- DB2 packages
- DB2 static SQL statements
- VSAM data sources (both Unix and Windows platforms)
- C-ISAM data sources (both Unix and Windows platforms)
- DISAM data sources (both Unix and Windows platforms)
- Micro Focus data sources (both Unix and Windows platforms)
- IMS data sources
- Redshift database (Windows, Solaris, Linux (Unix) platforms)

The CDD contains the metadata information about each data source and provides a graphical user interface for easy maintenance of the metadata, integrated security, and views. The CDD is required in order to use the CONNX ODBC driver.

Once a CDD is created, it should be used for all additional table definitions, if these additional tables are to be joined with existing tables.

The CONNX ODBC Driver

The CONNX ODBC Driver is a dynamic-link library, administered by the ODBC data source administrator. Applications can access data located in remote systems through the ODBC driver. The CONNX driver processes the ODBC function calls, submits requests to the appropriate data source, and then returns the results. The CONNX ODBC Driver:

- Is tightly coupled with the CONNX Data Dictionary.
- Uses Structured Query Language (SQL) as the standard for accessing information.
- Enables the use of off-the-shelf ODBC-compliant reporting and development tools.
- Supports Unicode and ANSI data types.

Related Topics

ODBC Driver Definition
ODBC Driver Architecture
CONNX ODBC Conformance
Features of the ODBC Driver

The CONNX JDBC Driver

The CONNX JDBC driver implements the JDBC specification developed for use with CONNX to enable connectivity to all types of databases. Used with the CONNX Data Dictionary (CDD), the JDBC driver provides a means of using many popular querying tools and application development tools. The CONNX JDBC driver works with JDBC-compliant software, which increases its flexibility when used by companies with a wide range of front-end applications and database types. The CONNX JDBC interface enables applications to access data in database management systems using the JavaSoft JDBC API to connect to the databases. The CONNX JDBC Driver:

- Is tightly coupled with the CONNX Data Dictionary.
- Uses Java as the standard for accessing information.
- Enables the use of off-the-shelf JDBC-compliant reporting and development tools.
- Supports Unicode and ANSI data types.

Related Topics

CONNX JDBC Server Definition

JDBC Driver Architecture

CONNX JDBC Driver Architecture

The CONNX OLE RPC Server

The CONNX OLE RPC Server allows the user to make remote procedure calls (RPC) from any programming language that supports OLE 2.0. Supported database systems include:

- RMS
- Oracle Rdb
- Codasyl DBMS
- VSAM
- IMS
- C-ISAM (for Unix only)

Supported tools include:

- Microsoft® Access
- Microsoft® Excel
- Microsoft® Visual Basic®
- Microsoft® Visual C++®

The CONNX Host Data Server (RMS, Oracle Rdb, DBMS, VSAM, and C-ISAM/Unix only)

The CONNX Host Data Server is a full-featured data server that translates SQL requests into native database requests. The CONNX ODBC driver makes the translation service transparent to the end user. The primary features of the driver include the following:

- SQL access to target sources
- Complete user-level and group-level security
- Low memory and disk resource utilization
- Complete file and/or table security
- RPC (Remote Procedure Call) support

Related Topics


CONNX Security Overview


CONNX Remote Procedures

System Requirements

The following table lists CONNX system requirements, including hardware requirements, for applicable databases and their operating systems.

Adabas SQL Gateway (CONNX for Adabas)	
Windows	
Hardware	N/A
Network	TCP/IP
Operating System	Windows
Memory	40 MB
Hard Drive	N/A
IBM z/OS	
Hardware	z/OS - FTP server required for installation only
Network	TCP/IP (OE stack)
Operating System	z/OS
Memory	
Hard Drive	
Security	OSS segment must be defined for the user ID in the security system (RACF / ACF2 / Top Secret) or it must be available by default.
VSE	
Hardware	IBM VSE
Network	TCP/IP or Barnard TCP/IP Communications Stack
Operating System	VSE
Memory	
Hard Drive	
Security	
HP-UX	
Hardware	Processor: PA-RISC, 512 MB Disk space: Installing the optimized version of Adabas requires approximately 45MB. An additional 30 MB are required if a demo database is created during installation. An additional 60 MB are required if you install the trace version of Adabas. These figures do not include disk space requirements for other

	databases that you will create. CD-ROM drive: Required.
Software	HP-UX 11.0 (64-bit) or HP-UX V11.11i (64-bit) or HP-UX 11.11 (32-bit) Remote Access: ENTIRE-NETWORK 2.1.1 or above is required for remote access from other machines.
Solaris	
Hardware	Processor: UltraSPARC Memory: 512 MB Disk space: Installing the optimized version of Adabas requires approximately 45MB. An additional 30 MB are required if a demo database is created during installation. An additional 60 MB are required if you install the trace version of Adabas. These figures do not include disk space requirements for other databases that you will create. CD-ROM drive: Required.
Software	Operating System: SUN Solaris Version 7 or SUN Solaris Version 8 Remote Access: ENTIRE-NETWORK 2.1.1 or above is required for remote access from other machines.
AIX	
Hardware	Processor: IBM e-Server P-Series or RS/6000 Memory: 512 MB Disk space: Installing the optimized version of Adabas requires approximately 45MB. An additional 30 MB are required if a demo database is created during installation. An additional 60 MB are required if you install the trace version of Adabas. These figures do not include disk space requirements for other databases that you will create. CD-ROM drive: Required.
Software	Operating System: IBM AIX5L Version 5.1 system maintenance level 2 (64-bit) or Version 5.2 Remote Access: ENTIRE-NETWORK 2.1.1 or above is required for remote access from other machines.
Linux	
Hardware	Processor: Intel Memory: 512 MB Disk space: Installing the optimized version of Adabas requires approximately 45MB. An additional 30 MB are required if a demo database is created during installation. An additional 60 MB are required if you install the trace version of Adabas. These figures do not include disk space requirements for other databases that you will create. CD-ROM drive: Required.
Software	Operating System: Red Hat 7, Red Hat 9, Suse, Linux 390 Remote Access: ENTIRE-NETWORK 2.1.1 or above is required for remote

	access from other machines.

RMS (any version)	
Hardware	Compaq VAX Server, Compaq AlphaServer, Itanium
Network	TCP/IP, DECnet, Phase IV and above
Operating System	OpenVMS
Memory	12 MB VAX, 32 MB Alpha
Hard Drive	20,000 blocks available (10 MB)
Codasyl DBMS	
Hardware	Compaq VAX Server, Compaq Alpha Server
Network	TCP/IP, DECnet Phase IV and above
Operating System	OpenVMS
Memory	12 MB VAX, 32 MB Alpha
Hard Drive	20,000 blocks available (10 MB)
Oracle Rdb	
Hardware	Compaq VAX Server, Compaq Alpha Server
Network	TCP/IP, DECnet Phase IV and above
Operating System	OpenVMS, Unix
Memory	12 MB VAX, 32 MB Alpha
Hard Drive	20,000 blocks available (10 MB)
Oracle	
Hardware	Compaq VMS Server (VAX or Alpha), Personal Computer (Intel)/Alpha Sun/Unix Workstation
Network	SQLNet 2.x
Operating System	OpenVMS (any version), Windows, Unix/Linux (any version)
Memory	N/A
Hard Drive	N/A
DataFlex version 2.3 and above	
Hardware	Personal Computer (Intel)/Alpha, Unix Workstation
Network	Any file-sharing protocol
Operating System	Windows, Novell Netware, Linux

Memory	N/A
Hard Drive	N/A
POWER*flex*	
Hardware	Personal Computer (Intel)/Alpha, Unix Workstation
Network	Any file-sharing protocol
Operating System	Windows, Novell Netware, Linux
Memory	N/A
Hard Drive	N/A
OLE DB Providers (Sybase Informix, Microsoft SQL Server, Microsoft Access)	
Operating System	Windows, MDAC 1.5 or later (See Notes below.)
ODBC Providers (Must support ADO and be fully ODBC Level 2 compliant.)	
Operating System	Windows, MDAC 1.5 or later (See Notes below.)
C-ISAM	
Hardware	HPUX, SCO, Solaris, AIX, Linux
Network	TCP/IP
Operating System	HPUX 10.2+, HPUX64, SUN OS 5.6, SUN OS 5.7+, SCO Openserver release 5, AIX 4.3+, AIX 4.2, Linux 7.2 and 9, or Tru 64 (Unix) Windows
Memory	N/A
Hard Drive	1 MB
DISAM	
Hardware	HPUX, SCO, Solaris, AIX, Linux
Network	TCP/IP
Operating System	HPUX 10.2+, HPUX64, SCO Server, SUN OS 5.6, SUN OS 5.7+, AIX 4.3+, AIX 4.2, Linux, or Tru 64 (Unix) Windows
Memory	N/A
Hard Drive	1 MB
Micro Focus 2.2	

Hardware	HPUX, Solaris, AIX, Linux
Network	TCP/IP
Operating System	HPUX 10.2+, HPUX64, AIX 4.3+, AIX 4.2, Linux, or Tru 64 (Unix), SUN 5.8 and 5.9 with Micro Focus 4.0 SP1 only Windows
Memory	N/A
Hard Drive	1 MB
Redshift	
Hardware	Personal Computer (Intel)/Alpha, Solaris, Linux(Unix)
Network	TCP/IP
Operating System	Linux, SUN 5.8 and 5.9, Windows
Memory	N/A
Hard Drive	N/A

DB2 Product	Operating System	Network
DB2/6000; DB2 UDB for AIX	AIX 4.3 and above	TCP/IP and SNA/LU 6.2
DB2/MVS V4R1 and above;	MVS	SNA/LU 6.2 only
DB2 UDB for z/OS and OS/390	z/OS and OS/390	TCP/IP and SNA/LU 6.2
DB2/400 V3R1 and above	OS/400	SNA/LU 6.2 only
DB2/400 V4R2 and above; DB2 UDB for iSeries	OS/400 and iSeries	TCP/IP and SNA/LU 6.2
DB2 UDB Enterprise Server Edition	Windows	TCP/IP and SNA/LU 6.2
DB2 UDB for Linux Enterprise Server Edition	Linux	TCP/IP

VSAM Product	Operating System	Supported File Types	Network Software	CICS Version/Release
CONNX for CICS/VSAM	z/OS	VSAM	TCP/IP V3R2 and above	V4R1 or TX 1.x and above
CONNX for VSAM /	z/OS	VSAM / QSAM /	TCP/IP V3R2 and	N/A

QSAM / PDS		PDS	above	
CONNX for CICS/VSAM	VSE 2.3 and below	VSAM	TCP/IP (CSI / IBM) or Barnard TCP/IP Stack	V2R3 and below
CONNX for CICS/VSAM	VSE 2.4 and above	VSAM	TCP/IP (CSI / IBM) or Barnard TCP/IP Stack	TS 1.1.1 and above

IMS Product	Operating System	Supported File Types	Network Software	CICS Version/Release
CONNX for IMS	z/OS	IMS	TCP/IP V3R2 and above	V4R1 or TX 1.x and above

Note: *CONNX has been tested or certified with the following TCP/IP software products on OpenVMS: UCX 3.0 and above, Multinet, TCPware, and Pathworks.*

Note: *OLE DB/ODBC providers use third-party data providers which have their own hardware and network requirements and are installed on the client machine.*

Important: *The server requirements are dictated by the third-party driver selected for operation.*

Important: *Place the license files on a server that allows for the same number of simultaneous connections as the license count purchased.*

CONNX requires that all computers using the CONNX ODBC driver use the same license path for all licensed databases. Do not choose a local hard drive for a license path unless you are using CONNX on a single computer or if you are using CONNX over a remote TCP/IP connection or Remote Active Service (RAS). If two or more computers have a license path that points to a local hard drive, the connection to CONNX may be rejected.

In previous versions of CONNX, if an N-tier license was being run on a machine that was connected to a remote desktop using an application such as PC Anywhere, Terminal Services, or Remote Desktop, an error appeared stating that the license count had been exceeded. This has been changed so that the license now reverts automatically to a database license.

CONNX periodically checks to make sure all computers using CONNX are configured to the same license path. If CONNX determines that one or more computers are not configured to the same license path, the connection may be denied.

Unix Client System Requirements

PC Linux Client System Requirements	
Hardware	Processor: Intel Pentium 4 class Memory: 514 MB
Operating System	**Any Linux OS which supports Linux Kernel 2.6**, for example, Fedora Core Release 4, RedHat Enterprise Linux, version 3, or SUSE Professional 9.3 Linux.
Free Hard Disk Space	50 MB
Software - ODBC Driver Manager	Any ODBC Driver Manager

Solaris Client System Requirements	
Hardware	Processor: UltraSPARC; Memory: 512 MB
Operating System	Sun OS 5.7 or above
Free Hard Disk Space	50 MB
Software - ODBC Driver Manager	Any ODBC Driver Manager

AIX Client System Requirements	
Hardware	Processor: IBM e-Server P-Series or RS/6000; Memory: 512 MB
Operating System	AIX 5.xOperating System: IBM AIX 5L Version 5.1, system maintenance level 2 (64-bit) or Version 5.2
Free Hard Disk Space	50 MB
Software - ODBC Driver Manager	Any ODBC Driver Manager

HP-UX Client System Requirements	
Hardware	Processor: PA-RISC, 512 MB
Operating System	HP-UX 11.0 (64-bit) or HP-UX V11.11i (64-bit)
Free Hard Disk Space	50MB
Software - ODBC Driver Manager	Any ODBC Driver Manager

CONNX Client and Web Server Requirements

Requirement	Minimum	Recommended
Processor	Intel Core or compatible processor	Intel Core or compatible processor
Memory	2 GB	8 GB
Free Hard Drive Space	25 GB	50 GB
Operating System and Server	Windows Server 2012/Windows 10 or above, Microsoft IIS 8 or above	Windows Server 2012/Windows 10 or above, IIS 8 or above
Software (3-tier enterprise)	Any web browser that supports HTML 3 or later	Any web browser that supports HTML 3 or later
Network Connectivity	TCP/IP (Winsock 1.1-compliant)	Microsoft TCP/IP (Winsock 2.9-compliant)

CONNX Components

CONNX is a universal data connectivity programming toolset that provides real-time access to multiple disparate databases.

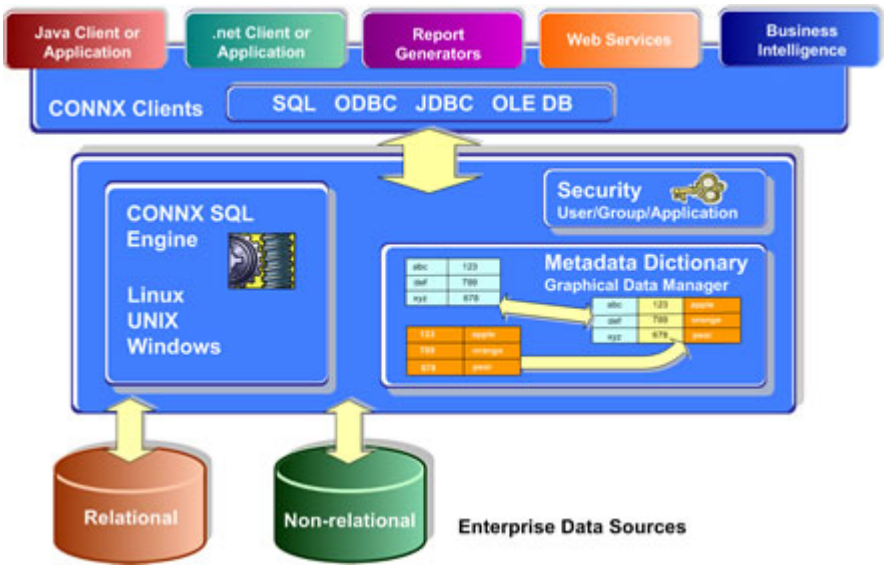
The table below illustrates the databases and relative operating systems supported by CONNX. All of the databases shown can be accessed on Windows and Unix client PCs.

CONNX-compatible Databases	Windows	Novell Netware	Solaris/SPARC	HPUX	IBM-AIX	OS/400	OS/390	MVS/ESA	z/OS	VSE	LINUX	VMS/VAX	VMS/Alpha	Open VMS/Integrity	CICS (z/OS, OS/390, VSE)
Oracle	X		X	X	X		X		X		X	X	X	X	
DataFlex	X	X									X				
POWER*flex*	X	X									X				
SQL Server	X														
Informix	X		X	X	X						X				
Sybase	X		X	X	X						X				
DB2	X		X	X	X	X	X	X	X		X				
Adabas	X		X	X	X		X	X	X	X	X		X		X
VSAM							X	X	X	X					X
IMS							X	X	X						
C-ISAM	X		X	X	X						X				
DISAM	X		X	X	X						X				
Micro Focus	X		X	X	X						X				
RMS												X	X	X	
Oracle Rdb												X	X	X	
Oracle Codasyl DBMS												X	X	X	
CONNXStore	X														
PostGreSQL	X		X	X	X						X				
OLE DB Provider Database*	X														
ODBC Provider Database*	X														
MySQL	X		X	X	X						X				
Redshift	X		X								X				

*CONNX supports any SQL-based ODBC data source or OLE DB provider.

With the CONNX product, data from any of the above database types can be retrieved and updated. The CONNX approach to data access is to present all of the different data sources as a single relational database. CONNX currently consists of seven main components: the CONNX Data Dictionary, the CONNX ODBC Driver, the CONNX OLE RPC Server, the CONNX Host Data Server (for RMS, Oracle Rdb, C-ISAM, VSAM, and DBMS), the CONNX JDBC Driver, the CONNX JDBC Server, and the CONNX JDBC Router.

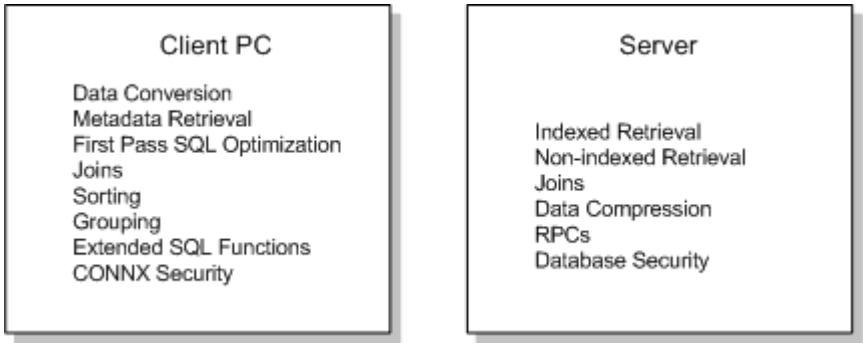
CONNX Architecture



CONNX has a distributed SQL engine, which means that the work of processing queries is distributed between the client and the server. Most of the CPU-intensive query processing, such as data conversion and sorting, is performed on the client computer. All of the data retrieval is performed on the server, although, in CONNX for DataFlex, all processing is done on the client computer.

The CONNX distribution of labor, except that for DataFlex, is shown in the following diagram:

CONNX Distribution of Labor



This distributed architecture has several advantages:

- When performing joins, significantly less data is sent across the network because no duplicates are transmitted.
- The workload on the server is minimal, because CPU-bound tasks are moved to the client, resulting in a reduction of load on the mainframe.
- When several users are issuing queries simultaneously, the CPU power of each client computer is utilized in addition to that of the server, resulting in true parallel processing. This makes CONNX highly scalable to a large enterprise.

CONNX Compatibility

In general, CONNX works with any ODBC-compliant application. Specifically, the programs listed here successfully utilize CONNX as a data access connectivity tool.

OLE DB (Windows only)	OLE Automation (RPC)	ODBC	JDBC	Application
	√			Any application that supports OLE automation
		√	√	Any JDBC-compliant application
√	□	√		Any ODBC- or OLE DB- compliant application
				Apache Web Server
√	√	√		Borland C++
√		√	√	Borland Delphi
		√	√	Borland JBuilder
	√	√		Cognos Impromptu
√	√	√		Crystal Reports
	√	√		Dharma ODBC Integrator
□		√		GIS (Geographical Information Software)
√	√	√	√	Internet Information Server (IIS)
	√	√		JetForms
√	√	√		Microsoft Access
	√	√		Microsoft Excel (MSQuery)
		√		Microsoft SQL Server (linked server technology)
		√		Microsoft Transaction Server (MTS)
√	√	√		Microsoft Visual Basic
√	√	√		Microsoft Visual Basic for Applications (VBA)
√	√	√		Microsoft Visual C++, Microsoft Visual Studio
√	√	√		Microsoft Visual Studio .NET
		√	√	Netscape (iPlanet) Enterprise Server
	√	√		Oracle Developer/Designer 2000
		√		Oracle Discover
		√		Oracle Heterogeneous Services
√	√	√		PowerBuilder

√	√	√		Paradox for Windows
		√		Sagent
		√	√	Star Office
			√	Sun Forte
			√	Sun Netbeans
			√	Sun Netra Web Server
	√	√		Visual FoxPro for Windows

64bit and 32bit port numbers for windows services

The following is a list of 32 and 64 bit components and the default TCP/IP ports they listen on:

Component Name	64-bit Version	32-bit Version
Enterprise Server Service	6502	6500
JDBC Server Service	7502	7500
License Server Service	Not Applicable	7501
DataSync Rest Server	Not Applicable	9500

If CONNX is installed in an environment where a firewall is present, these ports need to be opened.

Working with CDDs

Creating CDDs

The distributed architecture of the CONNX Data Dictionary administration tool enables users to access and leverage all of their data quickly and efficiently through easily created CDDs.

Just a few of the capabilities and functionality available in the CONNX Data Dictionary Manager are included in the following list:

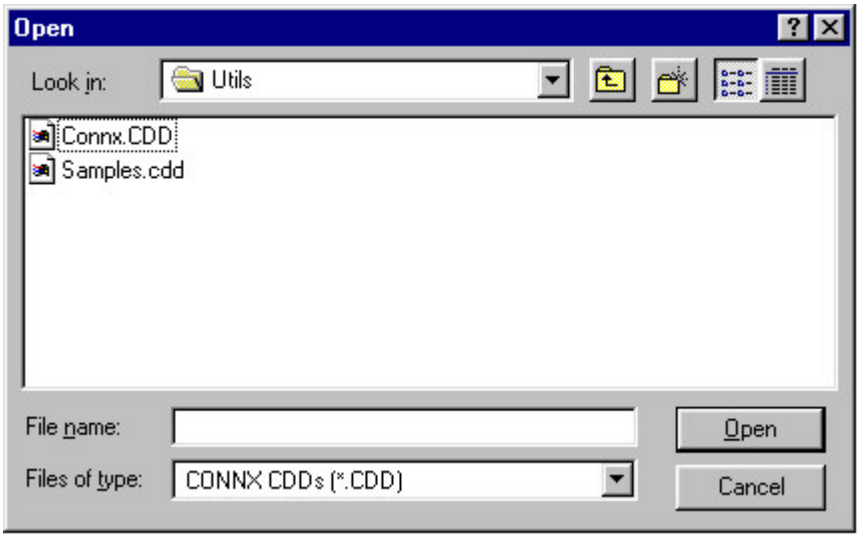
- Create new CDDs
- Import CDD entries
- Open existing CDDs
- Add columns to file entries
- Create file entries manually
- Document existing CDDs
- Save CDDs
- Add multiple tables
- Manage records
- Enable catalog support
- View database information
- Update statistics
- Link CDDs
- Browse CDDs
- Filter datatypes

Related Topics

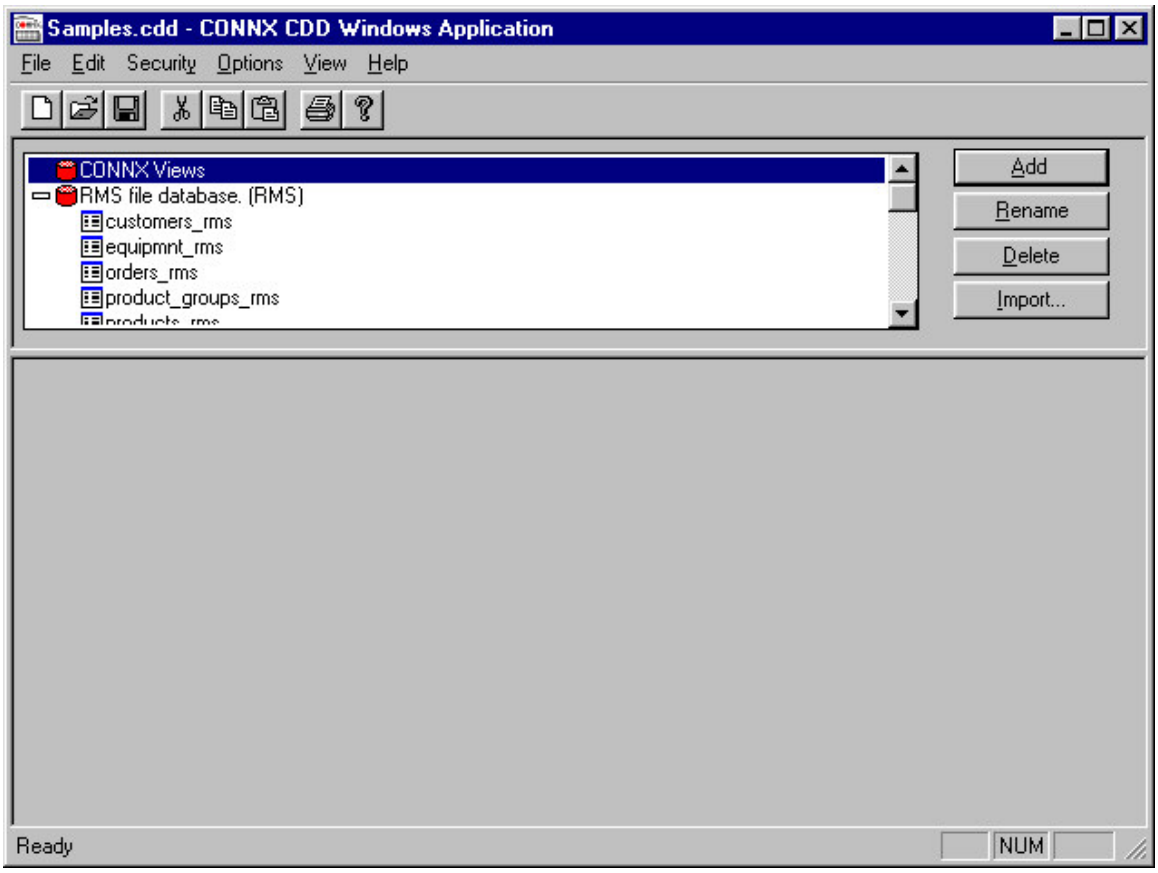
» To create a CDD
» Importing Existing Table Definitions
» To open an existing CDD
» To save a data dictionary entry
» To add a new user
» To add a new group
» To add security to tables and columns
» Managing Applications
» To add an application

To create a CDD

1. Click the **Start** button, and then point to **All Programs.** Point to **CONNX Driver** and then click **CONNX Data Dictionary.**
2. The **Open** dialog box appears. Click the **Cancel** button.



3. The CONNX Data Dictionary Manager window appears. You can add or import CDD objects (databases, tables, fields, views, and stored procedures) by clicking the **Add** or **Import** buttons in the CONNX Data Dictionary Manager window and then choosing an available object.



Related Topics

- To open an existing CDD
- To save a data dictionary entry
- To import from an RMS text file import specification

Importing Existing Table Definitions

The following lists table definitions that can be imported into the CONNX Data Dictionary.

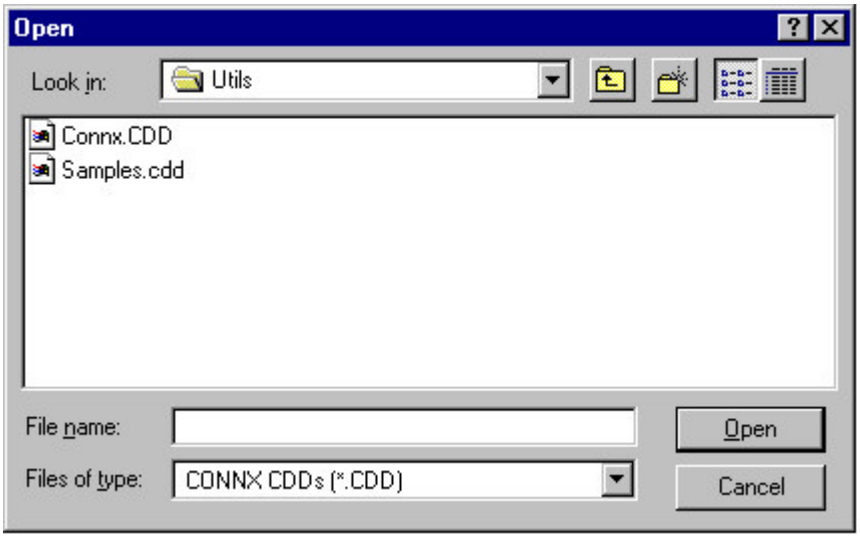
- COBOL FD (File Definition)
- Codasyl DBMS database
- DataFlex
- POWER*flex*
- DIBOL
- Formatted DDL (Data Definition Language)
- Powerhouse PDL (Powerhouse Definition Language)
- Rdb database
- SCT COBOL FD
- Specification text file (RMS)
- VAX, ALPHA, or Itanium (RMS only) CDD
- OLE DB- and ODBC-compliant data sources
- DB2 data sources
- Oracle data sources
- VSAM data sources
- IMS data sources
- Sybase data sources
- Informix data sources
- Adabas data sources

Once you import the metadata and define the table properties, you must save the CDD and create an ODBC data source name. The data in the tables can then be accessed and manipulated. Depending on the front-end application chosen and the types of databases that exist in your system, you can use the imported metadata to create new database objects, or to update or incorporate data quickly and efficiently.

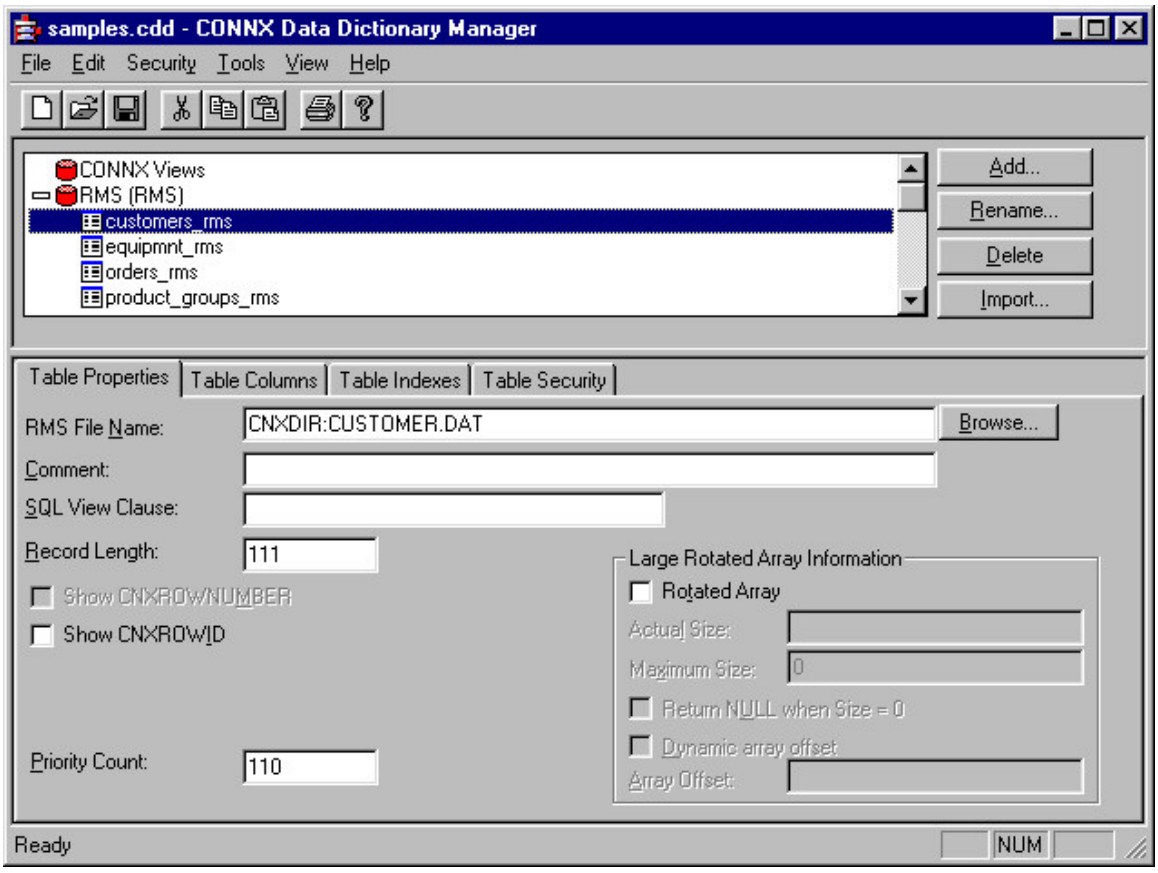
Once table definitions are imported, column, field, and row names can be changed independently of their source data. Note that any changes made in the table metadata imported into the CDD do not affect the source data in any way.

To open an existing CDD

1. On the **File** menu, click **Open** in the CONNX Data Dictionary Manager window.
2. Double-click an existing CDD in the **Open** dialog box or browse the directories shown in the **Look in** list box.



3. Use the scroll bar to the right of the CDD directory pane in the CONNX Data Dictionary Manager window to view the tables. Click on a database object to view its associated tables, views, or stored procedures.

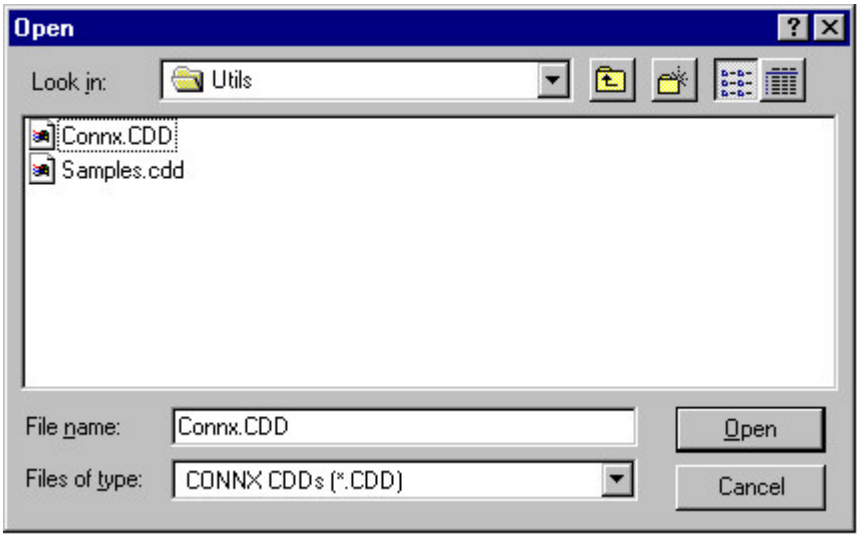


To save a data dictionary entry

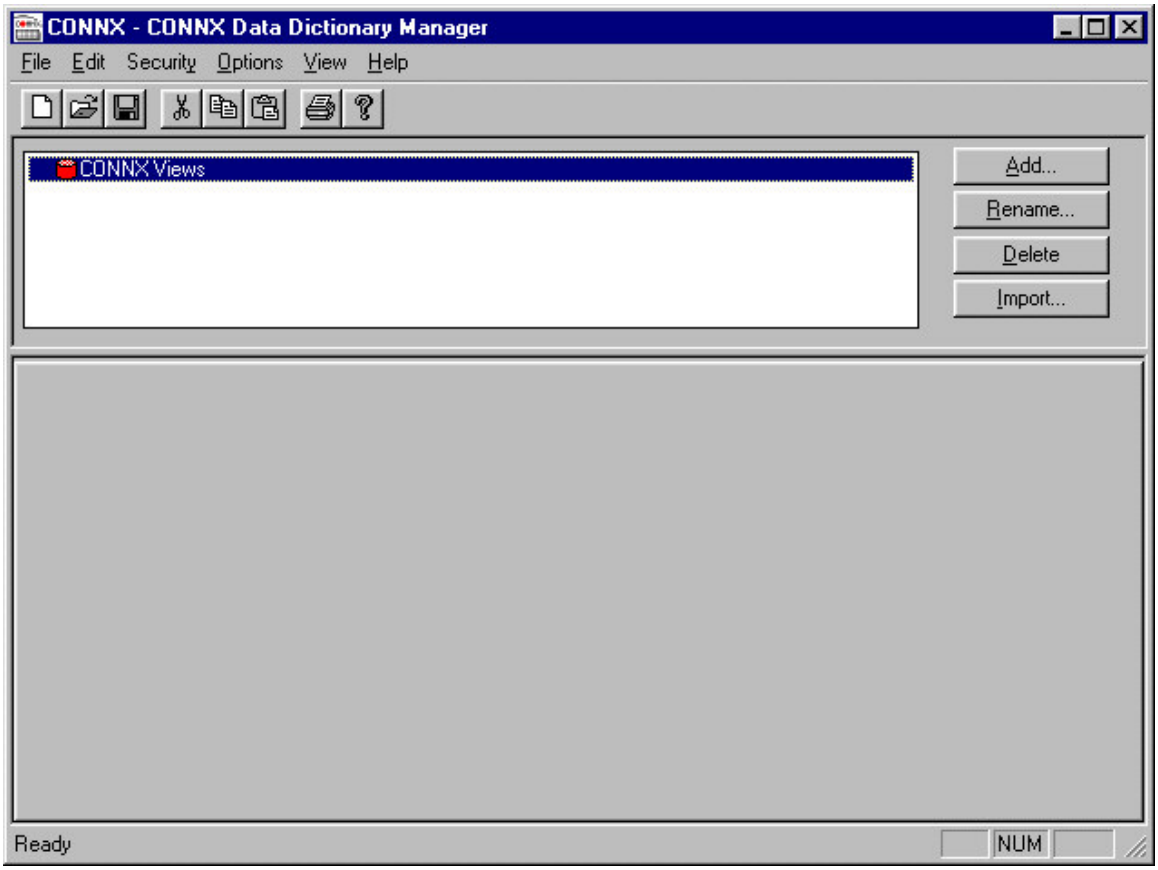
- On the **File** menu, click **Save**, or click the **Save** button on the CONNX toolbar.

To use the CONNX Find feature

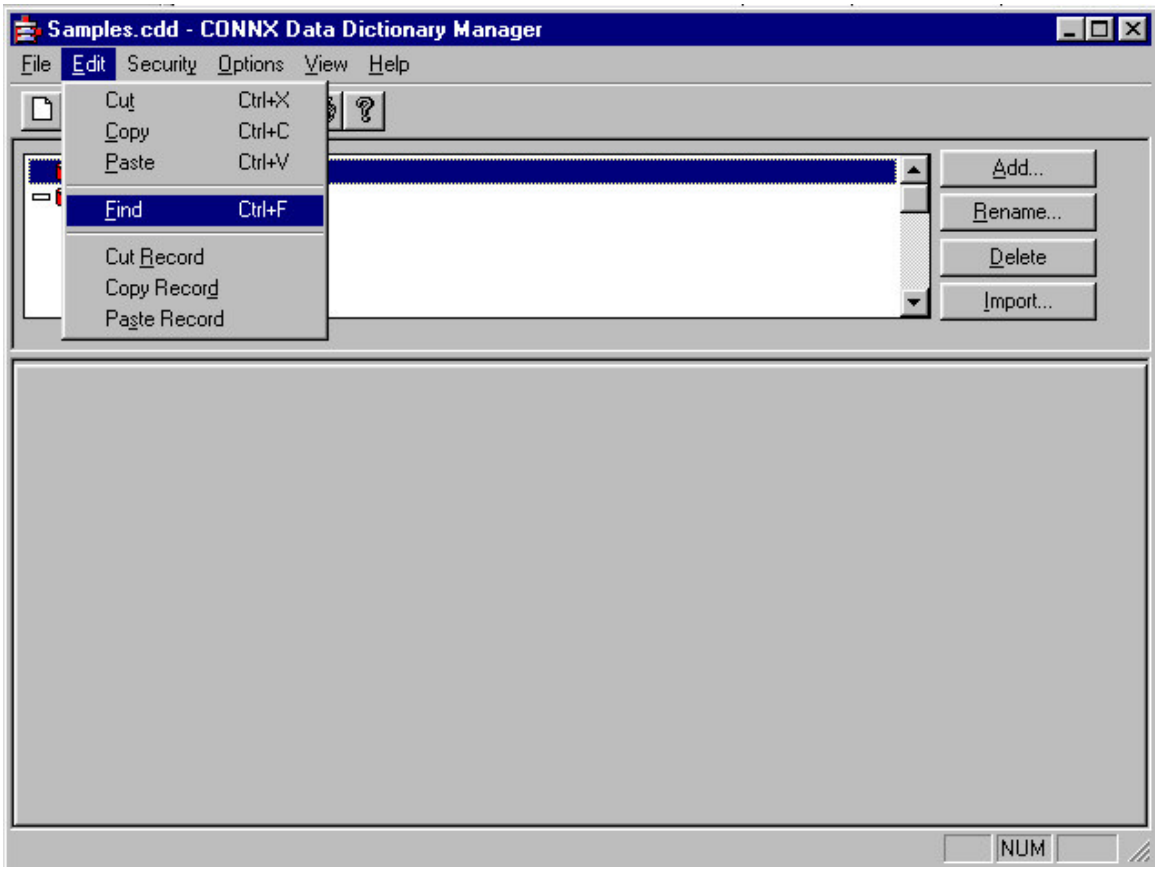
1. Click the **Start** button, and then point to **All Programs.** Point to **CONNX Driver** and then click **CONNX Data Dictionary.**
2. The **Open** dialog box appears. Select or browse to locate a CDD and then click the **Open** button.



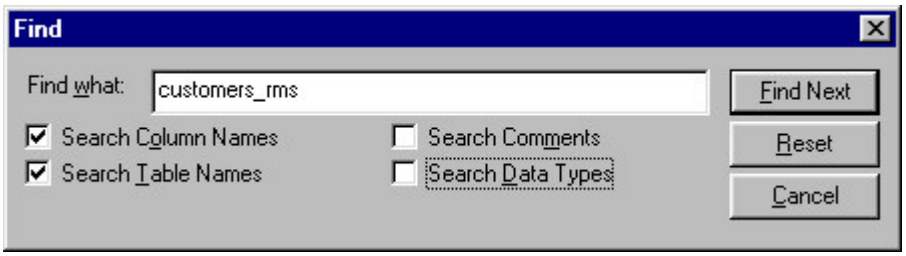
3. The CONNX Data Dictionary Manager window appears.



4. On the **Edit** menu, click **Find.**



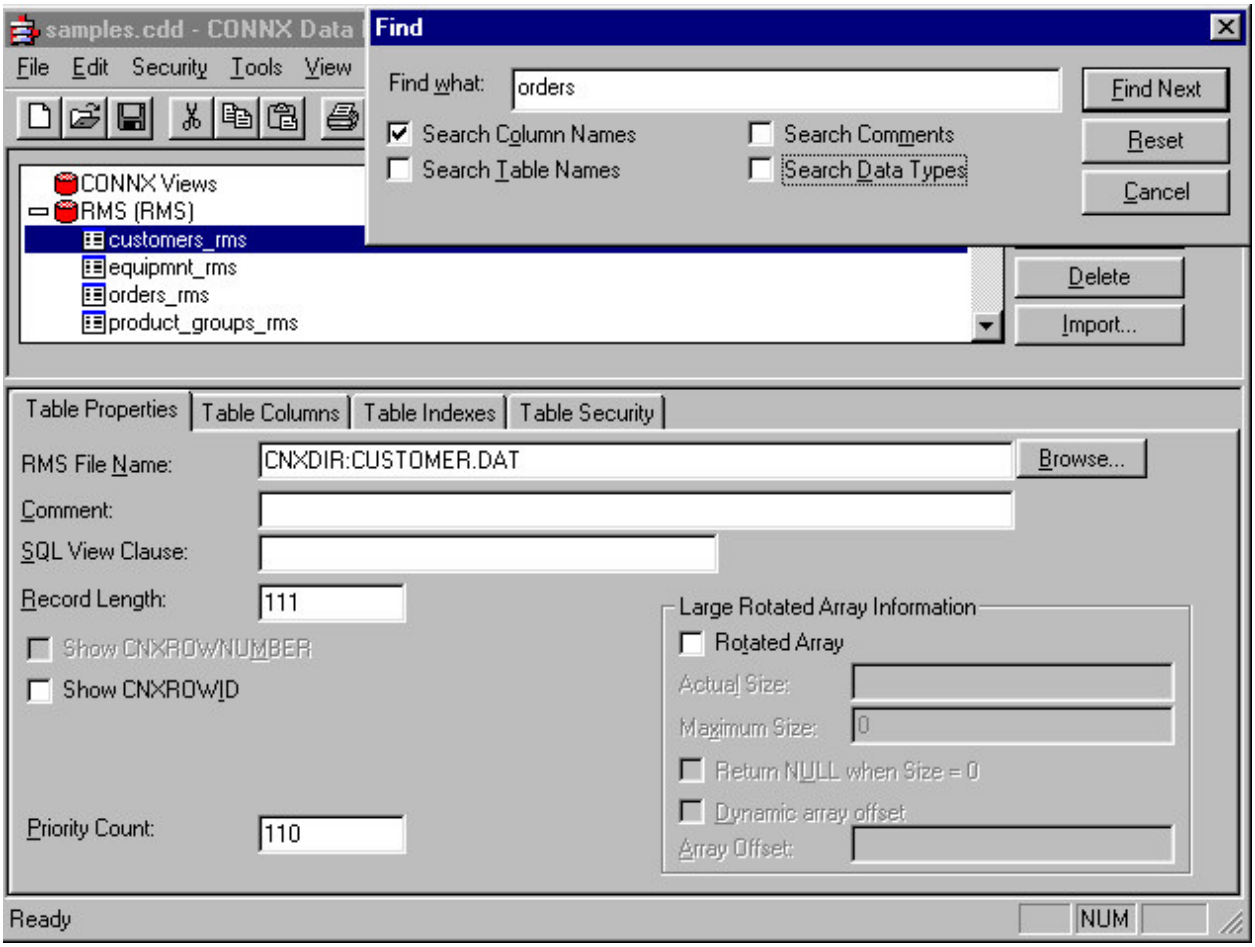
5. The **Find** dialog box appears.



6. Enter the name of the item for which you are searching, and then select a search option.

- **Search Column Names:** Select this check box when you are searching for specific column names in all databases included in the CDD.
- **Search Table Names:** Select this check box when you are searching for specific table names among the databases included in the CDD.
- **Search Comments:** Select this check box when you are searching for specific comments, for example, an entry date.
- **Search Data Types:** Select this check box when you are searching for a specific data type, for example, Binary or Quadword Decimal.

7. Click the **Find Next** button. The search results are displayed in the lower pane of the CONNX Data Dictionary window.

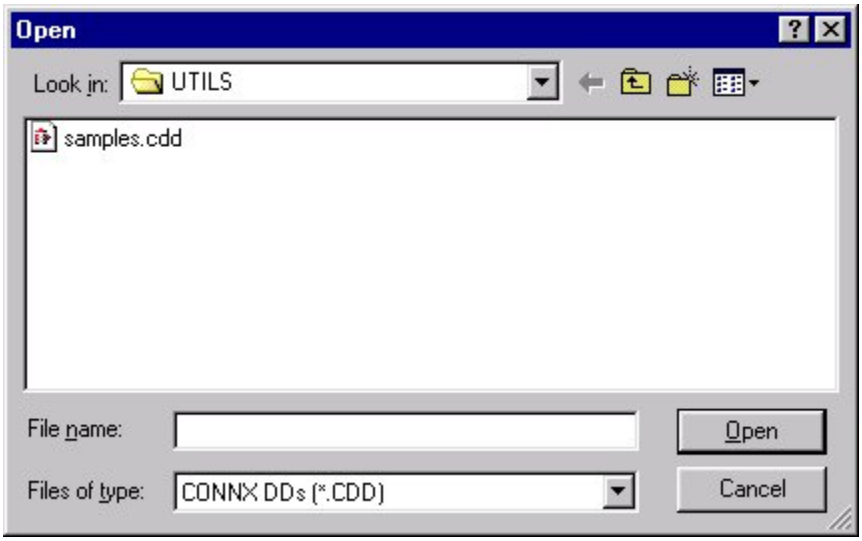


8. Click the **Reset** button to cancel the current search. Enter new search criteria in the **Find What** text box.

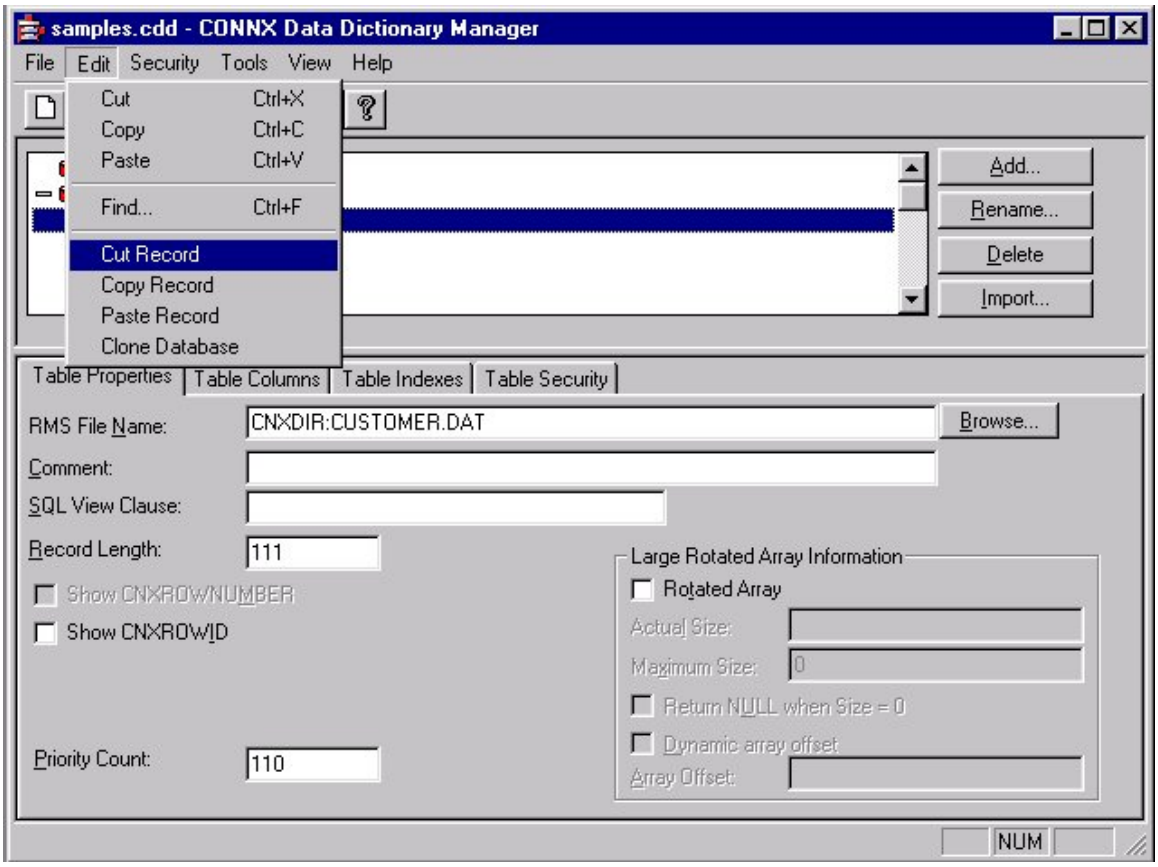
9. Click the **Cancel** button to return to the CONNX Data Dictionary Manager window.

To cut a record in the CDD

1. Click the **Start** button, and then point to **All Programs.** Point to **CONNX Driver** and then click **CONNX Data Dictionary.**
2. The **Open** dialog box appears.



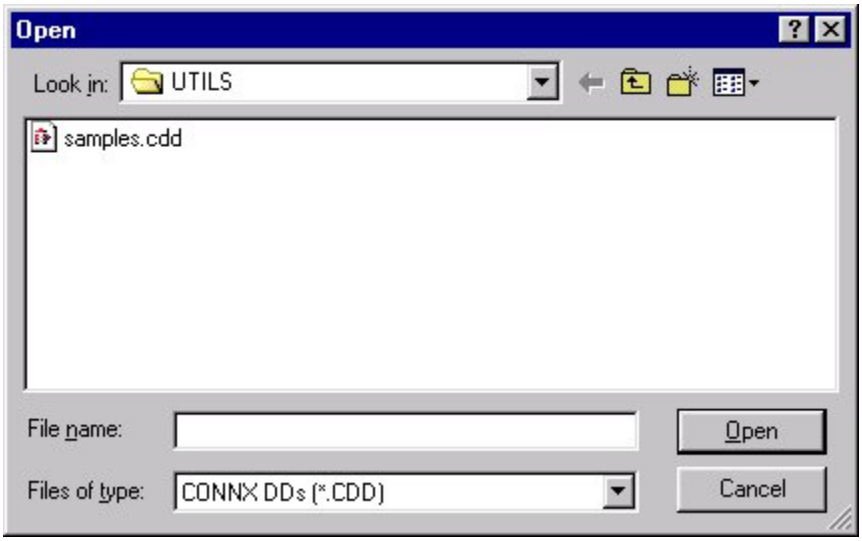
3. Select or browse to locate a CDD, and then click the **Open** button.
4. The CONNX Data Dictionary Manager window appears.
5. On the **Edit** menu, click **Cut Record.**



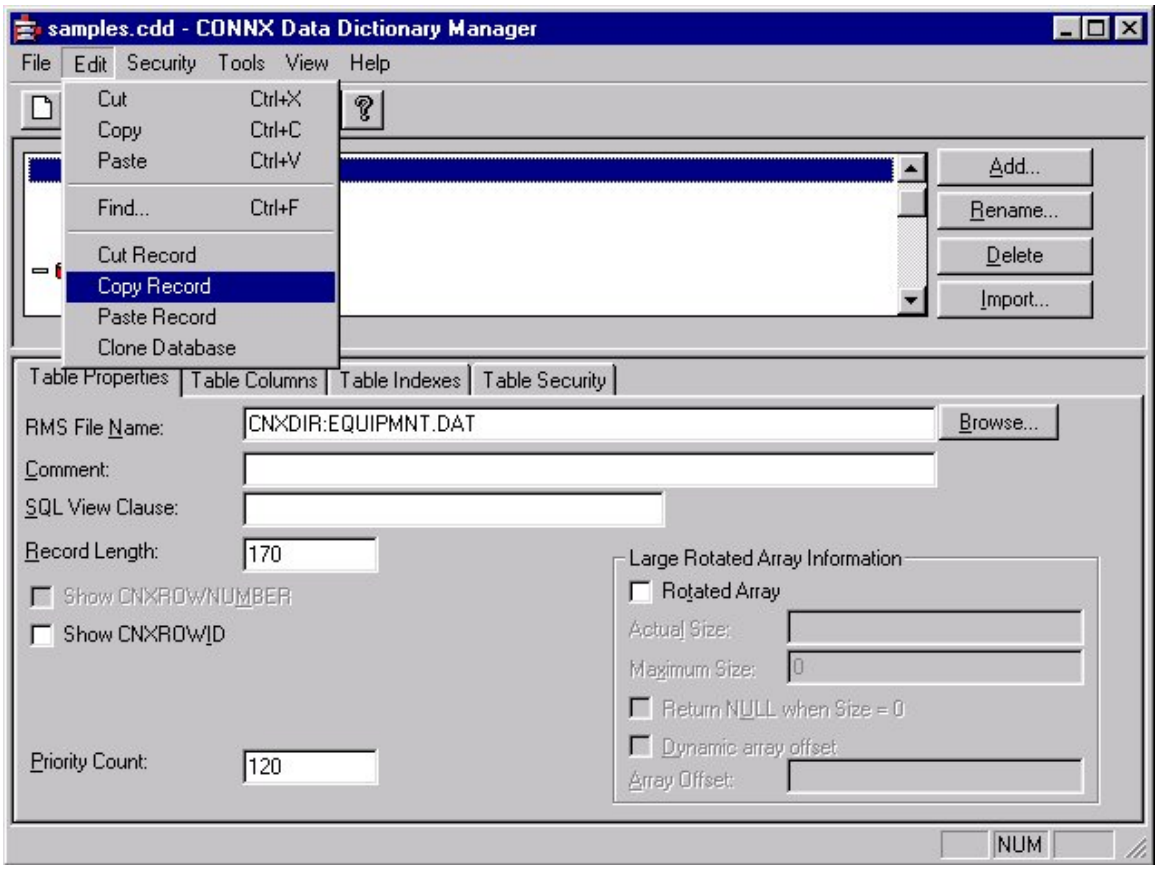
6. The record is removed from the list of records. See To paste a record in a CDD.

To copy a record in the CDD

1. Click the **Start** button, and then point to **All Programs.** Point to **CONNX Driver** and then click **CONNX Data Dictionary.**
2. The **Open** dialog box appears.



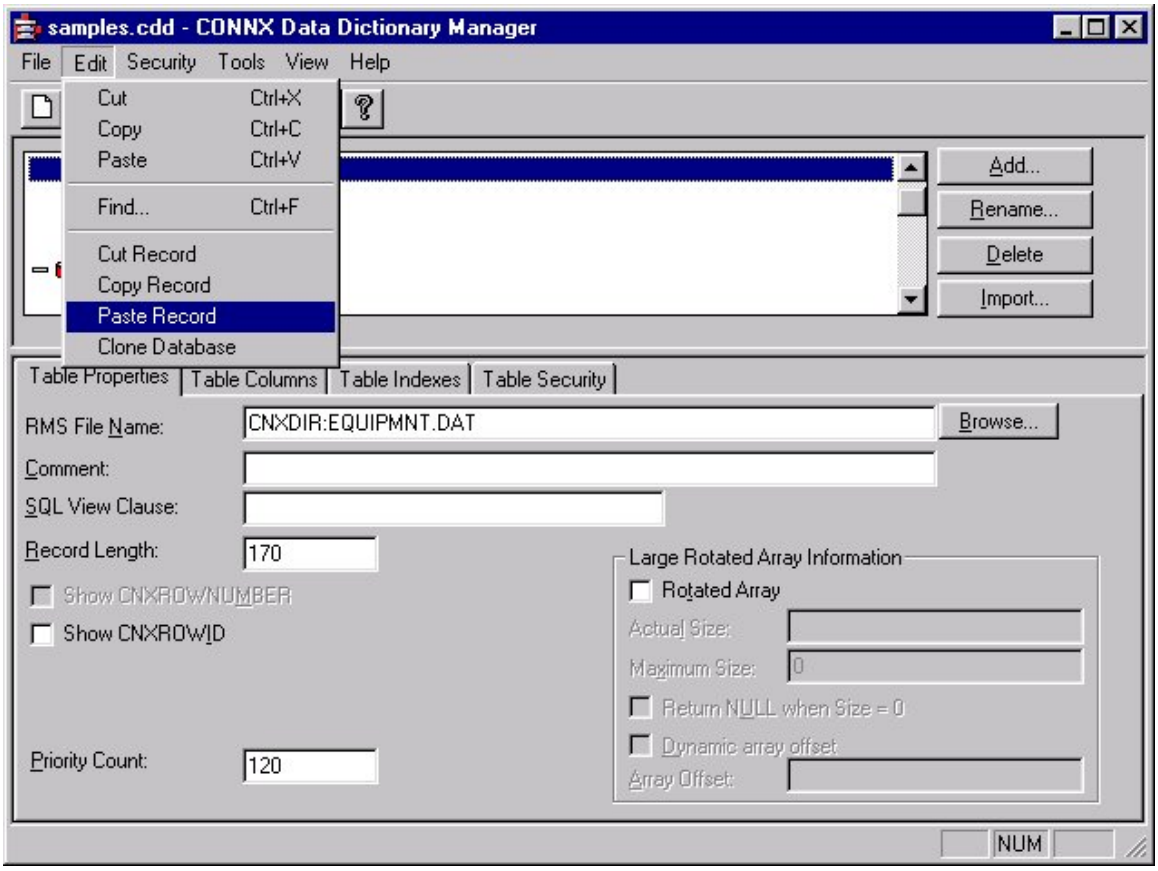
3. Select or browse to locate a CDD, and then click the **Open** button.
4. The CONNX Data Dictionary Manager window appears.
5. On the **Edit** menu, click **Copy Record.**



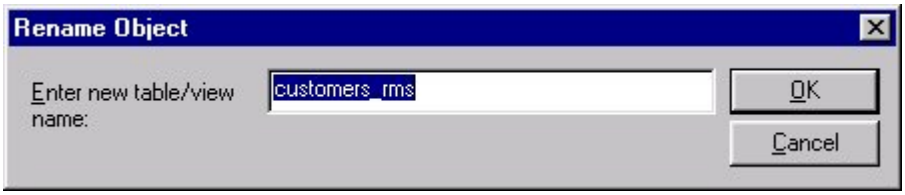
6. The record is copied from the list of records but does not appear. See To paste a record in the CDD.

To paste a record in the CDD

1. Select a location within the database for the cut or copied record.
2. On the **Edit** menu, select **Paste Record.**



3. The **Rename Object** dialog box appears.

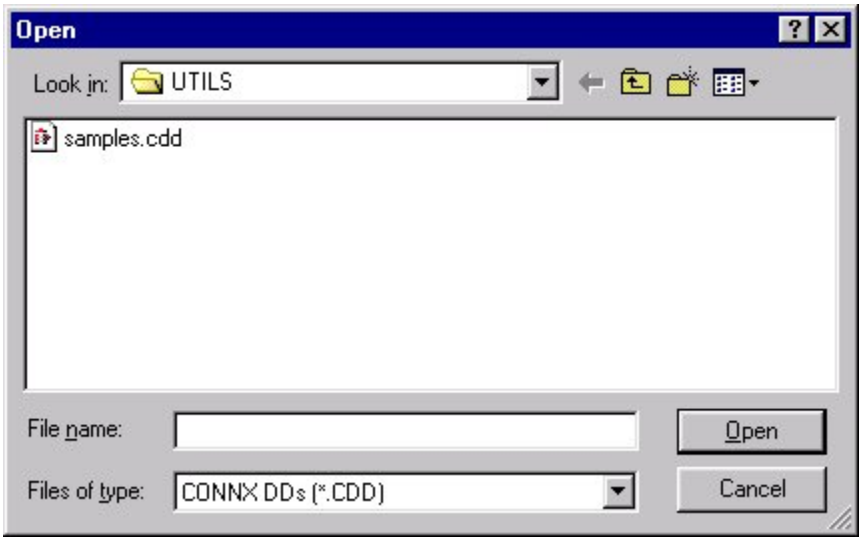


4. Type the new name of the object, and then click the **OK** button.
5. The record appears in the new location with the new name.

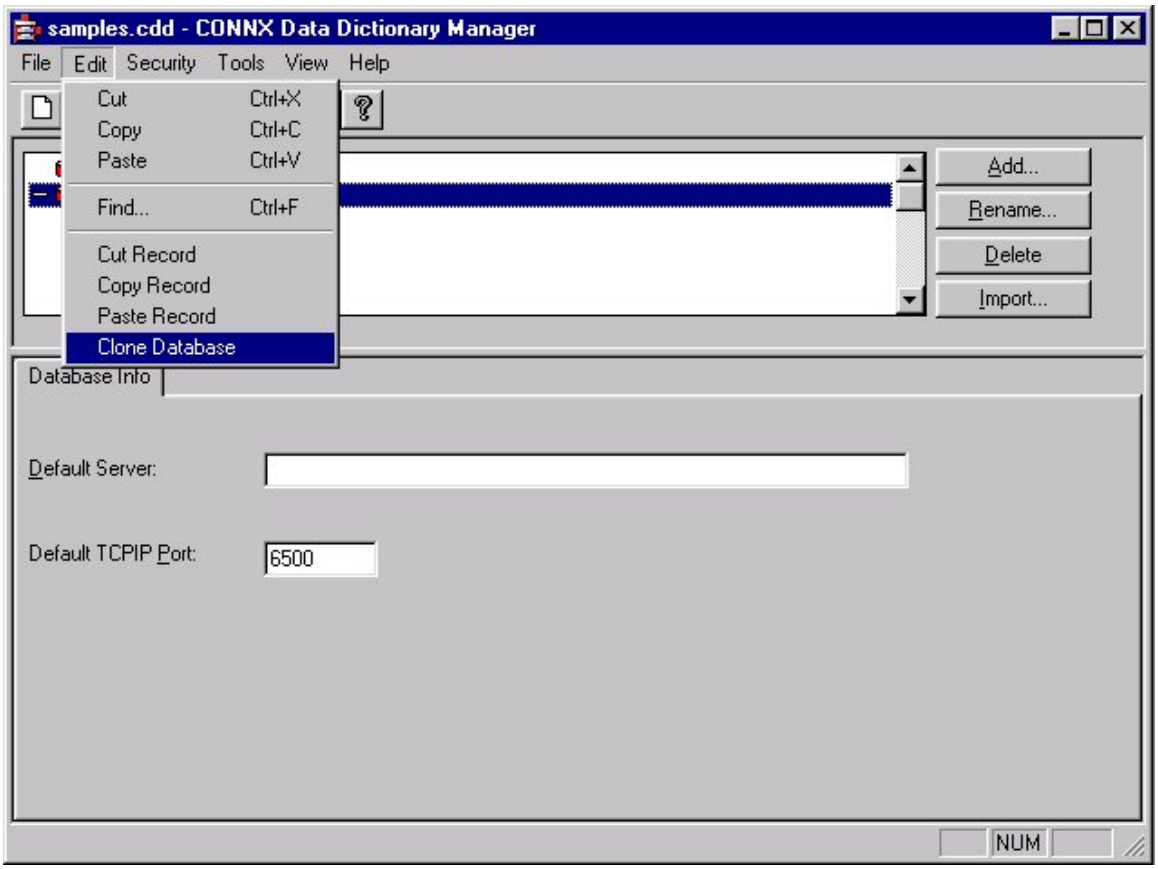
To clone a database in the CDD

Databases can be cloned and then renamed within the CONNX Data Dictionary Manager window.

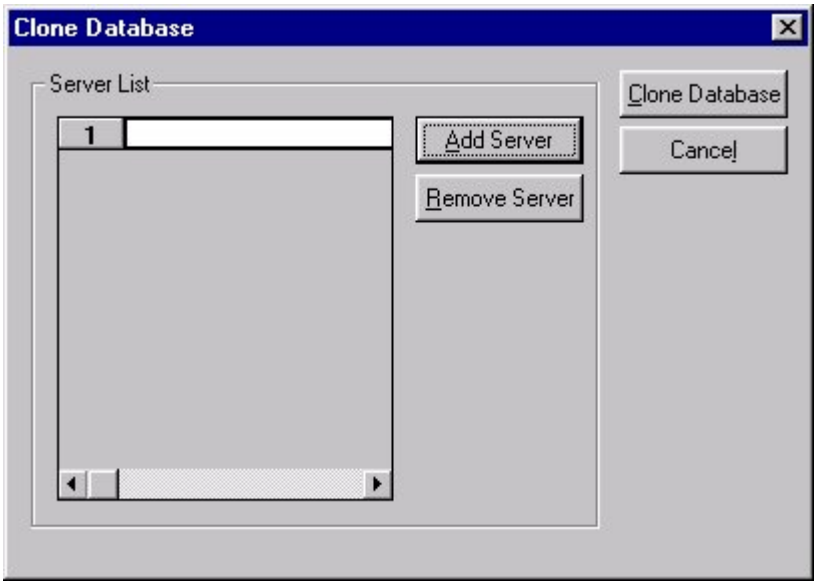
1. Click the **Start** button, and then point to **All Programs.** Point to **CONNX Driver** and then click **CONNX Data Dictionary.**
2. The **Open** dialog box appears.



3. Select or browse to locate a CDD, and then click the **Open** button.
4. On the **Edit** menu, select **Clone Database.**



5. The **Clone Database** dialog box appears.

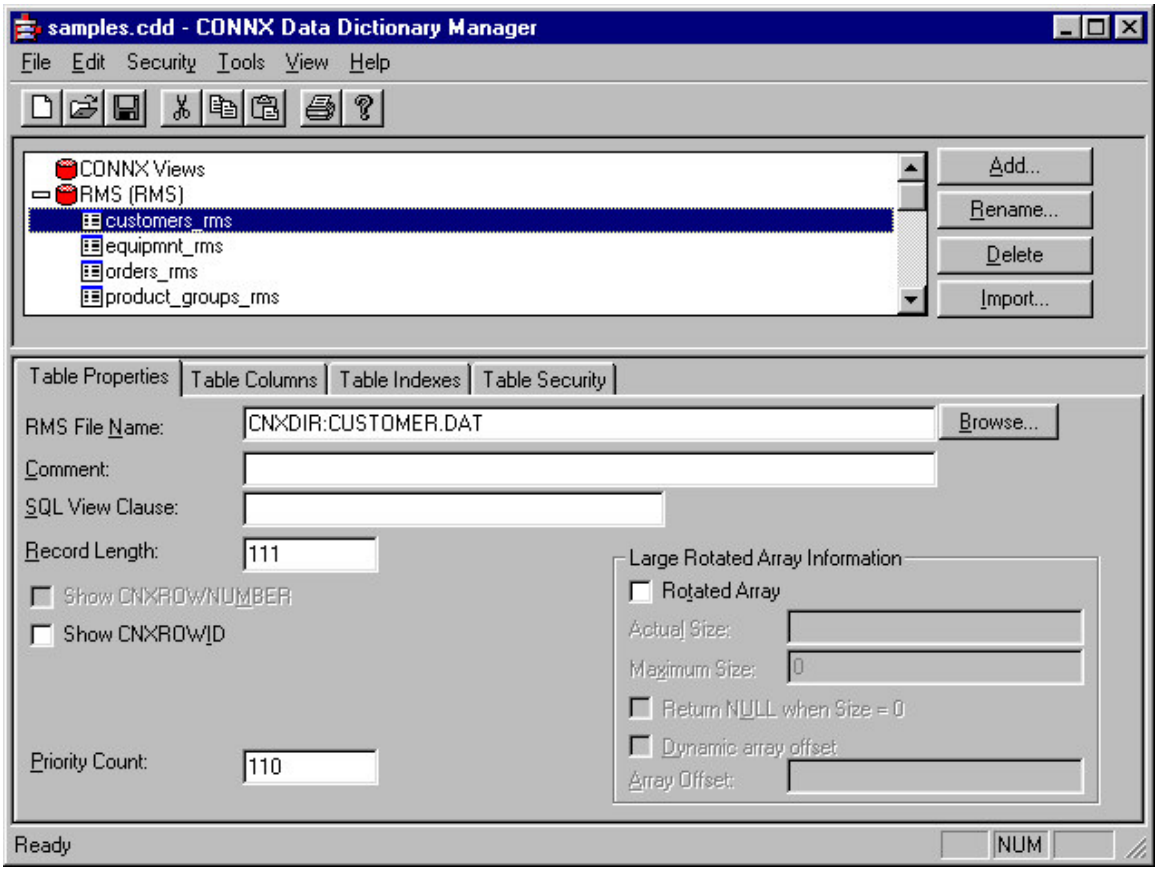


6. Select **Add Database,** and then type the name of the clone in the **Server List.** Click the **Clone Database** button.
7. The cloned database appears in the CONNX Data Dictionary Manager window.

To use the CONNX Browse button (RMS only)

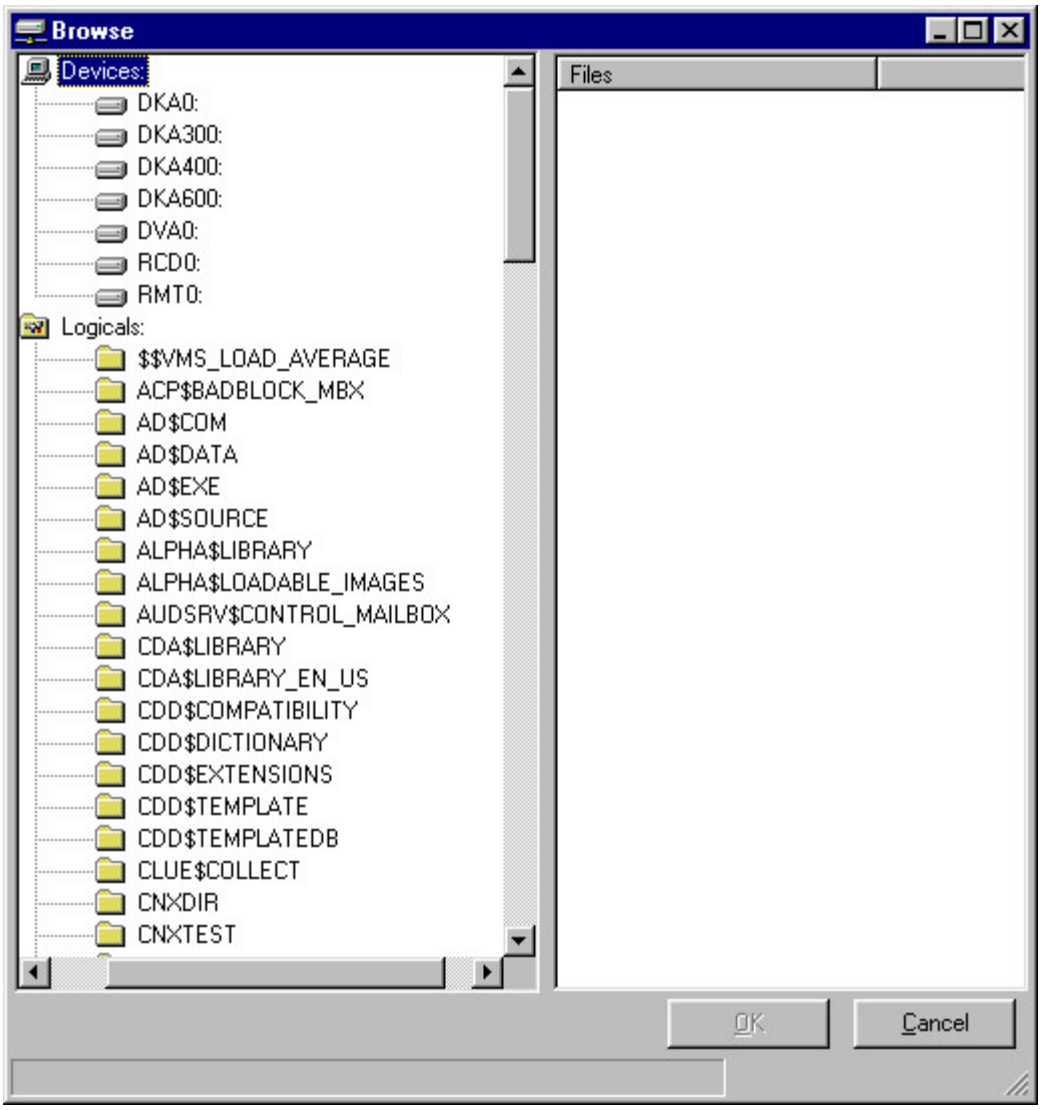
The CONNX Browse button opens a Browse dialog which enables users to traverse VMS devices and logicals to locate an RMS file for the Table Properties tab in the CONNX Data Dictionary Manager window.

1. In the CONNX Data Dictionary Manager window, select a table from an RMS database.
2. On the **Table Properties** tab, click the **Browse** button.

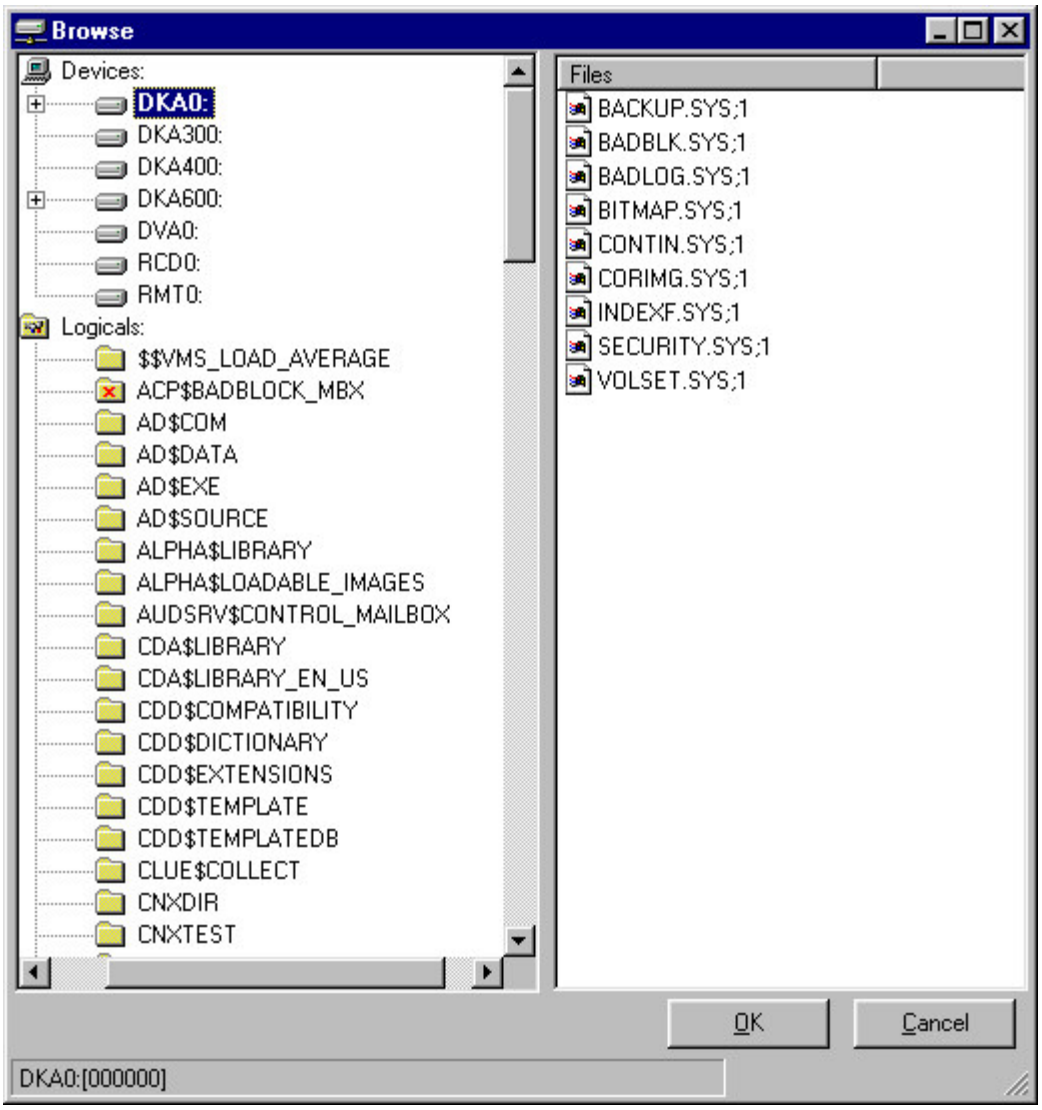


3. If you are connected to the VMS server, the **Browse** dialog box appears. Proceed to Step 6.

   If you are not connected to a VMS server, a **CONNX Database Logon** dialog box appears.
4. Type the **server name** or **IP address, user name, and password** in the corresponding text boxes.
5. Port 6500 is listed in the TCPIP Port text box by default. Any change made to the port setting in this text box becomes a permanent change to the port setting of the imported database. See **Editing the OpenVMS Site-Specific Startup Command** in the CONNX Installation Guide for information about changing the port setting on the server.
6. Click the **OK** button to log on to the server.



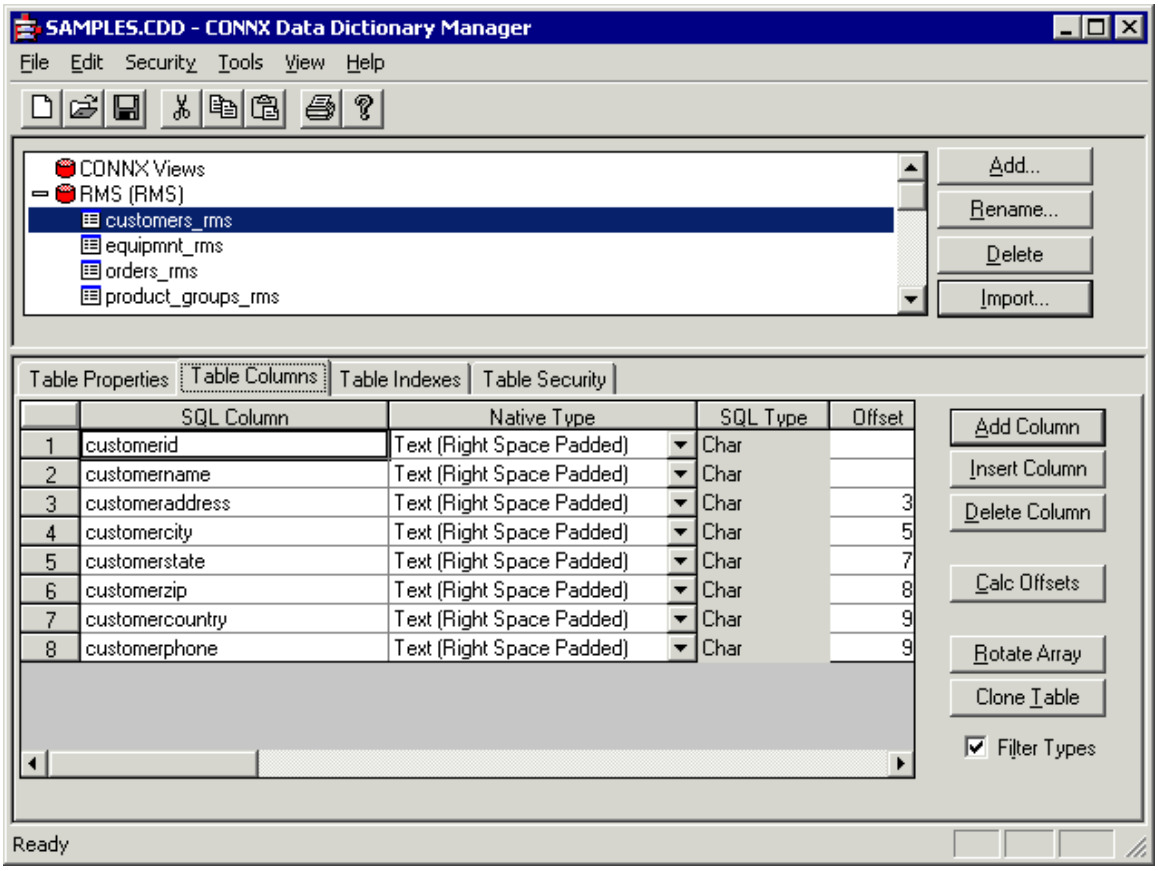
7. The **Browse** dialog box appears. The two branches of the tree in the left pane contain the devices connected to your VMS server and all of your VMS logicals.



8. Select a device or folder to access subfolders and files. All of the available files in the selected device or folder are displayed in the right pane. The current path of the VMS system is displayed in the status bar located in the lower right-hand corner of the dialog box.
9. Once you have located and selected a file, click the **OK** button to transfer the file name and path to the **RMS File Name** text box in the CONNX Data Dictionary Manager window.

Filtering CONNX Data Types

1. Import a VSAM, C-ISAM, or RMS table into the CONNX Data Dictionary Manager window.
2. Select the **Table Columns** tab.

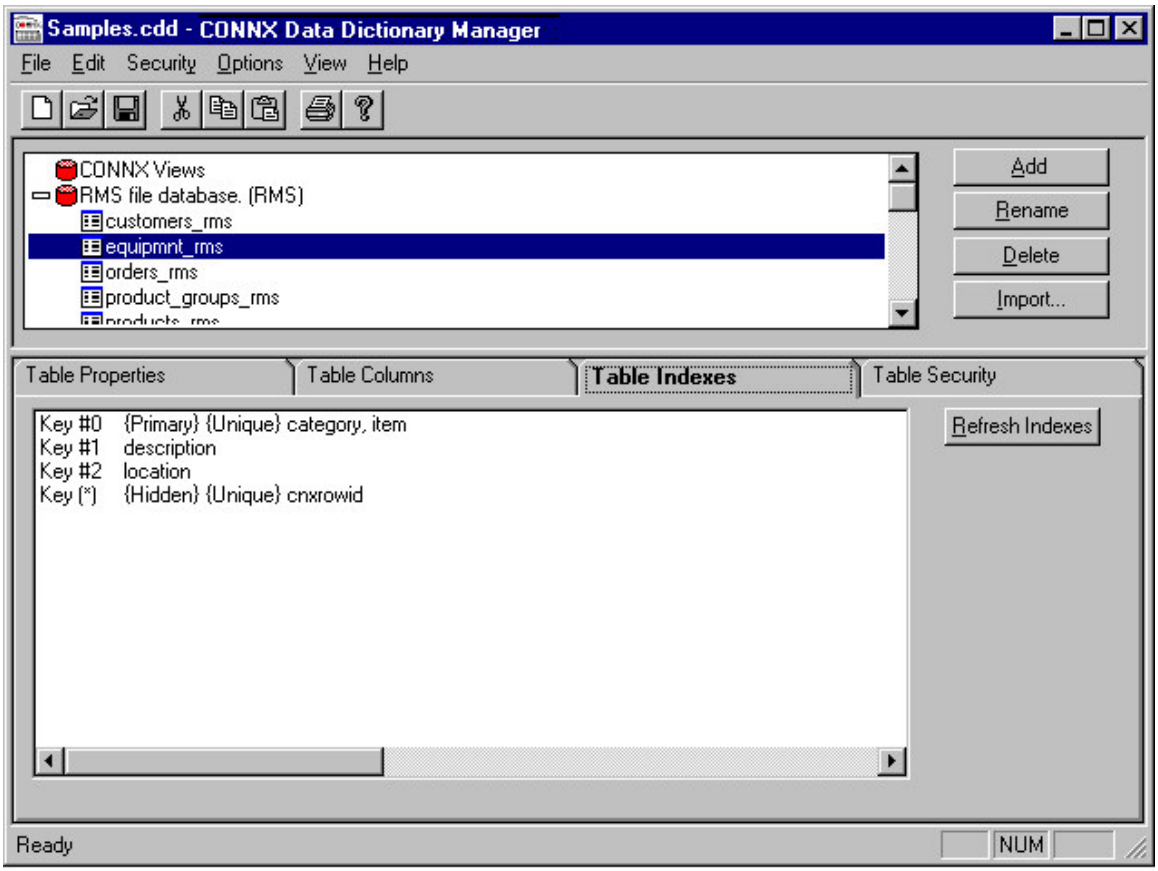


3. Select the **Filter Types** check box to filter the data types for your database.
4. The filtered data types appear in the **Native Type** list box for each type of database.

To view an index

The Table Indexes tab displays the indexes in the table. CONNX displays this list automatically. To view the indexes:

1. Click the **Table Indexes** tab in the CONNX Data Dictionary Manager window. (CONNX maintains its indexes automatically.)

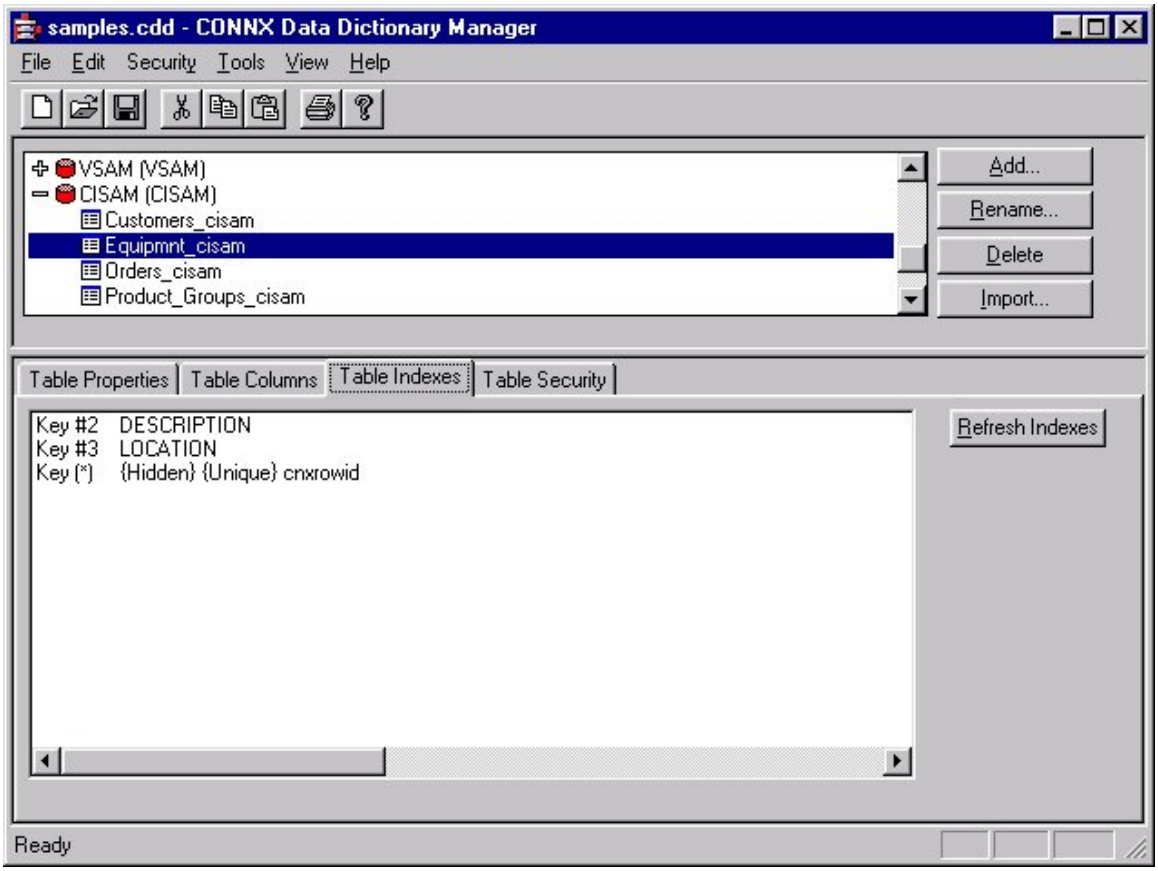


2. Click the **Refresh Indexes** button to refresh.

To refresh an index

The Table Indexes tab displays the indexes in the table. CONNX displays this list automatically. To refresh the indexes:

1. Click the **Table Indexes** tab in the CONNX Data Dictionary Manager window. (CONNX maintains its indexes automatically.) Table indexes may not initially appear.
2. Click the **Refresh Indexes** button to refresh.



Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

Importing from the command line

Command line imports must be performed from the CONNXCDD32.EXE Data Dictionary Utility install location. The default location is C:\CONNX32\UTILS.

To import COBOL copybooks from the command line, the syntax is as follows:

connxcdd32.exe <cdd path> <server name> <server userID> <server pwd> [<ssl | nossl>] import <import type> <import specification (copybook location) > <physical path of ISAM file> <SQL table name>

Example for Microfocus command line import without SSL:

connxcdd32.exe c:\cdd\importtest.cdd aixserver user password nossl import MFFD /user/fd/CUSTOMER.CPY /test1/customer.dat CUST01

Example for Dataflex command line import without SSL:

connxcdd32.exe c:\cdd\testcommandlineimport.cdd localhost user password nossl import DATAFLEX c:\connx32\utils NA

To import Adabas files from the command line, the syntax is as follows:

connxcdd32.exe <cdd path> <server name> <server userID> <server pwd> [<ssl | nossl>] import <import type> <database number> <input file>

Example for Adabas SYSOBJC command line import using an SSL connection:

connxcdd32.exe c:\cdd\importtest.cdd localhost myuser mypassword ssl import ADAOBJ 1 e:\workarea\EMPLOYEES_VEHICLES_FNR011_12.systrans

Note: if neither ssl nor nossl is specified, the import will default to nossl.

The list of valid import types:

RMS text specification file	RMSTXT
RMS VAX or Alpha CDD	RMSCDD
RMS Powerhouse PDL files	RMPDL
RMS COBOL FD files	RMSFD
RMS SCT COBOL FD files	RMSSCT
RMS SCT Datamart COBOL FD files	RMSSCTDM
RMS DIBOL files	RMSDIBOL
RMS Formatted DDL	RMSDDL
RMS VAX Basic file	RMSBASIC
PowerFLEX files from directory	DATAFLEX
RDB database RDB	DATAFLEX
DBMS Codasyl database	DBMS

DB2 database	DB2
Oracle database	ORACLE
Desktop OLEDB/ODBC Adapter	DESKTOP
Enterprise OLEDB/ODBC Adapter	ENTERPRISE
SQL Server	SQLSERVER
Sybase	SYBASE
Informix	INFORMIX
VSAM text specification file	VSAMTXT
VSAM COBOL FD files	VSAMFD
VSAM SCT COBOL FD files	VSAMSCT
VSAM SCT Datamart COBOL FD files	VSAMSCTDM
IMS database	IMS
IMS Index text specification file	IMSTXT
CISAM text specification file	CISAMTXT
CISAM COBOL FD files	CISAMFD
CISAM Powerhouse PDL files	CISAMPDL
CONNXStore	CONNXSTORE
ADABAS FDT Import	ADAFDT
ADABAS Import - SYSOBJH Transfer Format	ADAOBJ
ADABAS Dynamic DDL Import	ADADDL
DISAM text specification file	DISAMTXT
DISAM COBOL FD files	DISAMFD
MicroFocus text specification file	MFTXT
MicroFocus COBOL FD files	MFFD
SQLite database	SQLITE
RM/Cobol text	RMTXT

specification file	
RM/Cobol FD files	RMFD
JMS Import	JMS
Change Data Capture Target	CDC
ReDIS	REDIS
MySQL	MYSQL

CONNX Catalog Support

Catalog Support Defined

CONNX now fully supports three-part table names: catalogs, schemas, and objects.

The following mappings are automatically returned:

- Catalog name = logical database name
- Schema name = CONNX owner name (currently "dbo" for database, CONNXDB for views)
- Object name = table name (or view name)

Objects can be referenced using the fully qualified ANSI SQL catalog.schema.object syntax. You can also use a single object name, if there is no ambiguity.

Example:

```
SELECT * FROM rms.dbo.customer
```

Use of the schema name is optional, or you may substitute two periods for the schema name.

Example:

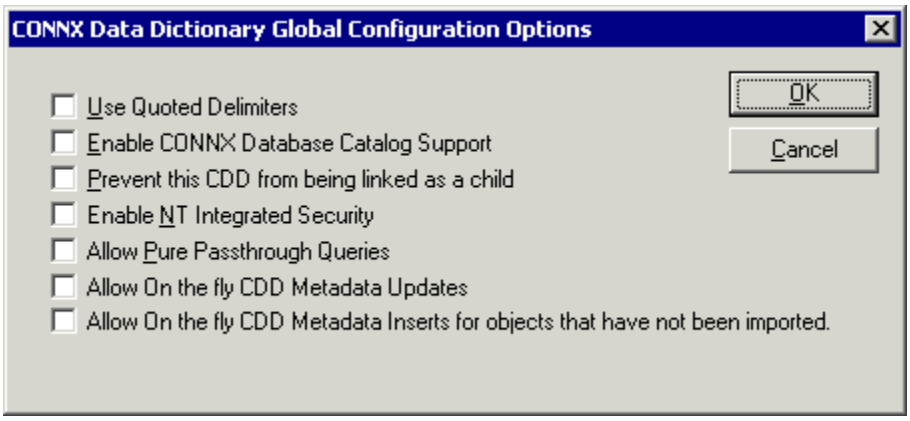
```
SELECT * FROM rms..customer
```

Important: *Microsoft Access supports two-part table names that include only the schema and object name. See More Access Tips for more information regarding the use of two-part table names in Microsoft Access.*

Important: *For an existing CDD, you must select* ***Enable CONNX Database Catalog Support*** *on the* ***Tools/Options*** *menu to take advantage of catalog support. For new CDDs, the feature is enabled by default.*

To enable CONNX database catalog support

1. Select a CDD in the CONNX Data Dictionary Manager window.
2. On the **Tools** menu, select **Options.**
3. The CONNX Data Dictionary Global Configuration Options dialog box appears.



4. Select **Enable CONNX Database Catalog Support.**
5. Catalog support in the selected CDD is disabled. Click the **OK** button to return to the CONNX Data Dictionary Manager window.

Important: *You must disable full catalog support if you intend to use single table names.*

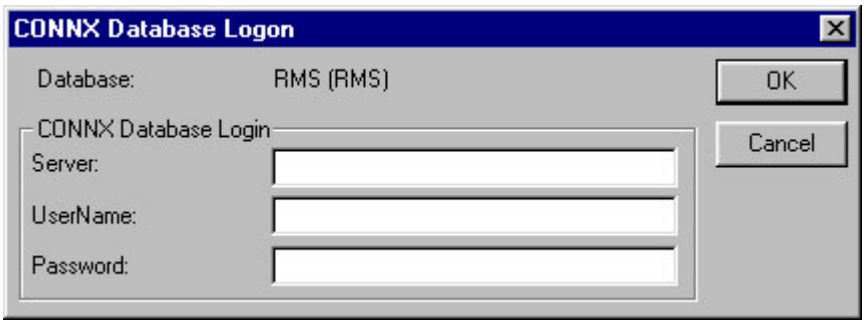
Updating Statistics in the CONNX Data Dictionary

To update statistics in the CONNX Data Dictionary Manager

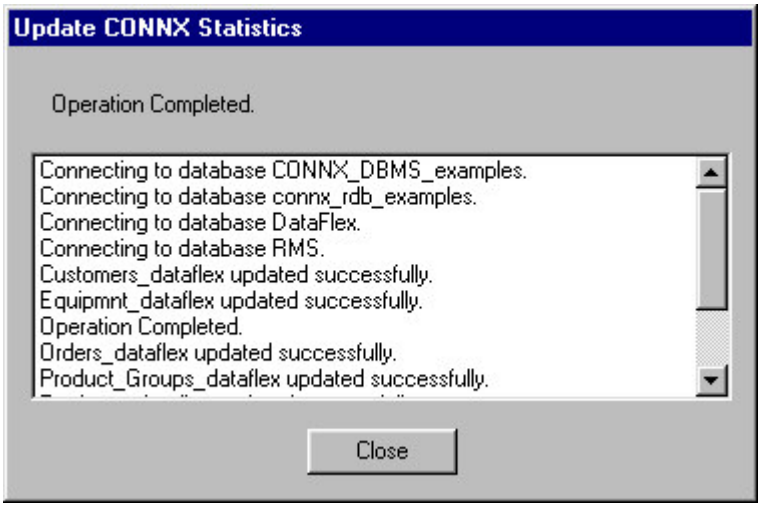
The Update Statistics function updates index and performance information about the tables that is stored in the CONNX Data Dictionary. This information is used by CONNX to form query plans and perform cross-database query optimization.

To update statistics in the CONNX Data Dictionary Manager

1. On the **Tools** menu in the CONNX Data Dictionary Manager window, click **Update Statistics.**
2. The **CONNX Database Logon** dialog box appears.



3. Log in using your CONNX user name and password for all databases to update.
4. The **Update CONNX Statistics** dialog box appears with messages stating that the databases have been successfully updated.



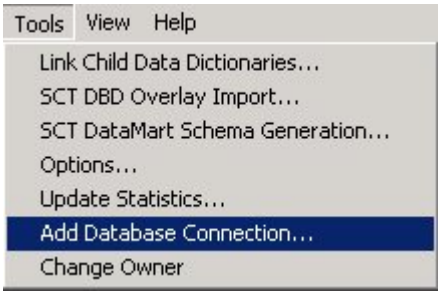
Adding a Database Connection

Adding a database connection

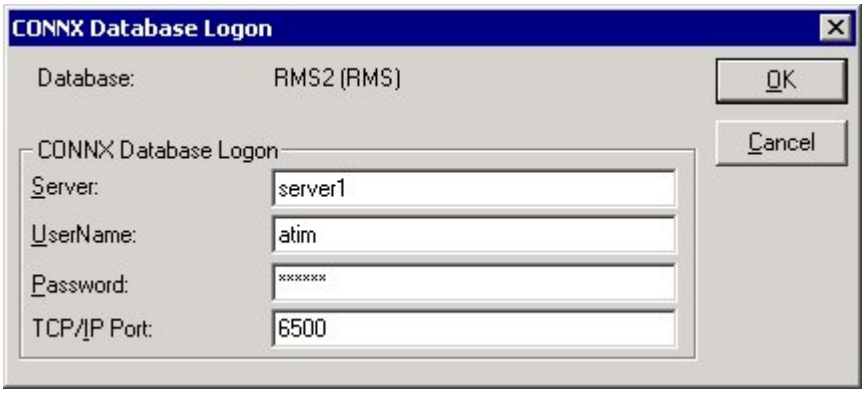
You can add a new or additional database connection to an already existing RMS, C-ISAM, and VSAM databases in the CONNX Data Dictionary Manager window. Each database listed in the CONNX Data Dictionary Manager window can be associated with a different server. This feature can be used for organizational purposes, or for importing tables from the same database in instances where the databases are located on two different servers and the CONNX Listener is listening on two different ports.

Note: *You should create the database connection before attempting to import an additional database from your server to prevent accidentally erasing files that may be duplicates of others already existing in another database container in the CONNX Data Dictionary Manager window.*

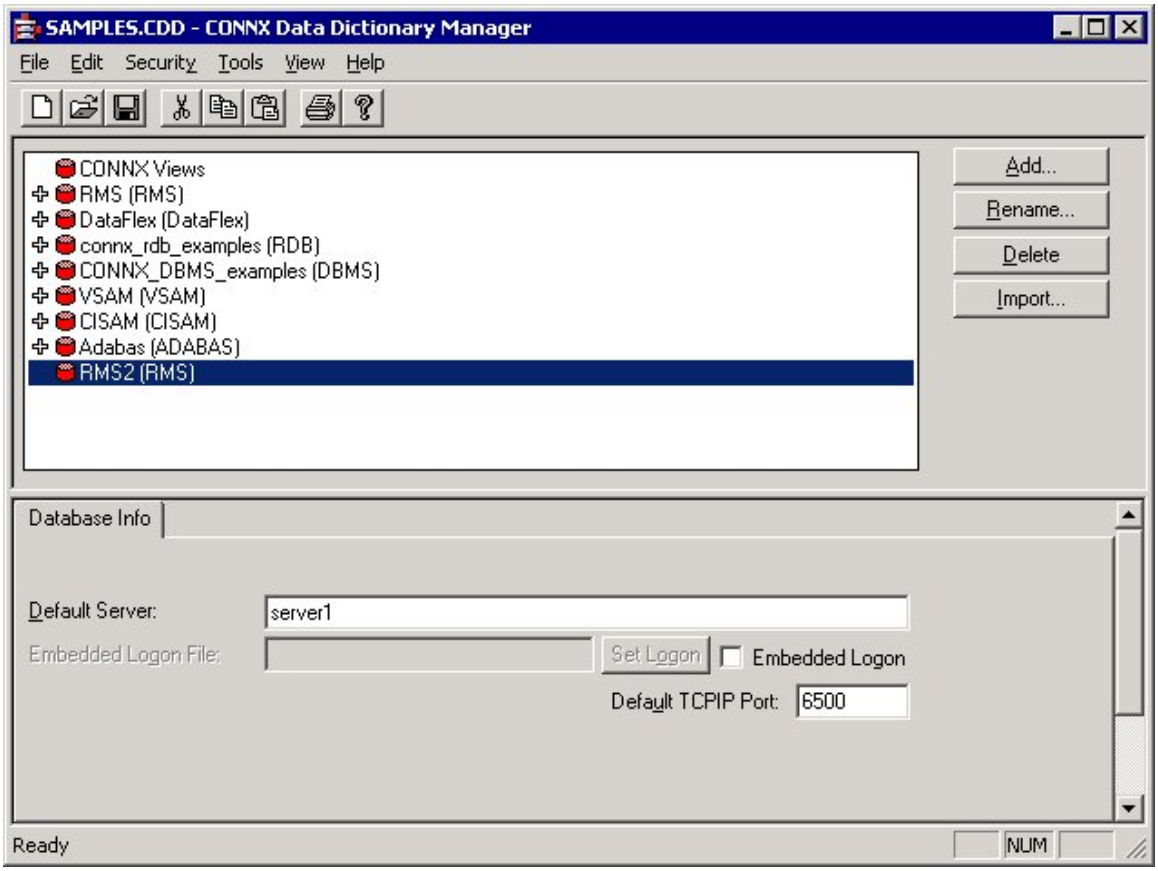
1. On the **Tools** menu in the CONNX Data Dictionary Manager window, click **Add Database Connection.**



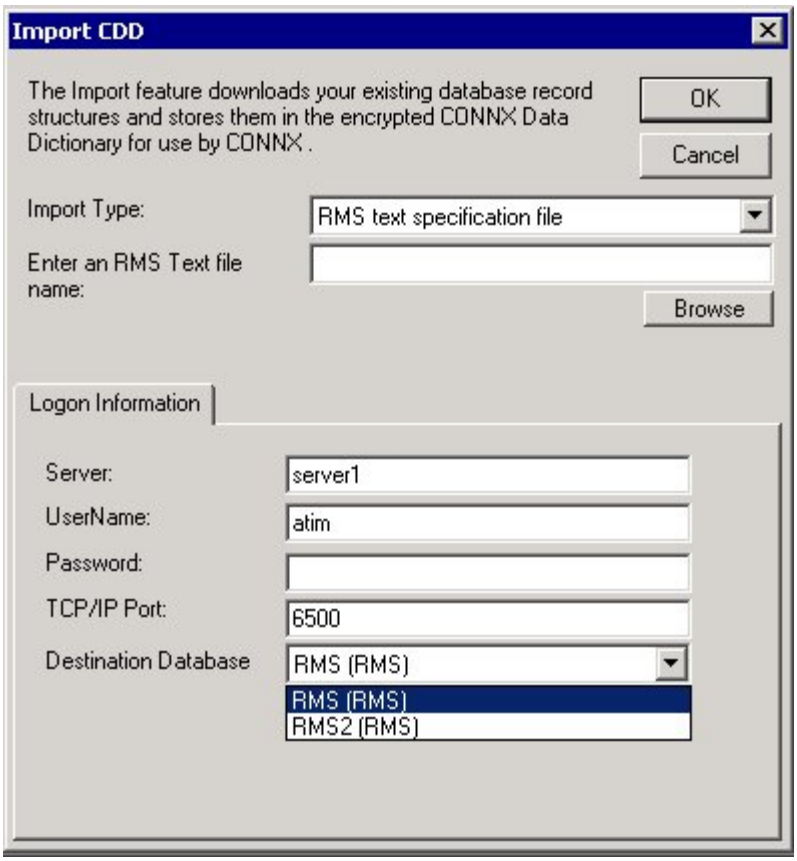
2. The **Enter the logical name of the new database:** dialog box appears. Enter the new name of the database from which you will be importing files. Select a database type (either RMS, C-ISAM, or VSAM) from the list box. Enter the server name, and then click the **OK** button.
3. The **CONNX Database Logon** dialog box appears. Log on to your system database by entering the server name, a user name and password, and the TCP/IP port. (If you are using Pathworks, the TCP/IP port text box does not appear.)



4. The new database container appears in the upper pane of the CONNX Data Dictionary Manager window.



5. To import files into the new database, click the **Import** button. The **Import CDD** dialog box appears. Enter the appropriate information for your database type. See the section on RMS imports ("CONNX for DataFlex, RMS, and DBMS Databases") for more information. Select the desired destination database from the Destination Database list box.



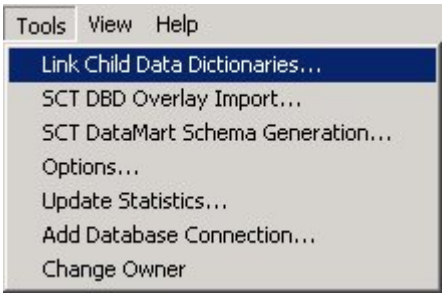
6. The file will be added to the new container in the CONNX Data Dictionary Manager window. Note that this example was created using the RMS text specification file import option. Imports using the other import options are performed in the same manner.

Linking Master CONNX CDDs

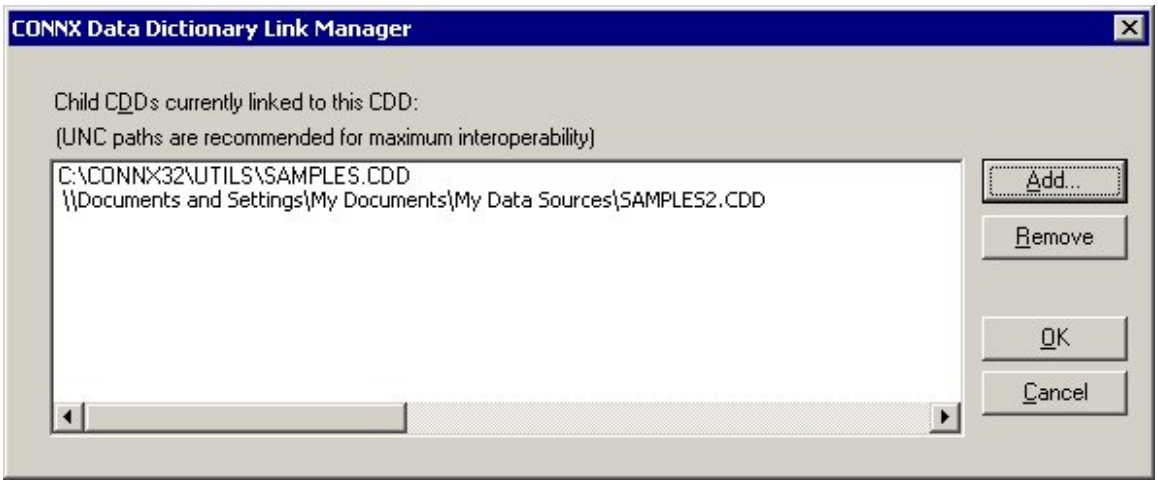
To link child dictionaries in the CONNX Data Dictionary Manager

Linking CONNX Data Dictionaries enables the aggregation of multiple CDDs into a single CDD. This type of distributed CDD can be useful in large organizations that have several distinct groups that may want to maintain ownership of a section of a larger CDD.

1. On the **Tools** menu in the CONNX Data Dictionary Manager window, click **Link Child Data Dictionaries.**



2. The **CONNX CDD Link Manager** dialog box appears.

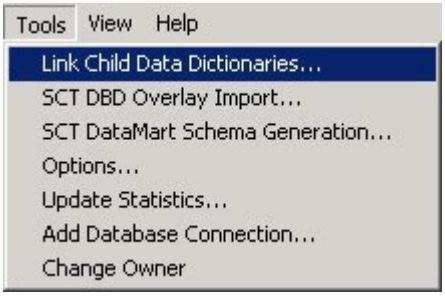


3. Click the **Add** button to locate a CDD to add as a child to any open CDD file.
4. The **Open** window appears.

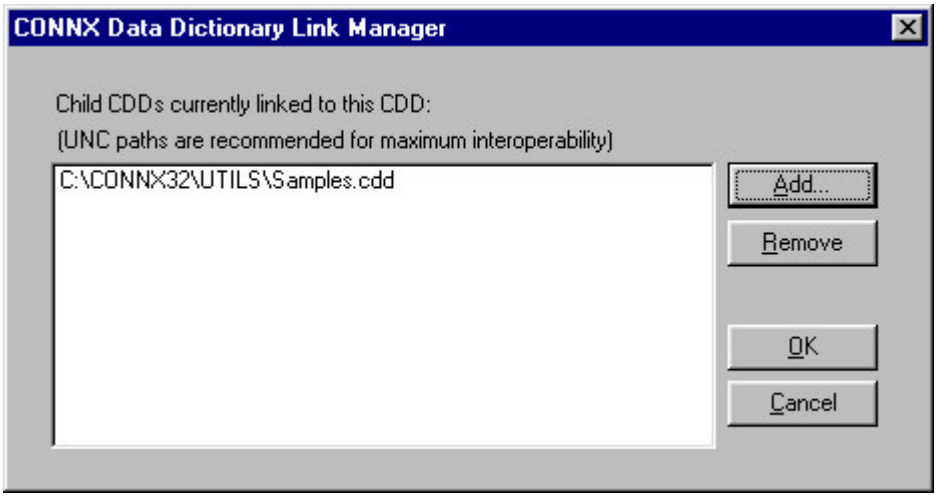
5. Select a CDD file and then click the **Open** button.
6. The CDD file is added as a child and appears in the Link Manager window.
7. Click the **OK** button to complete the link and return to the CONNX Data Dictionary Manager window.

To remove a child CDD in the CONNX Data Dictionary Manager window

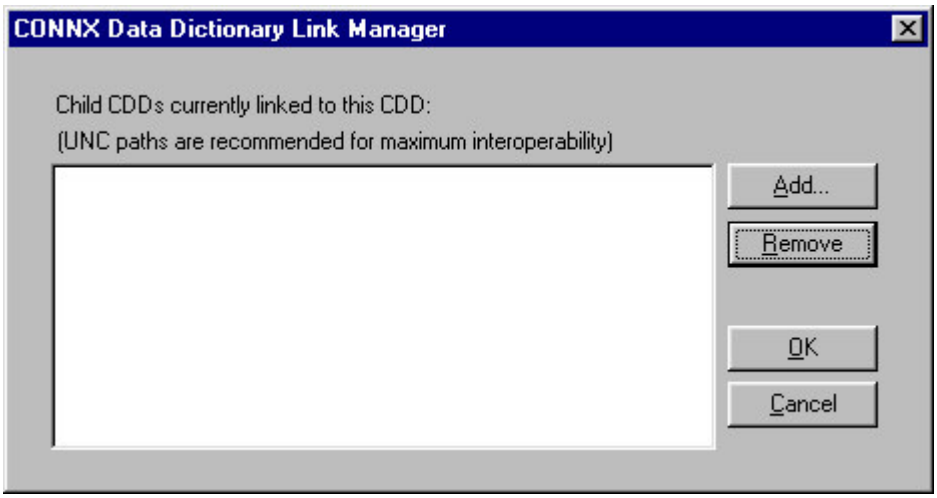
1. On the **Tools** menu in the CONNX Data Dictionary Manager window, select **Link Child Data Dictionaries.**



2. The Link Manager window appears with a list of linked child data dictionaries.



3. Select a child data dictionary to remove, and then click the **Remove** button.
4. The child data dictionary is removed.

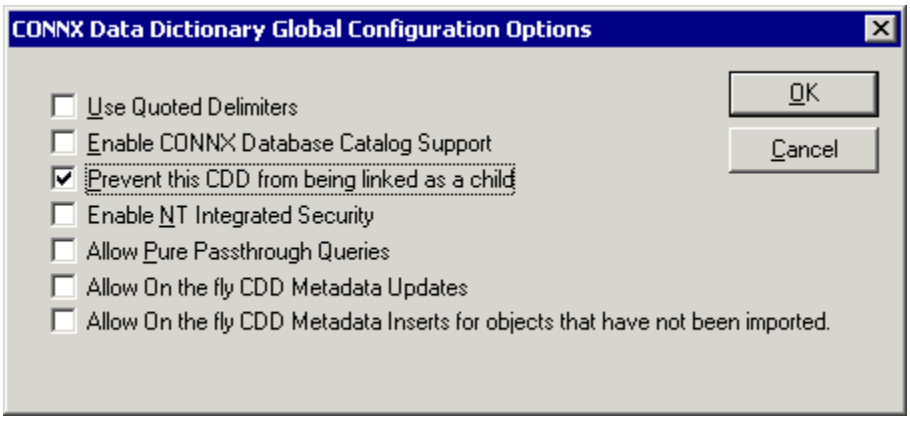


5. Click the **OK** button to return to the CONNX Data Dictionary Manager window.

To prevent a CDD from being linked as a child

You can prevent a CDD from being linked as a child as a security measure.

1. Select a CDD in the CONNX Data Dictionary Manager window.
2. On the **Tools** menu in the CONNX Data Dictionary Manager window, select **Options.**
3. The **CONNX Data Dictionary Global Configuration Options** dialog box appears.
4. Select **Prevent This CDD From Being Linked as a Child.**



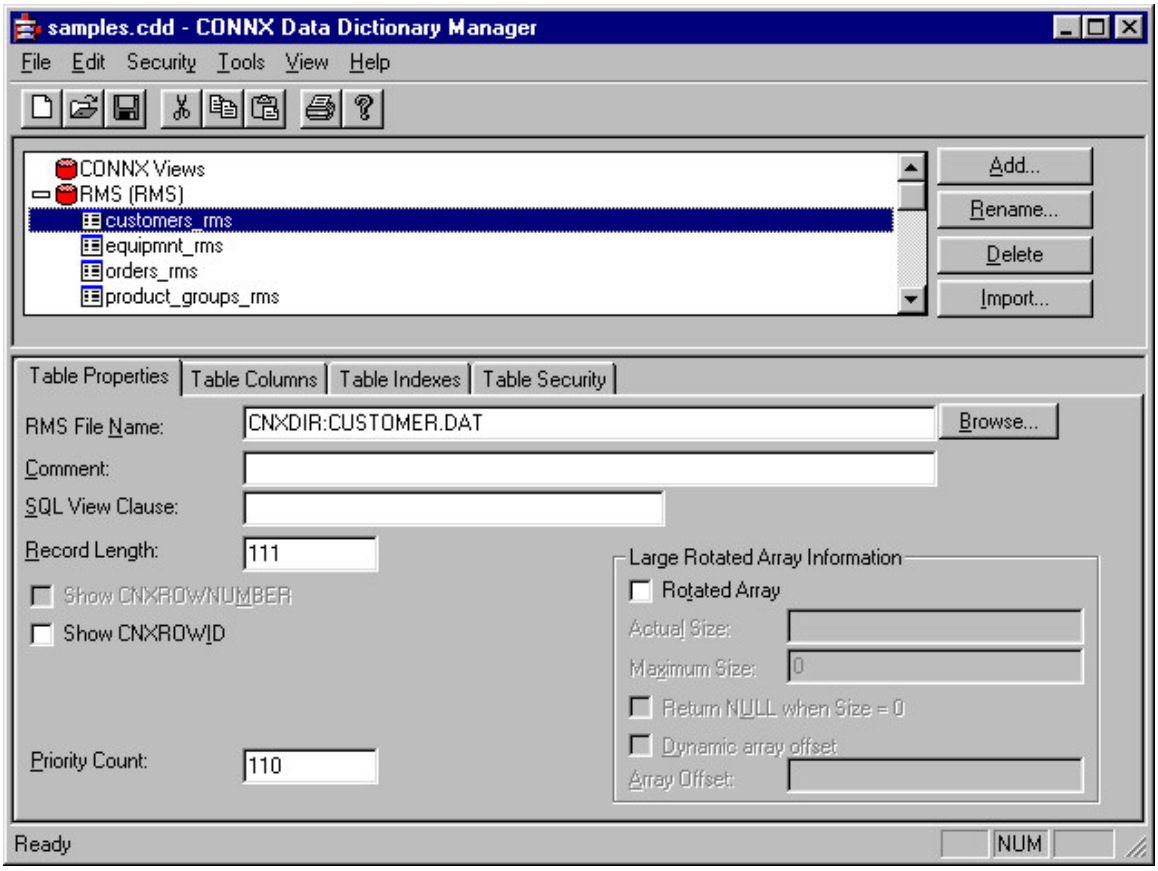
5. The selected CDD is secure. Click the **OK** button to return to the CONNX Data Dictionary Manager window.

Viewing Database Information by Database, Object, or Owner

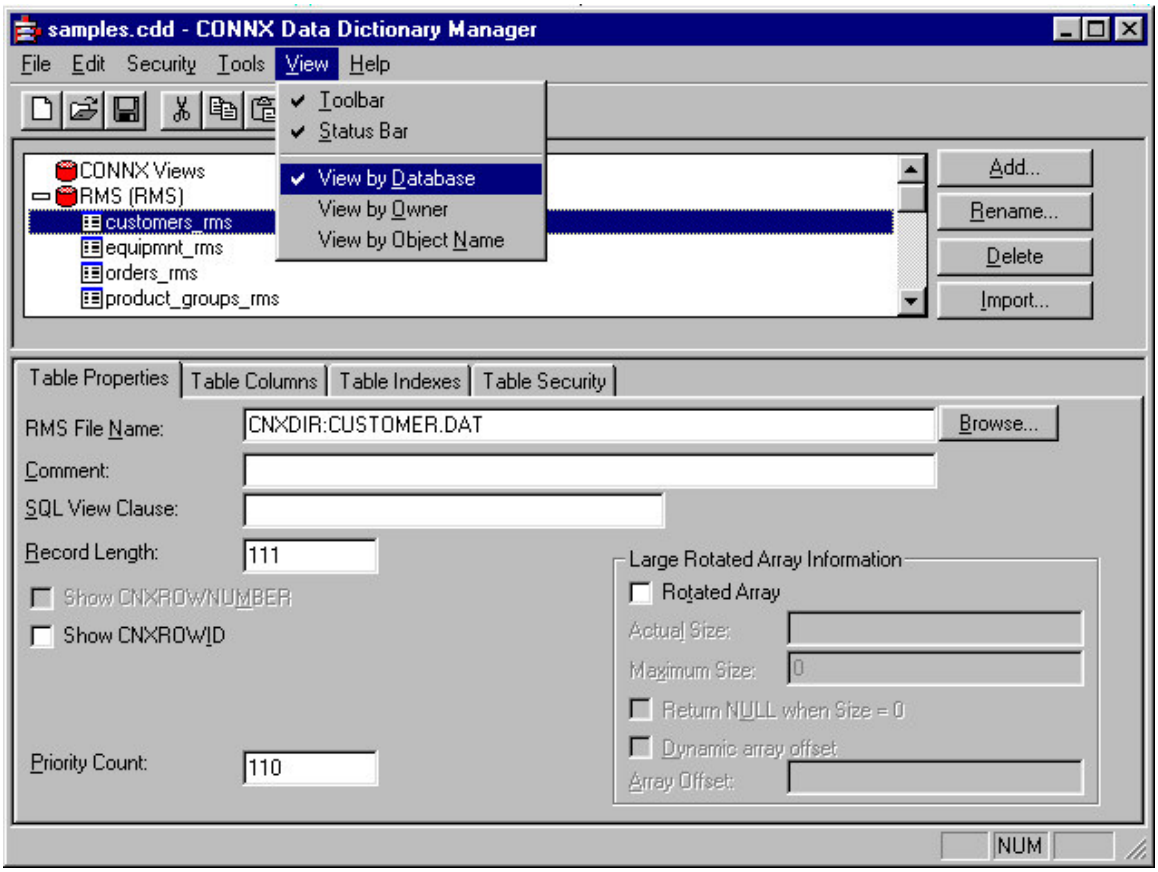
To view database information by database name

Users may view database information within the CONNX Data Dictionary by database, object, or owner name by selecting one of the choices available on the View menu.

1. Select a CDD in the CONNX Data Dictionary window.



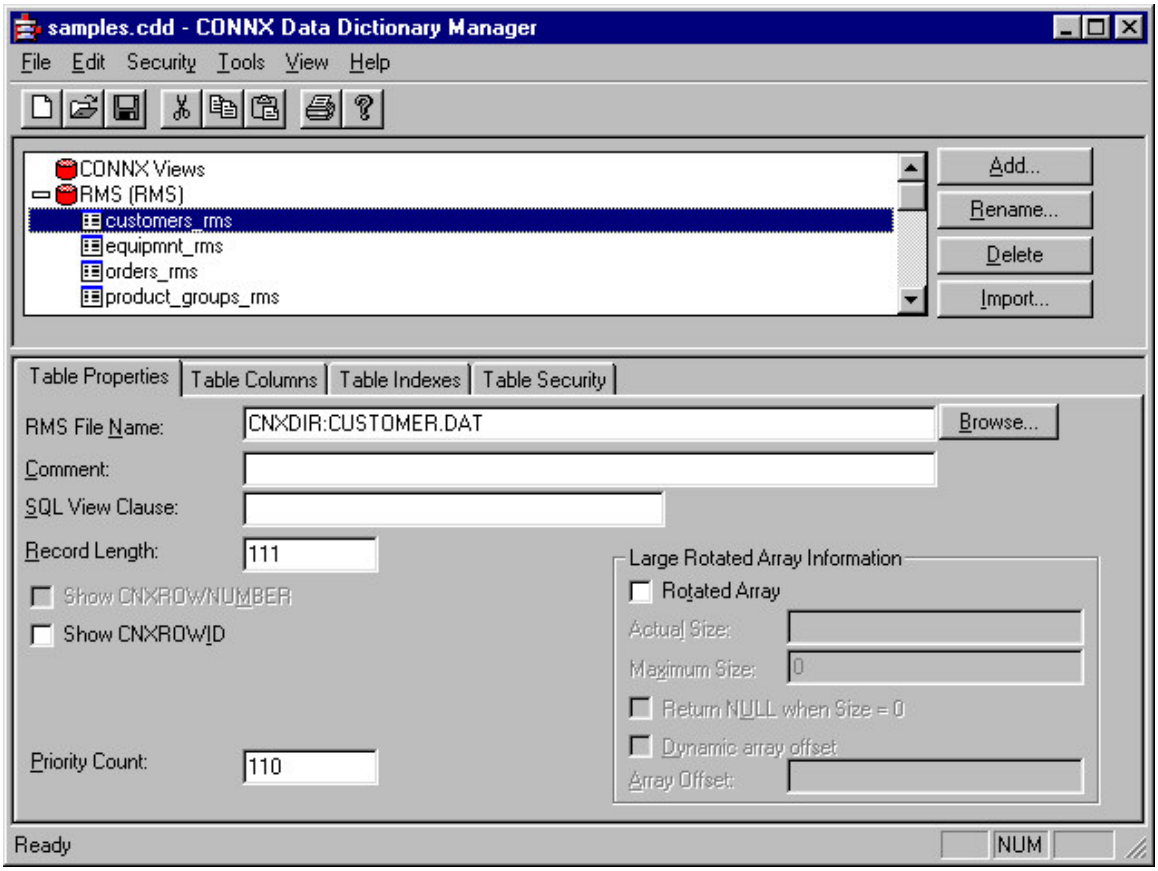
2. On the **View** menu, select **View By Database.**



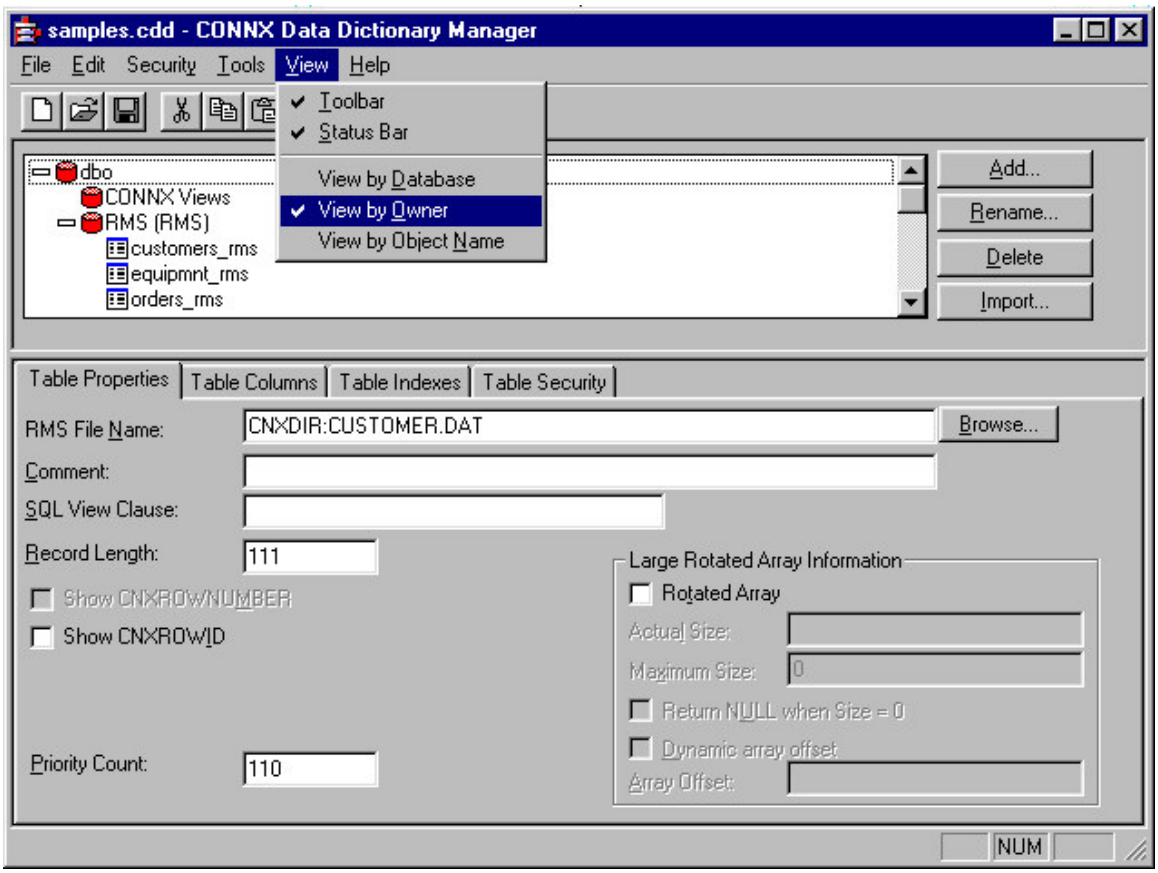
3. The database selections are available for viewing.

To view database information by owner

1. Select a CDD in the CONNX Data Dictionary window.



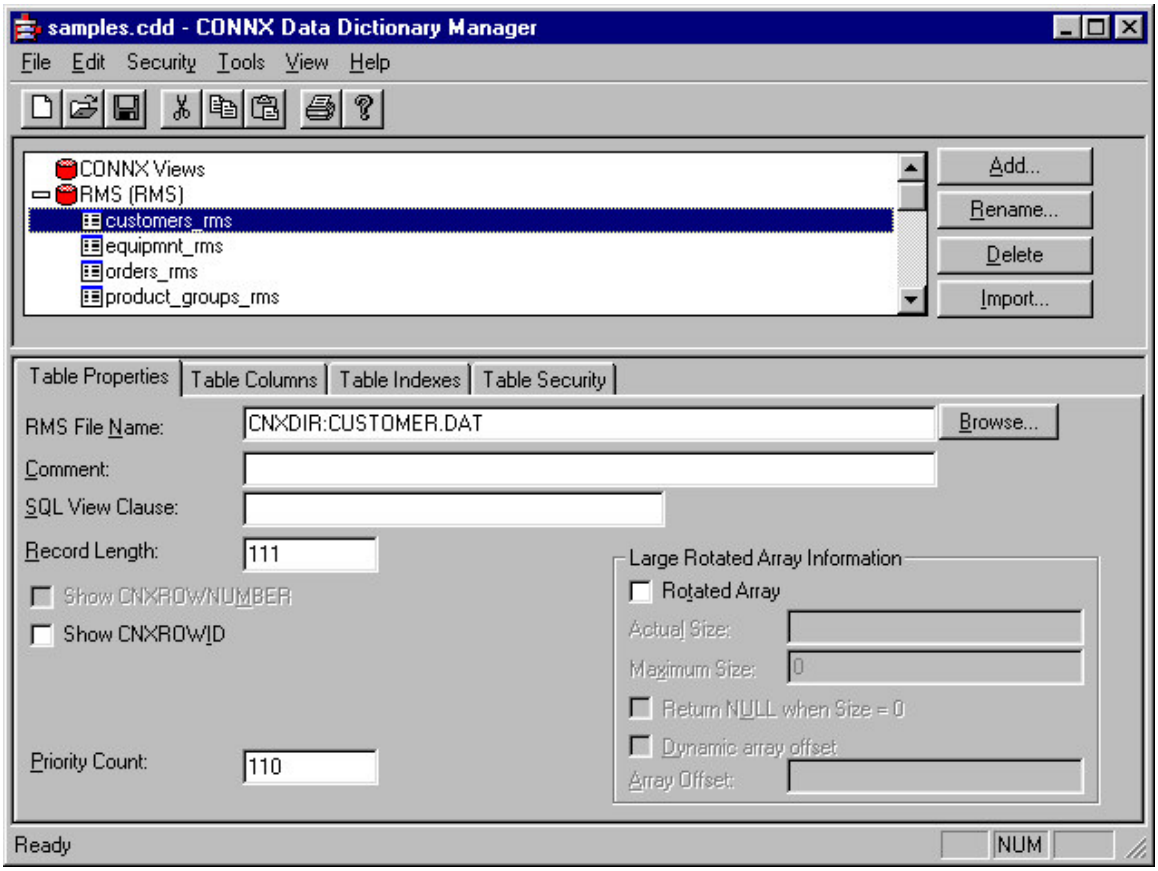
2. On the **View** menu, select **View By Owner.**



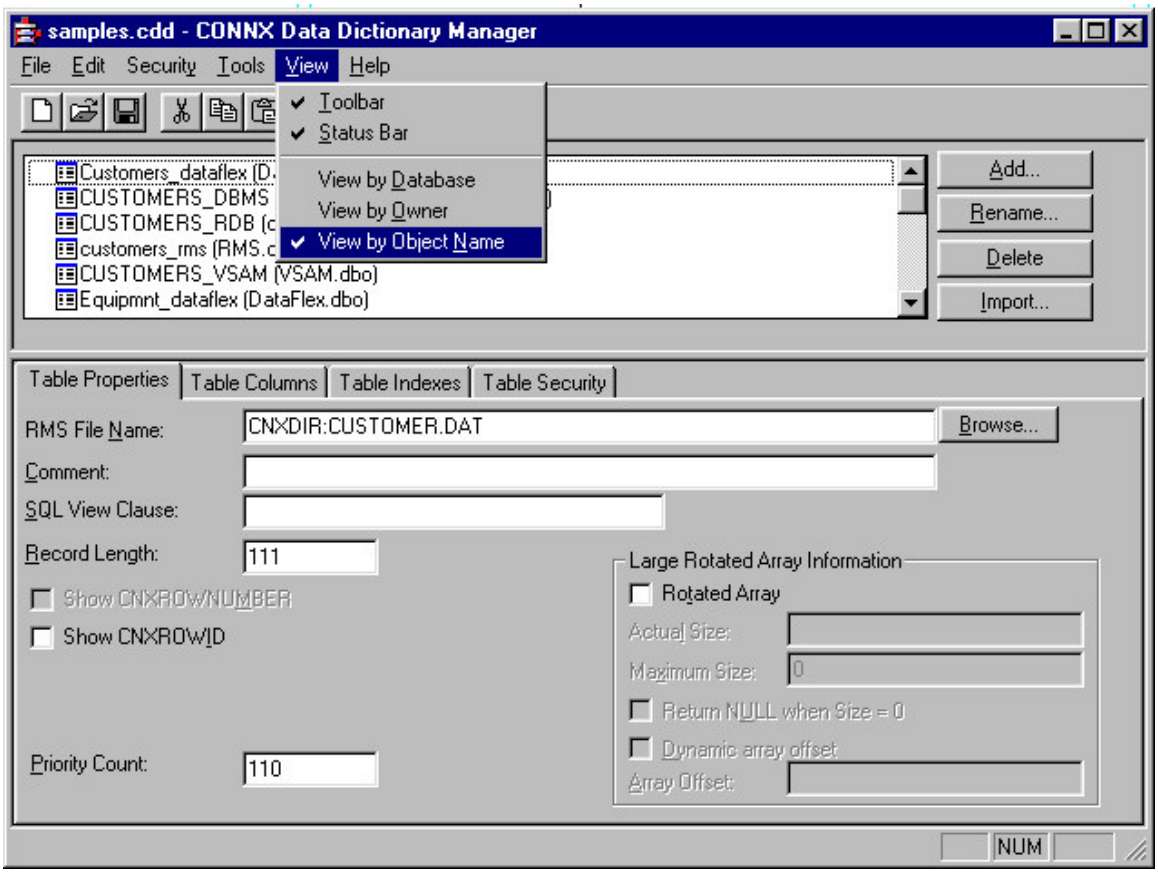
3. The database selections are available for viewing by owner name. A grouping appears for each user in the CONNX Data Dictionary.

To view database information by object name

1. Select a CDD in the CONNX Data Dictionary window.



2. On the **View** menu, select **View By Object Name.**



3. The database selections are available for viewing by object name.

CONNX Adapter: Enterprise and Database Modules

CONNX Adapter: OLE DB and ODBC Data Sources

The CONNX Enterprise OLE DB/ODBC module enables users to connect to most OLE DB- or ODBC-compliant data source tables.

The Enterprise version of the OLE DB/ODBC Adapter can be used to access data from any existing third-party ODBC or OLE DB data source, including IDMS, Ingres, UniVerse, or Unidata data sources for which a CONNX module does not exist.

The Desktop version of the OLE DB/ODBC Adapter can be used to provide read/write access to data sources normally stored on the desktop, in a hard drive, or on a network drive, including Paradox, FileMaker, FoxPro, Lotus Notes, dbase, Microsoft Excel, Microsoft Access, and delimited text files in database format.

Important: *The selected data source driver must be installed on all client machines that have access to the data source tables since they will be using the same DSN (Data Source Name) as described in ODBC Data Source Names Used with Multiple Users.*

Important: *The driver used to import table information must support ADO and be fully ODBC Level 2 compliant.*

Related Topics

- CONNX Enterprise OLE DB/ODBC Adapter
- CONNX OLE DB/ODBC Desktop Database Adapter
- CONNX SQL Server Module
- CONNX Sybase Data Module
- CONNX Informix Data Module

ODBC Data Source Names Used with Multiple Users

There are three types of ODBC DSNs that can be used when importing ODBC tables into CONNX: user, system, or file-based. The configuration of the DSN must be carefully considered whenever a CDD is intended for use by multiple users on multiple machines.

If ODBC database tables are imported with a user- or system-based DSN, every machine that will use this CONNX data dictionary will have to have the same DSN registered in its ODBC driver manager. For example, if the creator of the COMP_INVEN.CDD has a user DSN "My_inventory" that connects to the server machine warehouse, then every user of that CDD must have the same DSN "My_inventory" registered on their machine. To enable all users to access a newly added ODBC data source, it is recommended that you use a file-based DSN.

A file-based DSN is not specific to any machine or user and can be made accessible to all. The file-based DSN can be placed on a network share. However, with file-based DSNs the path entered by the user who imported the ODBC tables is stored in the CDD. For example, if the creator of the CDD file COMP_FDINV.CDD enters a file DSN of X:\ACCOUNTING\INVEN.DSN, all users with access to that CDD must be able to access X:\ACCOUNTING\INVEN.DSN.

One convention used to solve this scenario is that all user machines are given access to the same network drive. Another convention is to use a UNC (universal naming convention), such as \\MySERVER\Myshare\INDEV.DSN, which grants users access to the network drive without allocating a specific drive letter.

As with any database connection, if a DSN is modified, tables used within the CDD may need to be re-imported.

Regardless of DSN configuration, however, the ODBC driver or OLE DB provider must be installed on all client machines accessing the data source except in three-tier scenarios where it must be installed on the middle tier.

Related Topics

» Registering the Data Source Name

» To configure an existing data source

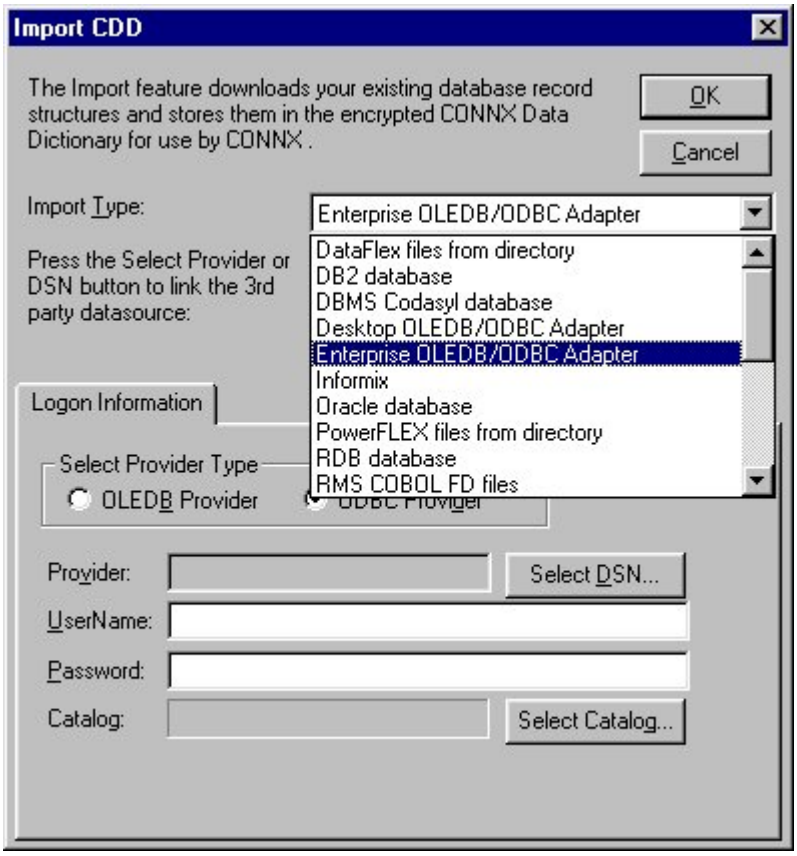
CONNX Adapter: OLE DB Data Sources

OLE DB Data Sources: Enterprise Modules

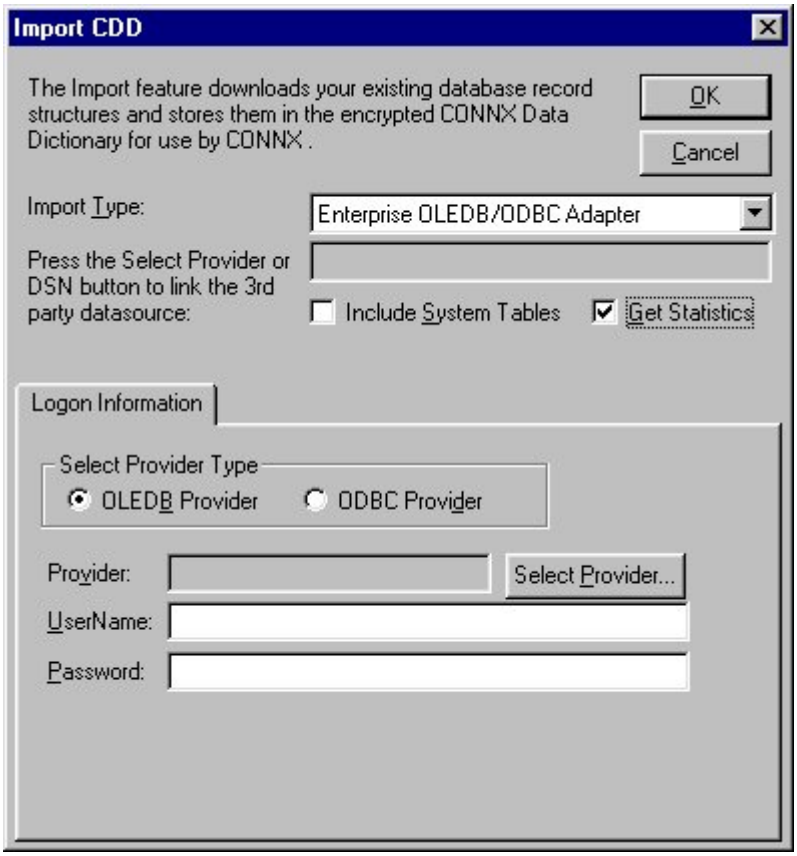
With the Enterprise version of the CONNX Adapter, users can access all data through a single driver and can quickly transfer data from one data source to another.

To import an existing table from an OLE DB-compliant data source using the CONNX Adapter

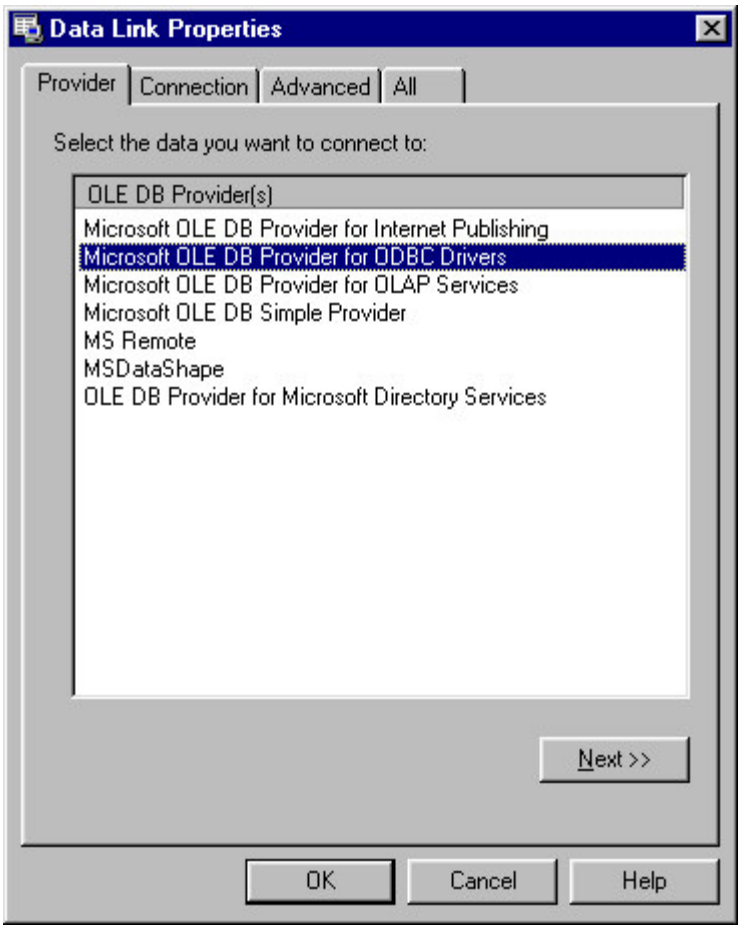
1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **Enterprise OLE DB/ODBC Adapter** from the **Import Type** list box in the **Import CDD** dialog box.



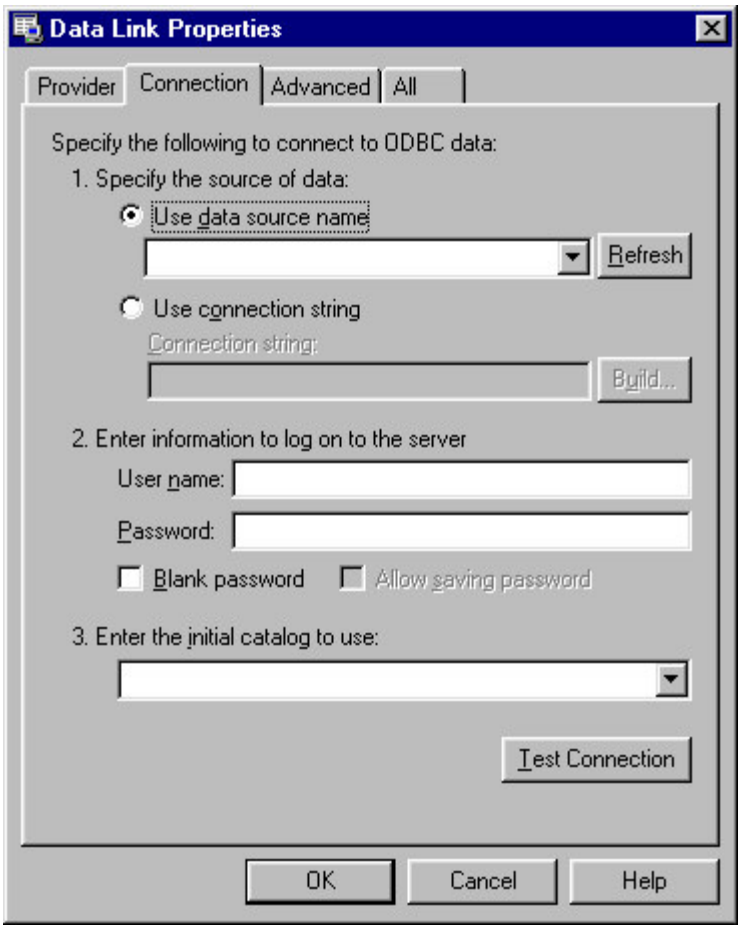
3. Select **OLE DB Provider** under **Select Provider Type,** and then click the **Select Provider** button in the **Logon Information** pane.



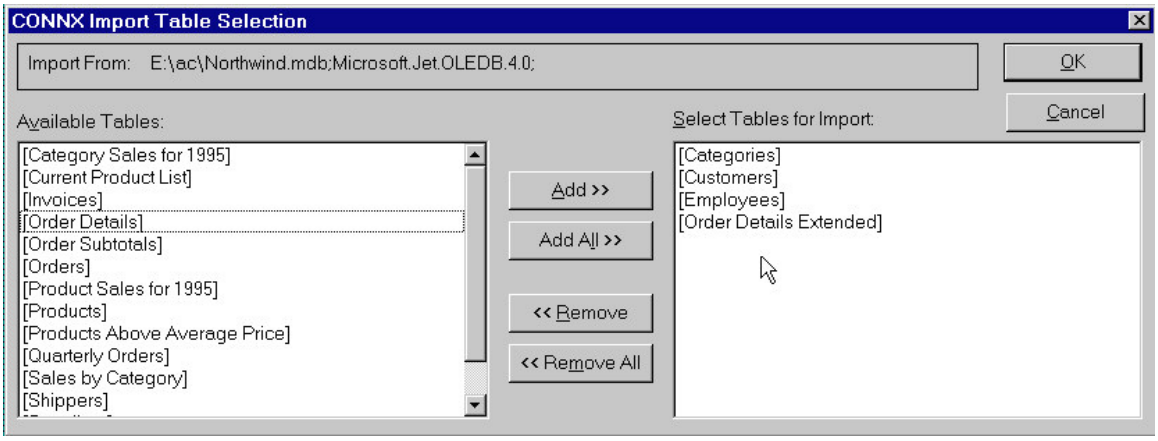
4. Select an **OLE DB Provider** from the list shown in the **Data Link Properties** dialog box. Click the **Next** button or the **Connection** tab.



5. Enter a data source name in **Item 1** on the **Connection** tab in the **Data Link Properties** dialog box.



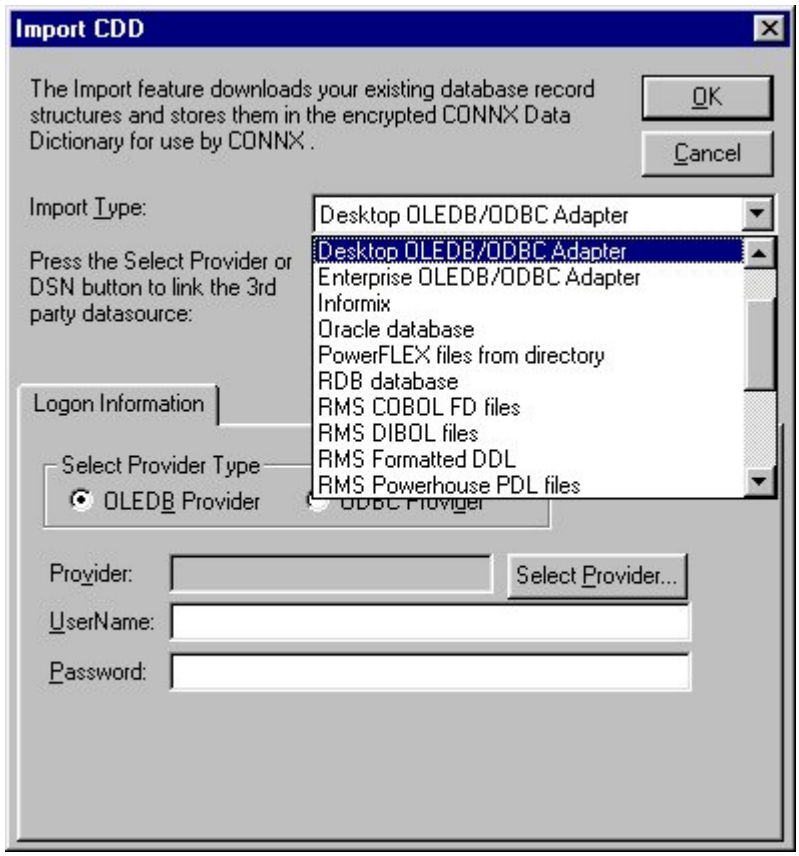
6. Enter a user name and password in section 2.
7. Select an object from the list box in **Item 3.** Click the **Test Connection** button to verify that it is available. Click the **OK** button.
8. The **Import CDD** dialog box appears. Reenter the user name and password, if required.
9. Normally, only user-defined tables can be selected for import. Select the **Include System Tables** check box to enable the import of non-user-defined tables.
10. Select the **Get Statistics** check box to identify the table sizes. This is used by CONNX query optimization. Click the **OK** button.
11. The **CONNX Import Table Selection** dialog box appears. Click the **Add** or **Add All** button to move the database objects to the **Select Tables for Import** pane.



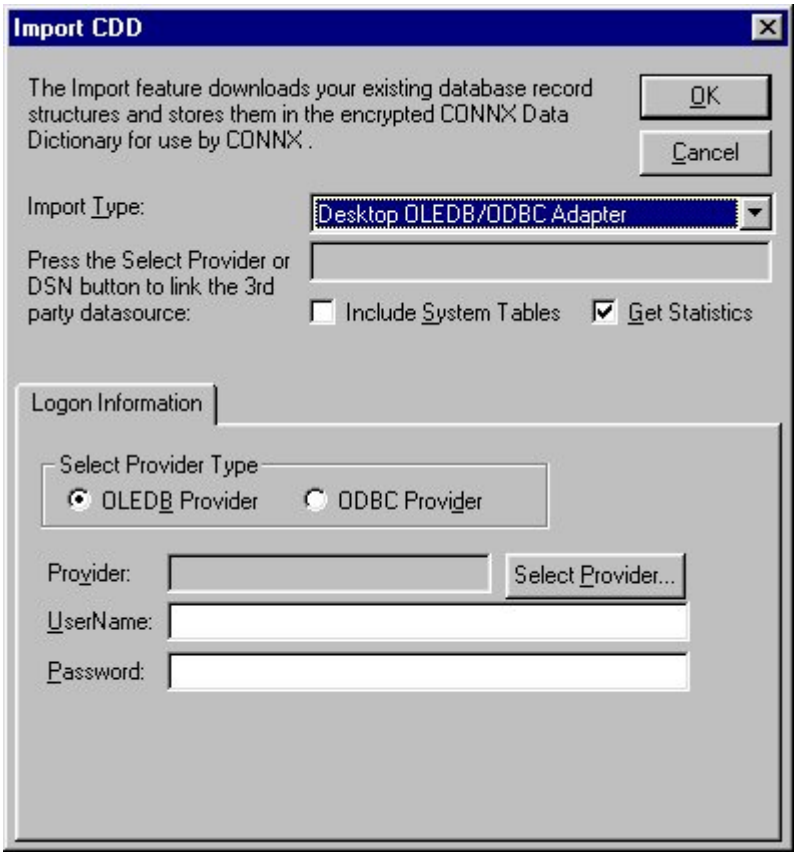
12. Click the **OK** button to import the selected objects into CONNX. The imported catalogs appear in the list of accessible objects in the CONNX Data Dictionary Manager window.

To import an existing table from an OLE DB-compliant provider data source using the Desktop OLE DB Adapter

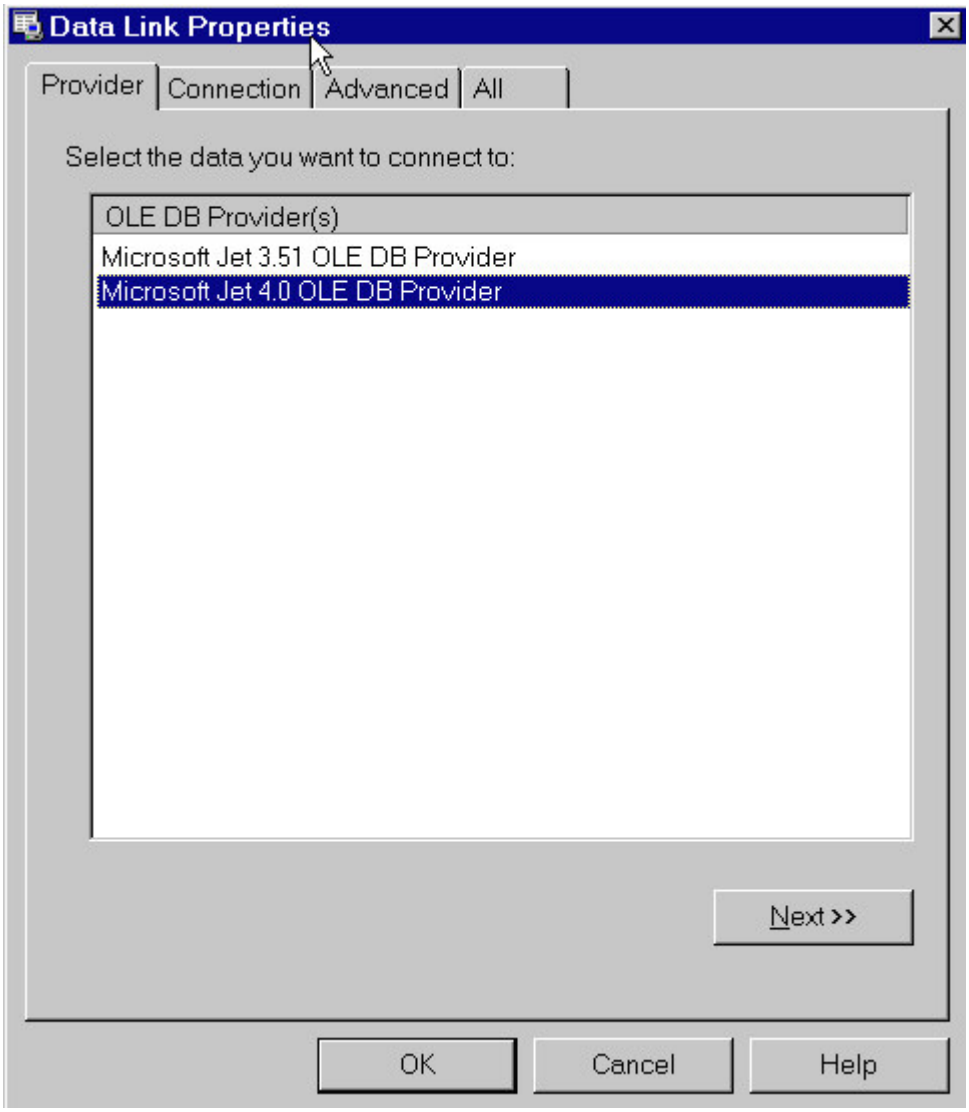
1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **Desktop OLE DB/ODBC Adapter** from the **Import Type** list box.



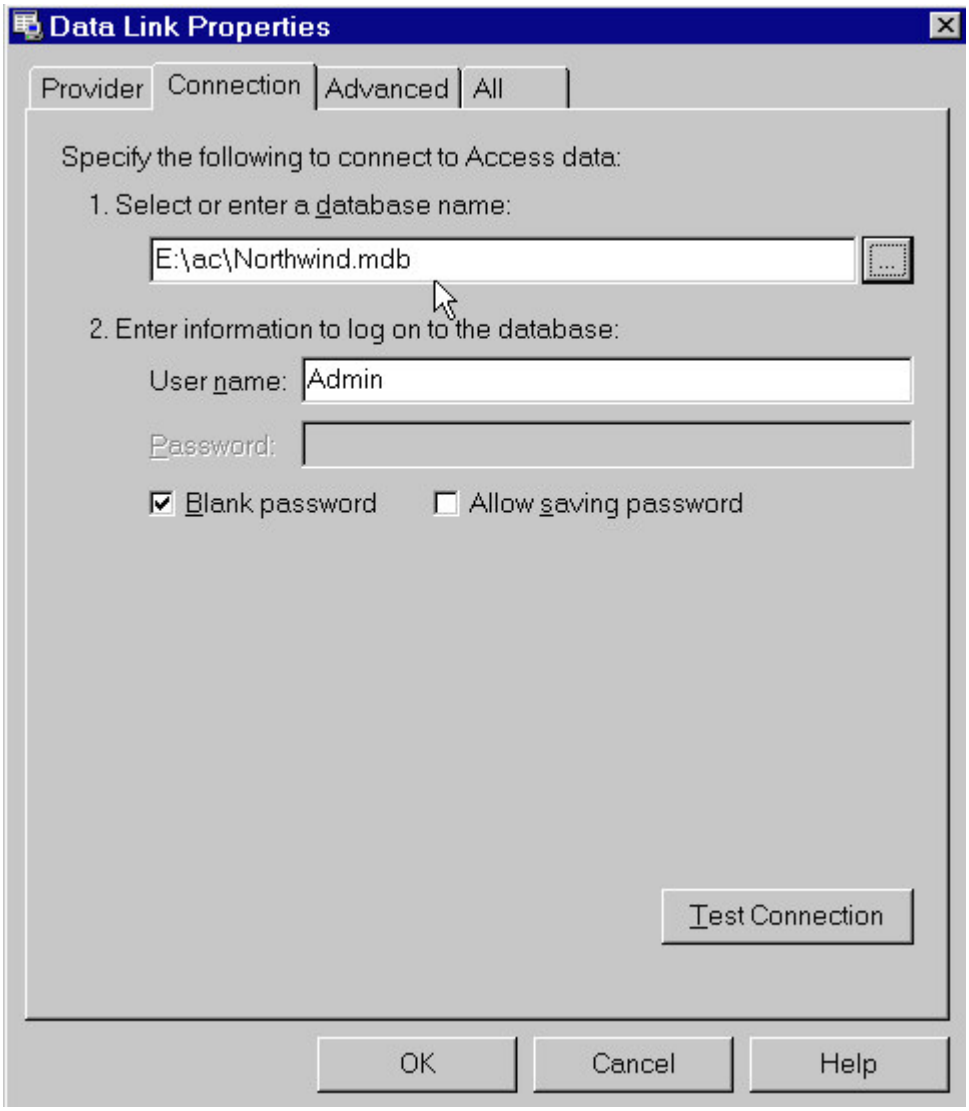
3. Select **OLE DB Provider** under **Select Provider Type,** and then click the **Select Provider** button in the **Logon Information** pane.



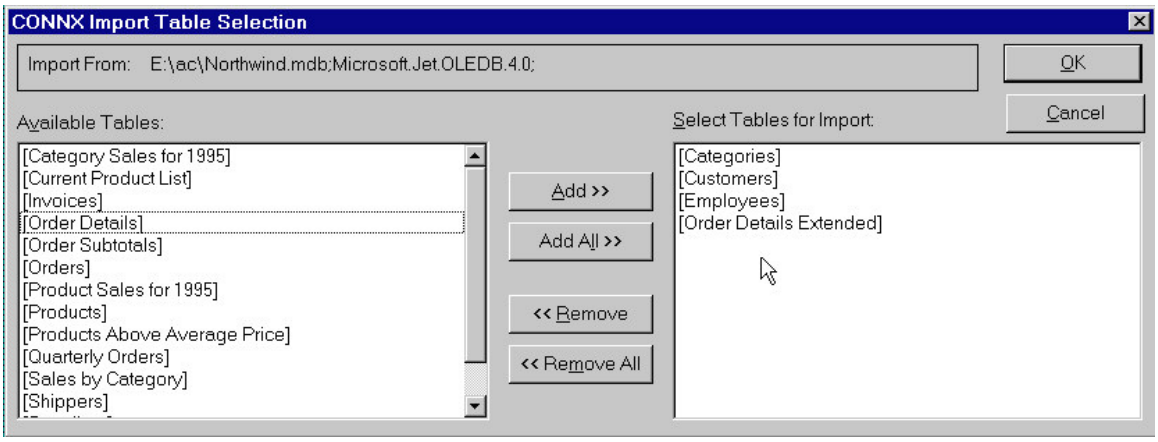
4. Select a data source from the list provided in the **Data Link Properties** dialog box. Click the **Next** button or the **Connection** tab.



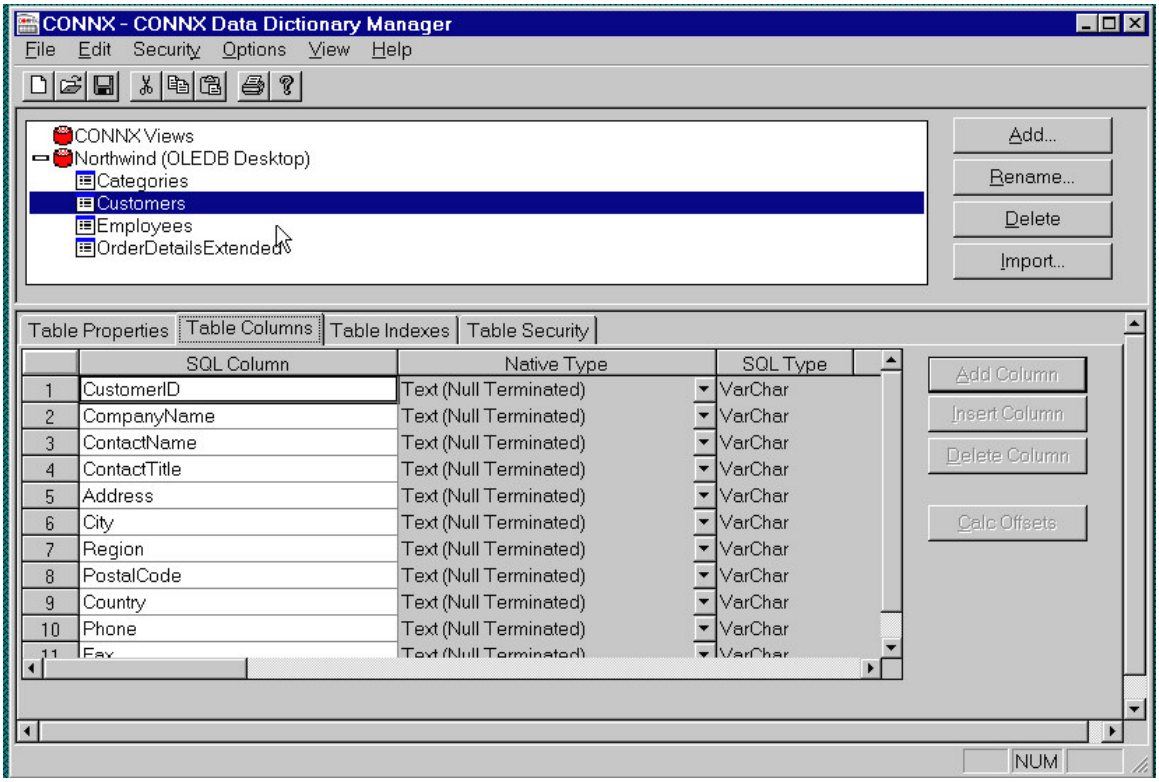
5. Enter a path name to an available file in Item 1 on the **Connection** tab in the **Data Link Properties** dialog box, or click the **Browse (...)** button to locate a database file. Use the full UNC path if the data source is to be accessed from multiple clients.



6. Enter the user name and password for the data source. (The default user name for Access is Admin. No password is required.) Clear the **Blank Password** check box to specify passwords. Select the **Select Allow Saving of Password** check box to confirm the password in the CONNX CDD. Click the **Test Connection** button to verify that the database is available.
7. Click the **OK** button to return to the **Import CDD** dialog box.
8. Normally, only user-defined tables can be selected for import. Select the **Include System Tables** check box to access system table definitions.
9. Select the **Get Statistics** check box to identify the table sizes. This is used for query optimization, using the table sizes as a guide.
10. Click the **OK** button. The **CONNX Import Table Selection** dialog box appears with a list of available tables. Click the **Add** or **Add All** button to move the selected tables to the **Select Tables for Import** pane.



11. Click the **OK** button to import the selected tables into CONNX. The imported tables are added to the list of accessible objects in the CONNX Data Dictionary Manager window.



CONNX Adapter: ODBC Data Sources

ODBC Data Sources: Desktop Modules

The CONNX OLE DB Desktop Database Adapter can be used with many data sources, including Paradox, dbase, FoxPro, Microsoft Access, Microsoft Excel, and delimited text files (read only).

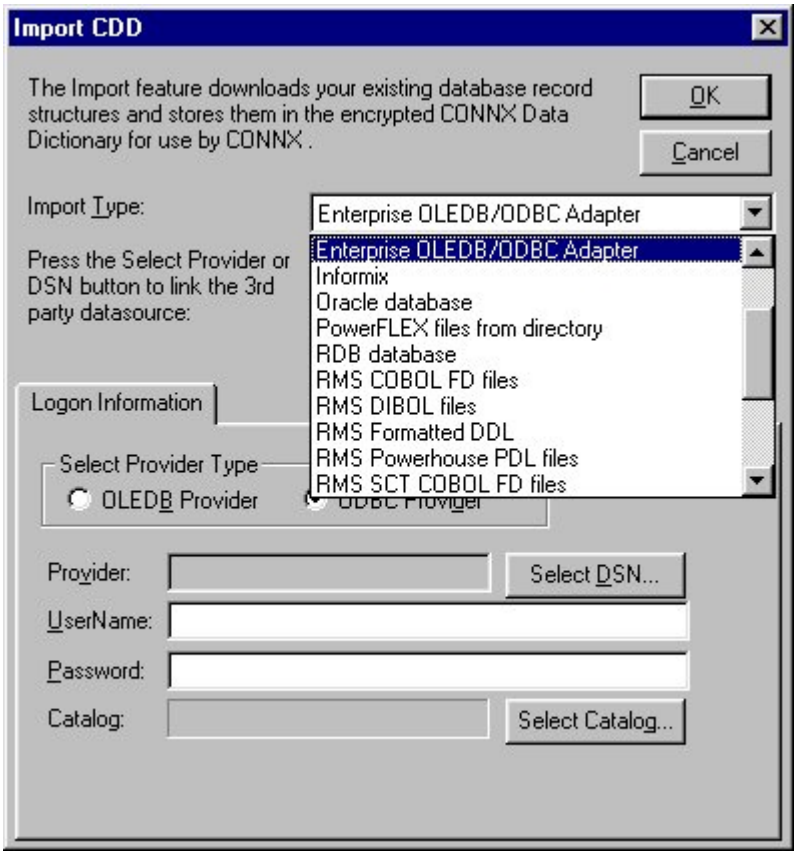
Related Topics

» To import an existing OLE DB-compliant data source table

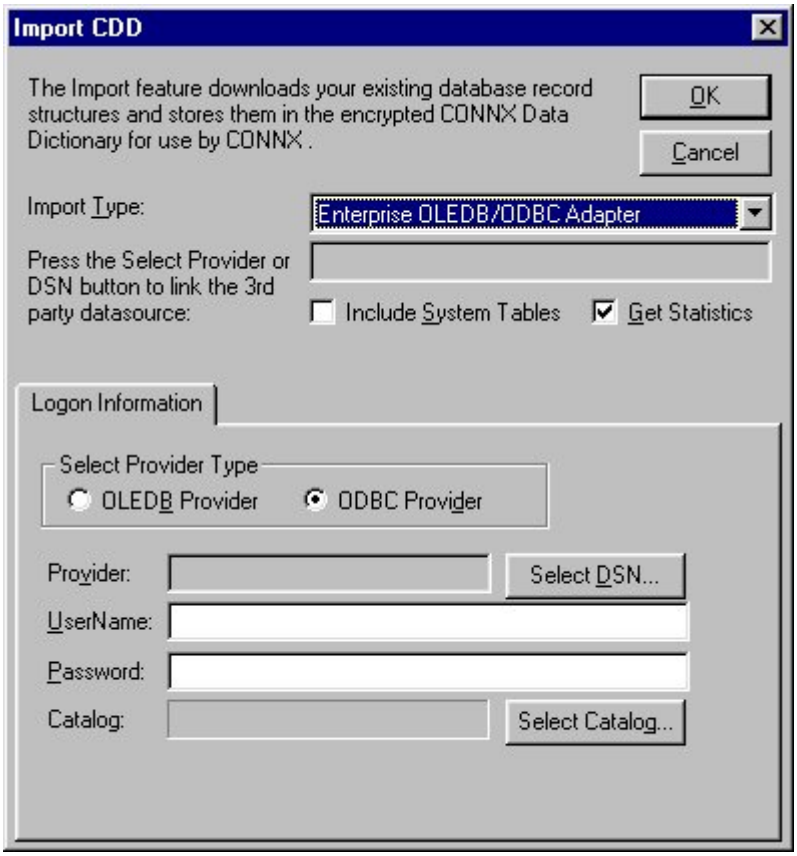
» To import objects from an ODBC provider data source

To import an existing table from an ODBC-compliant provider data source using the CONNX Adapter - Enterprise Module

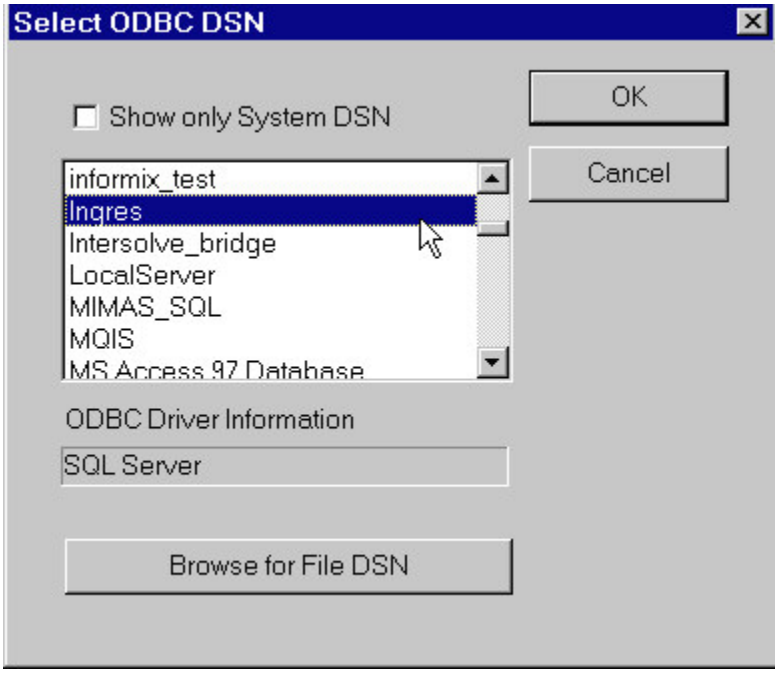
1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **Enterprise OLE DB/ODBC Adapter** from the **Import Type** list box.



3. Select **ODBC Provider** under **Select Provider Type,** and then click the **Select DSN** button in the **Logon Information** pane.



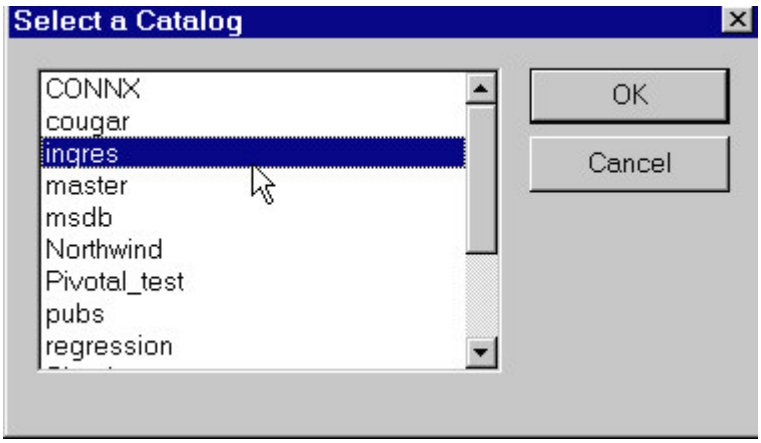
4. The **Select ODBC DSN** dialog box appears. You can choose either a user, system, or file-based DSN, although it is recommended that a file DSN be used, since it can be accessed through a network by multiple users.



- **File DSN (Recommended)**
  Select the Browse for File DSN check box to open the Open dialog box. Select a file DSN from lists of available .dsn files.
- **System DSN or User DSN**
  Select the Show Only System DSN check box to view a list of system DSNs only. Clear the check box to show a list of both user and system DSNs. Note that the ODBC Driver Information text box displays information about the selected ODBC DSN. The same information can be found in the ODBC Data Source Administrator dialog box. Click the **OK** button in the **Select ODBC DSN** dialog box to return to the **Import CDD** dialog box, which displays the selected DSN.

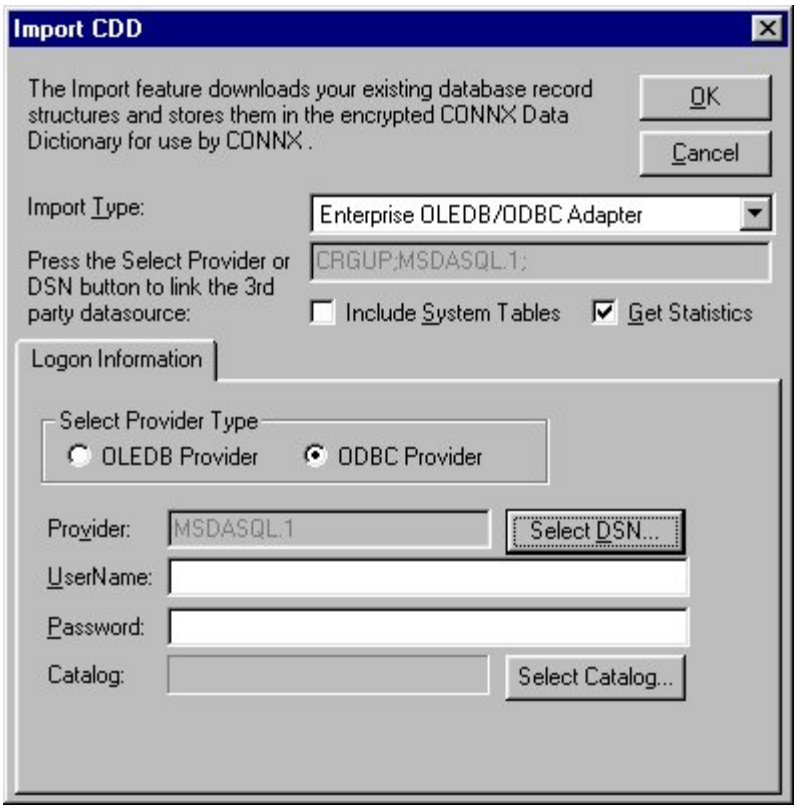
5. The **Import CDD** dialog box appears. Some ODBC databases support catalogs. To specify a catalog, enter a name and password for the database, and then click the **Select Catalog** button to log into the database and view a list of available catalogs.

The **Select a Catalog** dialog box appears.



Your ODBC data source may require additional information. Review the documentation provided by your driver vendor for additional configuration options.

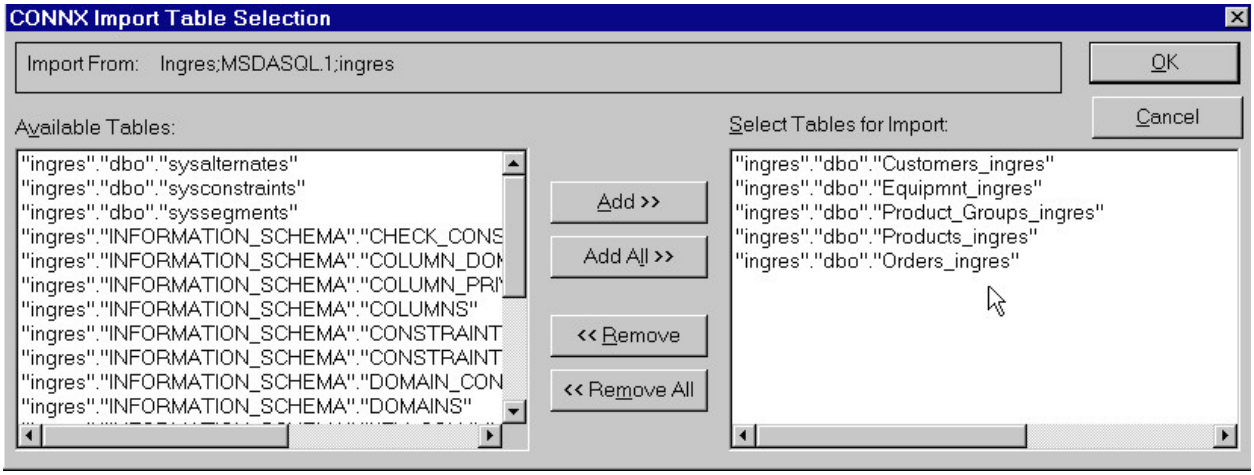
6. Select a catalog, and then click the **OK** button to return to the **Import CDD** dialog box. Normally, only user-defined tables can be selected for import.



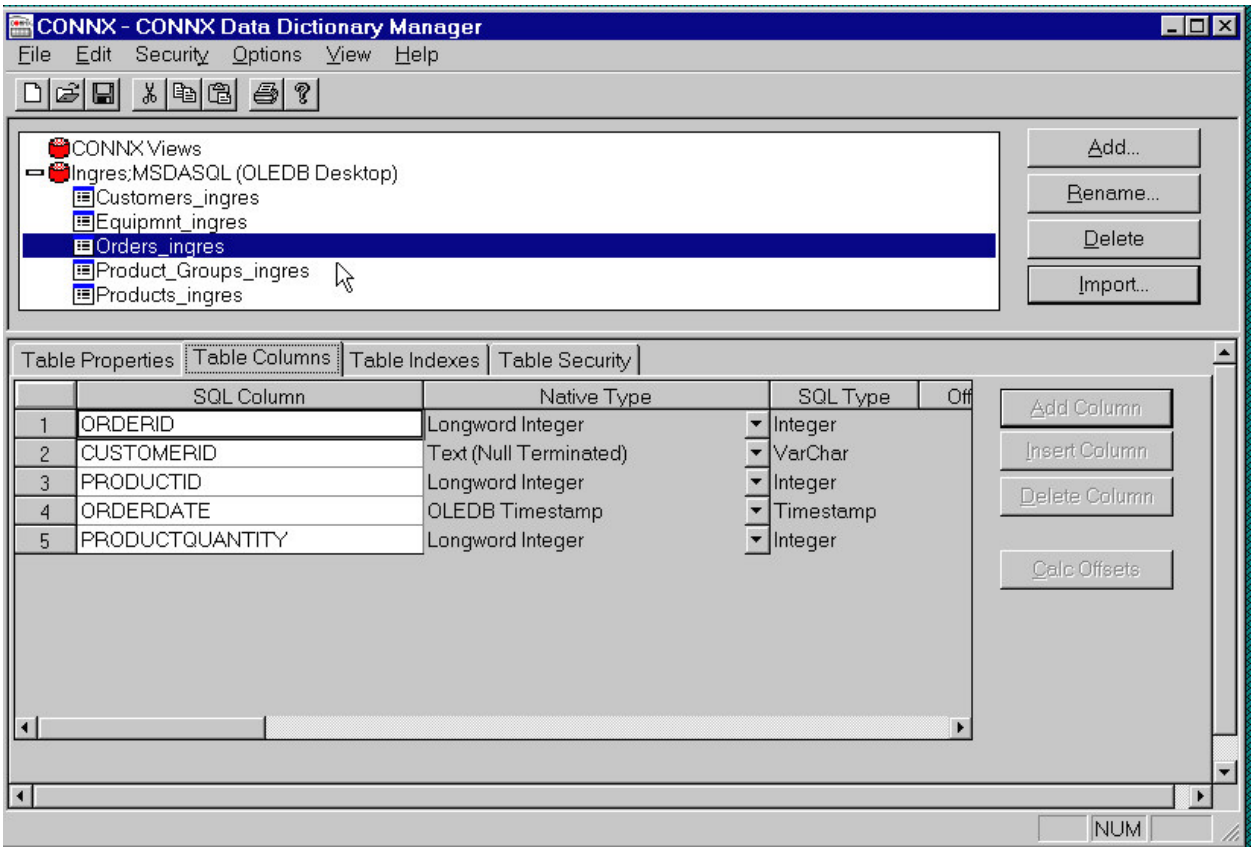
7. Select the **Include System Tables** check box to access system tables.

8. Select the **Get Statistics** check box to identify table sizes. This is used by CONNX query optimization.

9. Click the **OK** button. The **CONNX Import Table Selection** dialog box appears with a list of available table names. Click the **Add** or **Add All** button to move the tables to the **Select Tables for Import** pane.

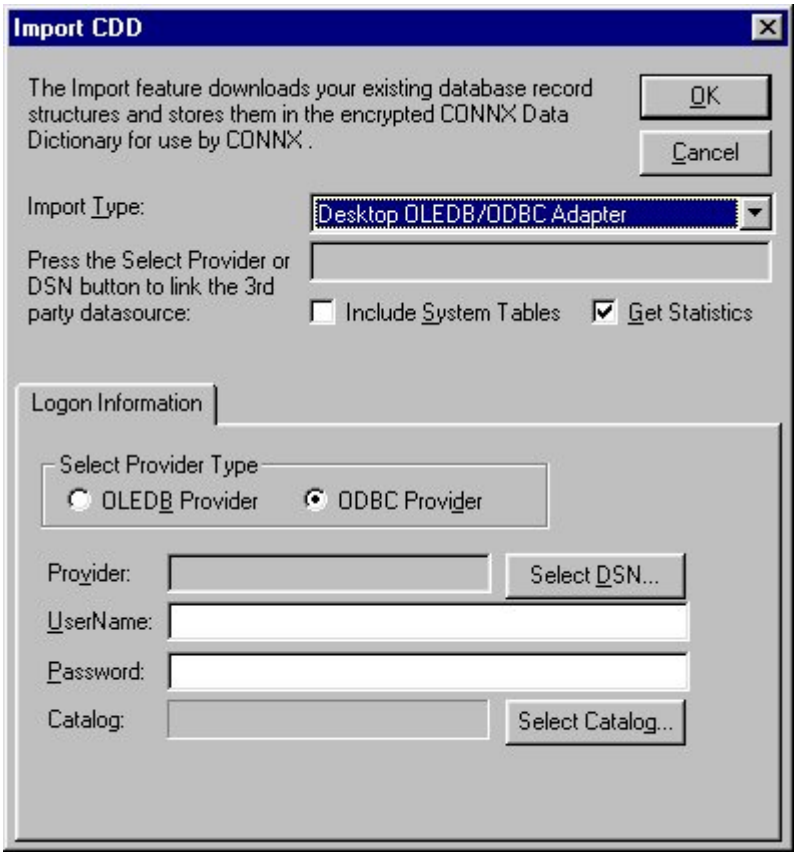


10. Click the **OK** button to import the selected tables into CONNX. The imported tables are added to the list of accessible objects in the CONNX Data Dictionary Manager window.

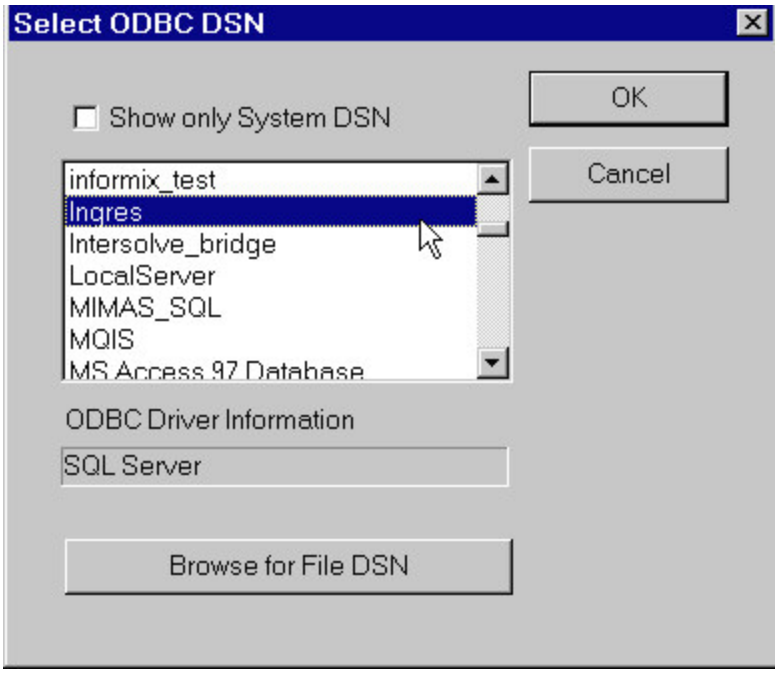

CONNX - CONNX Data Dictionary Manager
File Edit Security Options View Help
CONNX Views
Ingres;MSDASQL (OLEDB Desktop)
Customers_ingres
Equipmnt_ingres
Orders_ingres
Product_Groups_ingres
Products_ingres
Add...
Rename...
Delete
Import...
Table Properties
Table Columns
Table Indexes
Table Security
SQL Column
Native Type
SQL Type
Off
1 ORDERID Longword Integer Integer
2 CUSTOMERID Text (Null Terminated) VarChar
3 PRODUCTID Longword Integer Integer
4 ORDERDATE OLEDB Timestamp Timestamp
5 PRODUCTQUANTITY Longword Integer Integer
Add Column
Insert Column
Delete Column
Calc Offsets
NUM

To import an existing table from an ODBC-compliant data source object using the CONNX Adapter - Desktop Module

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **Desktop OLE DB/ODBC Adapter** from the **Import Type** list box.



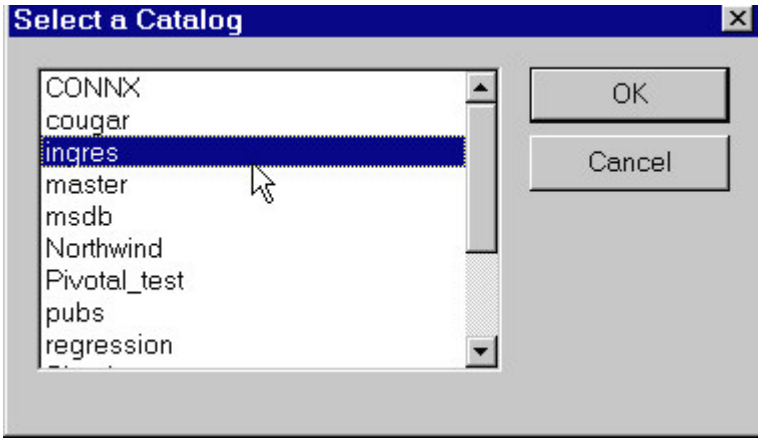
3. Select **ODBC Provider** under **Select Provider Type,** and then click the **Select DSN** button in the **Logon Information** pane.



4. The **Select ODBC DSN** dialog box appears. You can choose either a user, system, or file-based DSN, although it is recommended that a file DSN be used, since it can be accessed through a network by multiple users.

    5. **File DSN (Recommended)**
    Select the Browse for File DSN check box to open the Open dialog box. Select a file DSN from lists of available .dsn files.
    6. **System DSN or User DSN**
    Select the Show Only System DSN check box to view a list of system DSNs only. Clear the check box to show a list of both user and system DSNs. Note that the ODBC Driver Information text box displays information about the selected ODBC DSN. The same information can be found in the ODBC Data Source Administrator dialog box. Click the **OK** button in the Select ODBC DSN dialog box to return to the **Import CDD** dialog box, which displays the selected DSN.

5. The **Import CDD** dialog box appears. Some ODBC databases support catalogs. To specify a catalog, enter a name and password for the database, and then click the **Select Catalog** button to log into the database and view a list of available catalogs. The **Select a Catalog** dialog box appears.



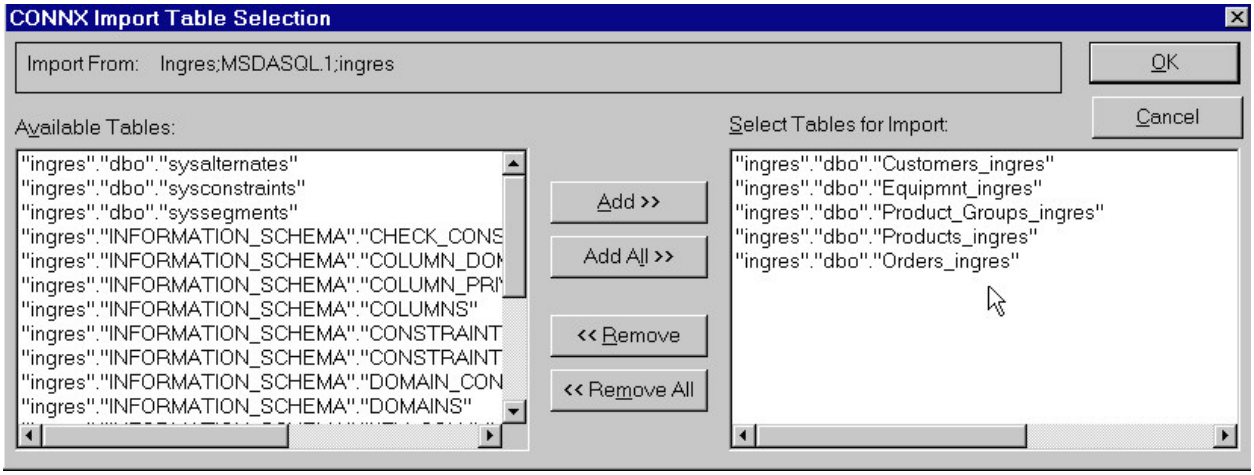
Your ODBC data source may require additional information. Review the documentation provided by your driver vendor for additional configuration options.

6. Select a catalog, and then click the **OK** button to return to the **Import CDD** dialog box. Normally, only user-defined tables can be selected for import.

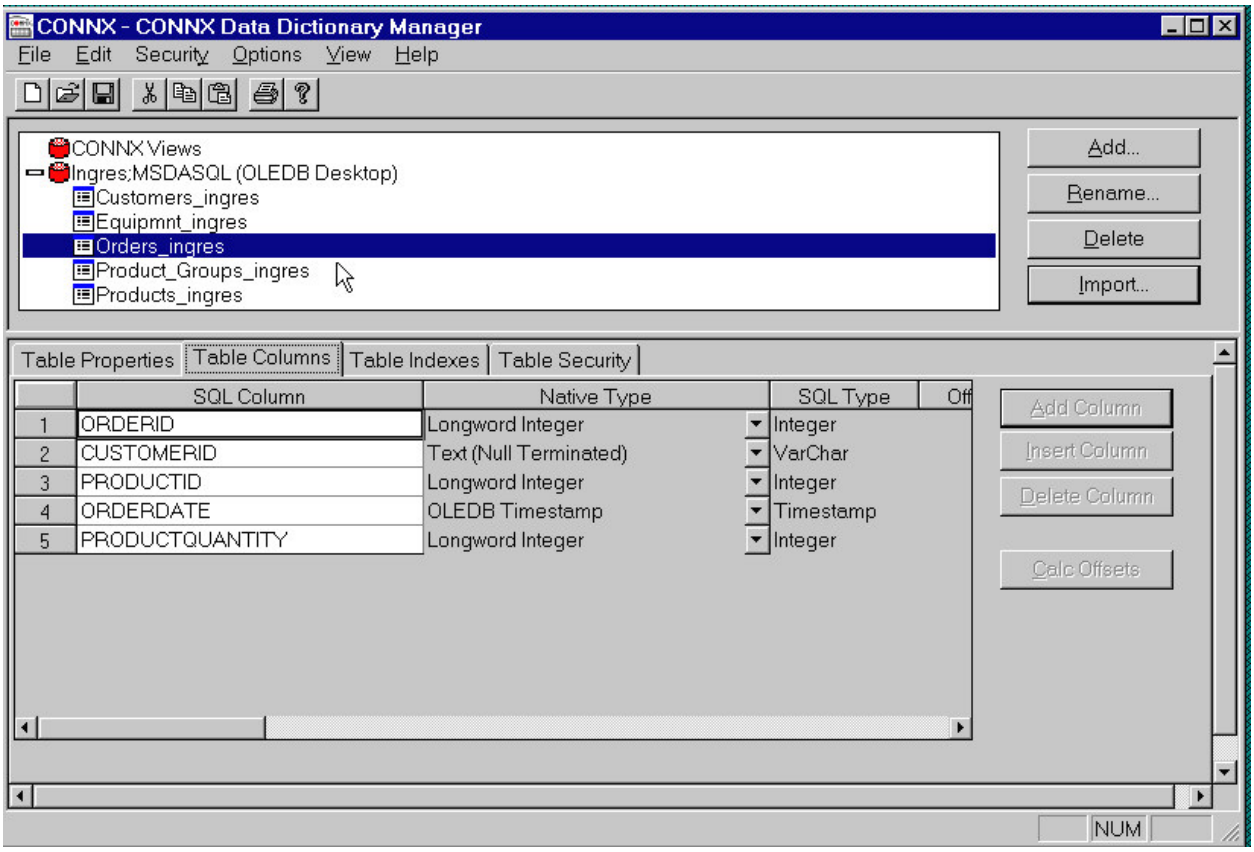
7. Select the **Include System Tables** check box to access system tables.

8. Select the **Get Statistics** check box to identify table sizes. This is used by CONNX query optimization.

9. Click the **OK** button. The **CONNX Import Table Selection** dialog box appears with a list of available table names. Click the **Add** or **Add All** button to move the tables to the **Select Tables for Import** pane.



10. Click the **OK** button to import the selected tables into CONNX. The imported tables are added to the list of accessible objects in the CONNX Data Dictionary Manager window.


CONNX - CONNX Data Dictionary Manager
File Edit Security Options View Help
CONNX Views
Ingres;MSDASQL (OLEDB Desktop)
Customers_ingres
Equipmnt_ingres
Orders_ingres
Product_Groups_ingres
Products_ingres
Add...
Rename...
Delete
Import...
Table Properties
Table Columns
Table Indexes
Table Security
SQL Column
Native Type
SQL Type
Off
1 ORDERID Longword Integer Integer
2 CUSTOMERID Text (Null Terminated) VarChar
3 PRODUCTID Longword Integer Integer
4 ORDERDATE OLEDB Timestamp Timestamp
5 PRODUCTQUANTITY Longword Integer Integer
Add Column
Insert Column
Delete Column
Calc Offsets
NUM

Adabas SQL Gateway Import Types

CONNX for Adabas has three types of imports:

1. Adabas DYNAMIC DDL Import
2. Adabas FDT Import
3. Adabas SYSOBJH Import

Adabas Dynamic DDL Import offers the ability to use Create Table Description and Create Cluster Description syntax to enable users to import metadata from Adabas scripts.

Each of those imports are responsible for accessing the Adabas metadata and storing this in the CONNX Data Dictionary. Although Adabas has arrays (MUs), groups (PEs), and arrays within groups (MUPE), CONNX represents these as logical relational tables after the import is performed.

Limitations and Suggestions

1. If more than one MU has been logically defined within a Create Table Description that represents a Periodic Group, CONNX cannot import all of the MUs. This is a limitation that will be solved on future versions of the product.
2. Group Definitions are not imported, but their fields are.
3. Phonetic and hyperdescriptor keys are not imported.
4. Other than renaming, it is best not to delete or change field information generated from the imports. CONNX Views are a simple way of creating logical views from selected columns representing a table or tables that are joined together.

Import from Natural

Introduction

The CONNX for Adabas product is also known as the Adabas SQL Gateway. In order to import Adabas metadata, it is necessary to first generate DDM extract files using the SYSOBJH utility (Object Handler). The extract file will serve as input to the CONNX Data Dictionary Tool. For details on this utility please see the following locations:

For Mainframe:

http://documentation.softwareag.com/natural/nat823mf/utis/sysobjh.htm#sysobjh

For OpenSystems:

Windows: http://documentation.softwareag.com/natural/nat6314win/utis/sysobjh.htm

Unix: http://documentation.softwareag.com/natural/nat6314unx/utis/sysobjh.htm

For OpenSystems imports, use the instructions located at To generate an OpenSystems SYSOBJ extract file

For mainframe imports, use the instructions located at To generate a Mainframe SYSOBJ extract file

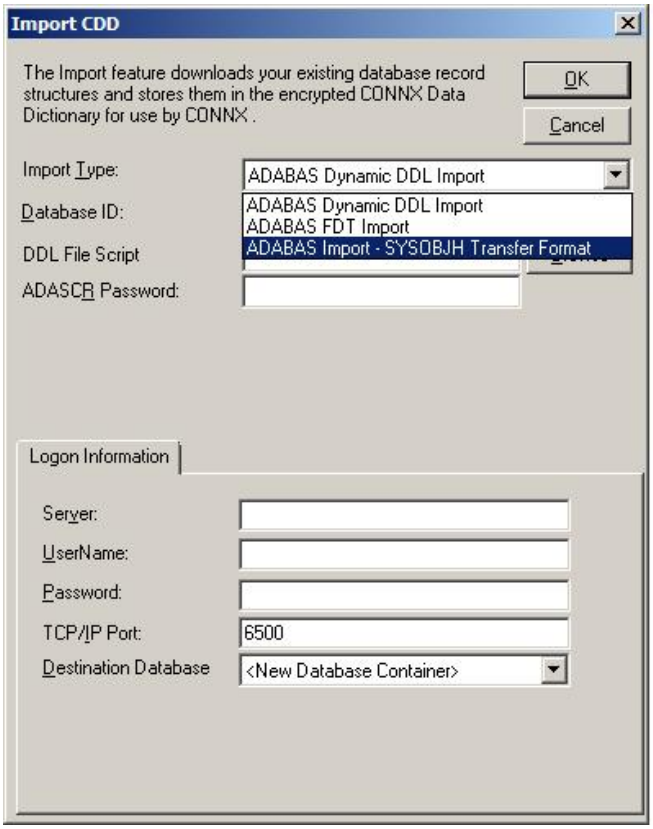
To import from Adabas FDT, use the instructions located at To import Adabas FDT (read field definitions).

To import from Adabas files (Natural)

Notes:

- *If you wish to retain hyphens during a **Windows** import, you must first set the KeepHyphens variable. The default is to have it unset (0).*
- *If you desire to have Natural Logicals be treated as a Bit instead of a Byte (Default), you must set the ADA_NATURALBYTEASBIT variable to 1. The default is to treat Natural Logicals as a Byte field.*
- *On Empty Fields (Fields in which the values are 0 or an empty string), CONNX treats these fields as a SQL Null by default. If you desire to have the fields be treated as a 0 or an empty string, you must set the variable ESQLNULL to 0.*

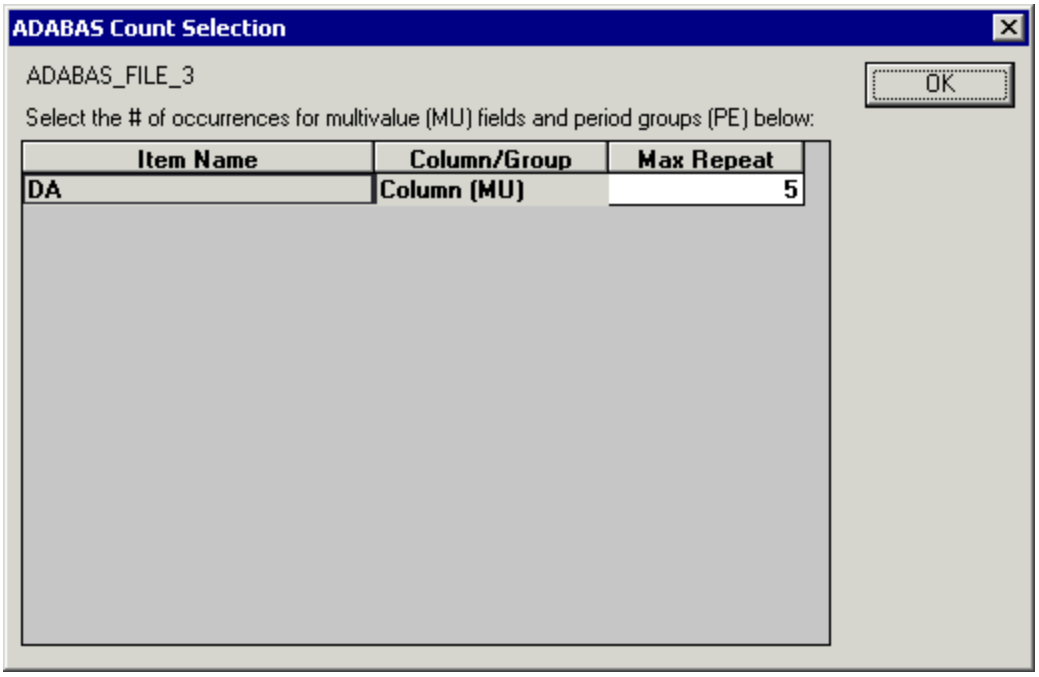
1. The Adabas SQL Gateway server setup must be complete, and the CONNX Listener must be running on the mainframe.
2. Click the **Import** button in the CONNX Data Dictionary Manager window.
3. The **Import CDD** dialog box appears.



4. Select **Adabas Import SYSOBJ Transfer Format** in the **Import Type** list box.
5. Enter a **Dictionary Database ID** in the text box.
6. Enter the name of a SYSOBJ extract file. If necessary, use the **Browse** button to locate a file on the network. Instructions on how to do are at OpenSystems are and Mainframe.
7. Enter an **ADASCR password,** if you are using the ADASCR security method.
8. If the database you are importing from is located on the same Windows machine that you are importing to, enter "localhost" in the **Server** text box. For all other instances, enter the TCP/IP address or server name in the **Server** text box. The CONNX Listener Task attempts to access the

given server. If the server is unavailable or cannot be located, the following message appears: "The CONNX Listener process (CNXRUN##_MAIN) is not running on the system."

9. Enter a CONNX user name and password.
10. Enter a TCP/IP port number. 6500 is the default TCP/IP port.
11. Select a **Destination Database** for the imported tables. See Adding a Database Connection for more information.
12. Click the **OK** button.
13. The **CONNX Import Table Selection** dialog box appears. Select each file to import, and then click the **Add** or **Add All** button.
14. The **Adabas Count Selection** dialog box appears. Enter the number of maximum occurrences of each field under **Max Repeat,** and then click the **OK** button.



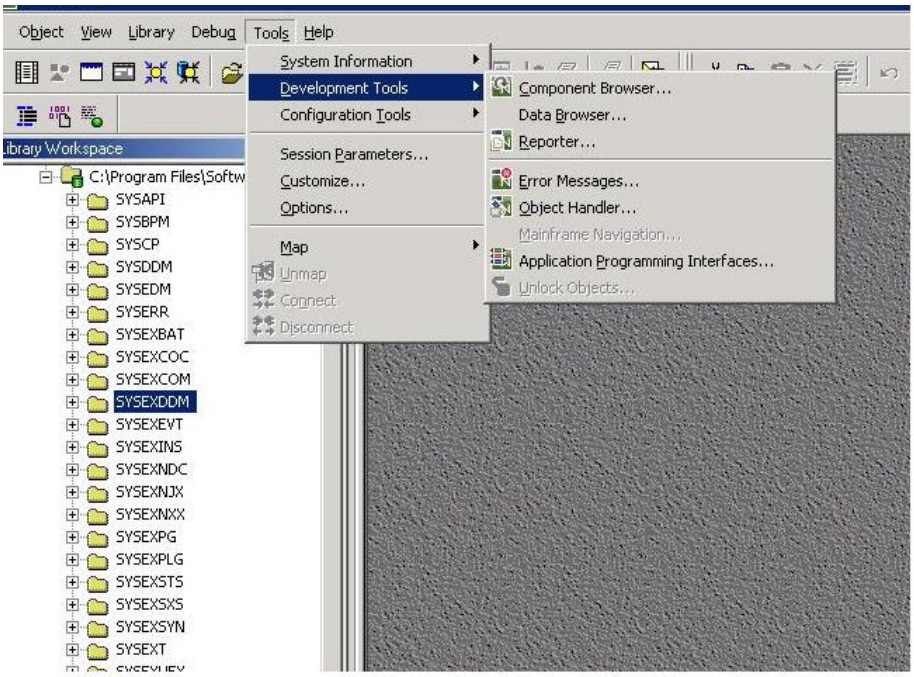
15. Save the CDD by selecting the **File** menu and then clicking **Save.** The CDD appears under Adabas in the upper pane of the CONNX Data Dictionary Manager.

Extract Files

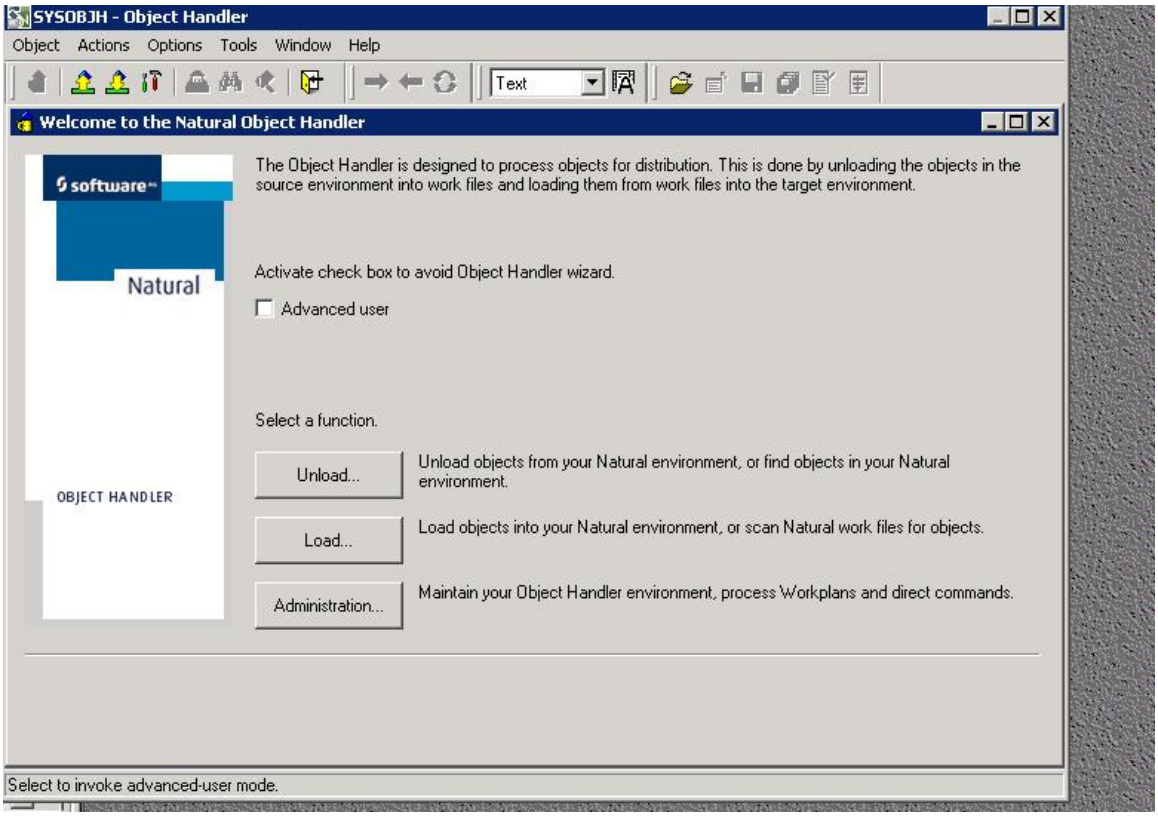
Windows

To generate an OpenSystems SYSOBJ extract file

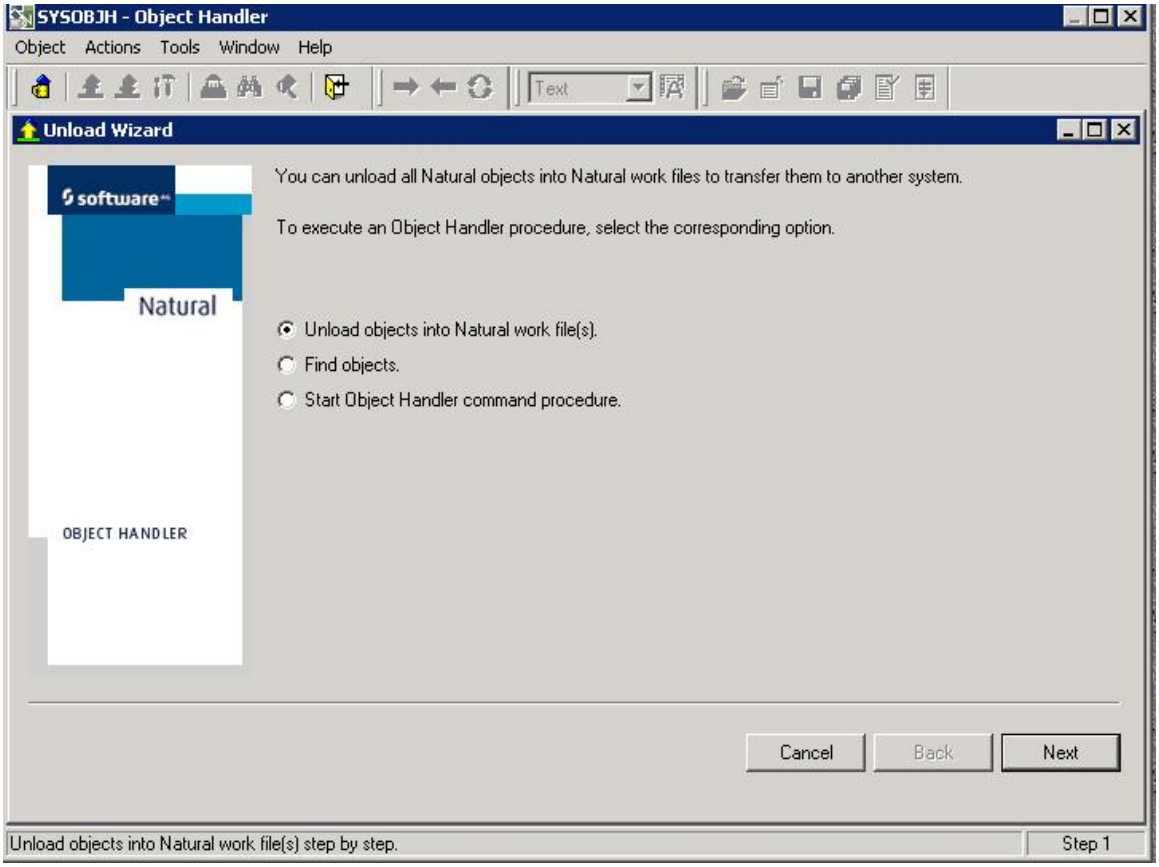
1. Before continuing with import procedures for CONNX for Adabas, it is necessary to first import the Natural DDM files, using the Natural Object Handler. Select the SYSEXDDM folder, as this is where the DDM files are located.



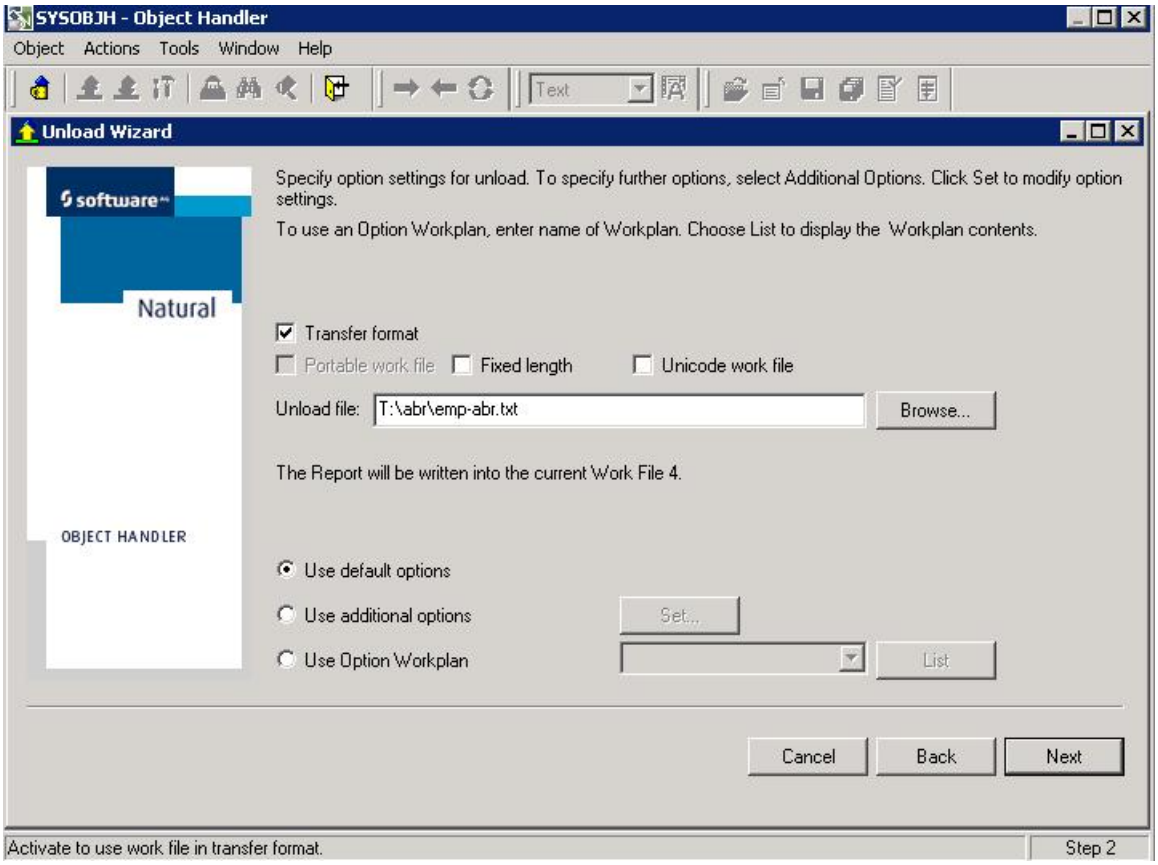
2. You will then select "unload" to unload the DDM's to an extract file.



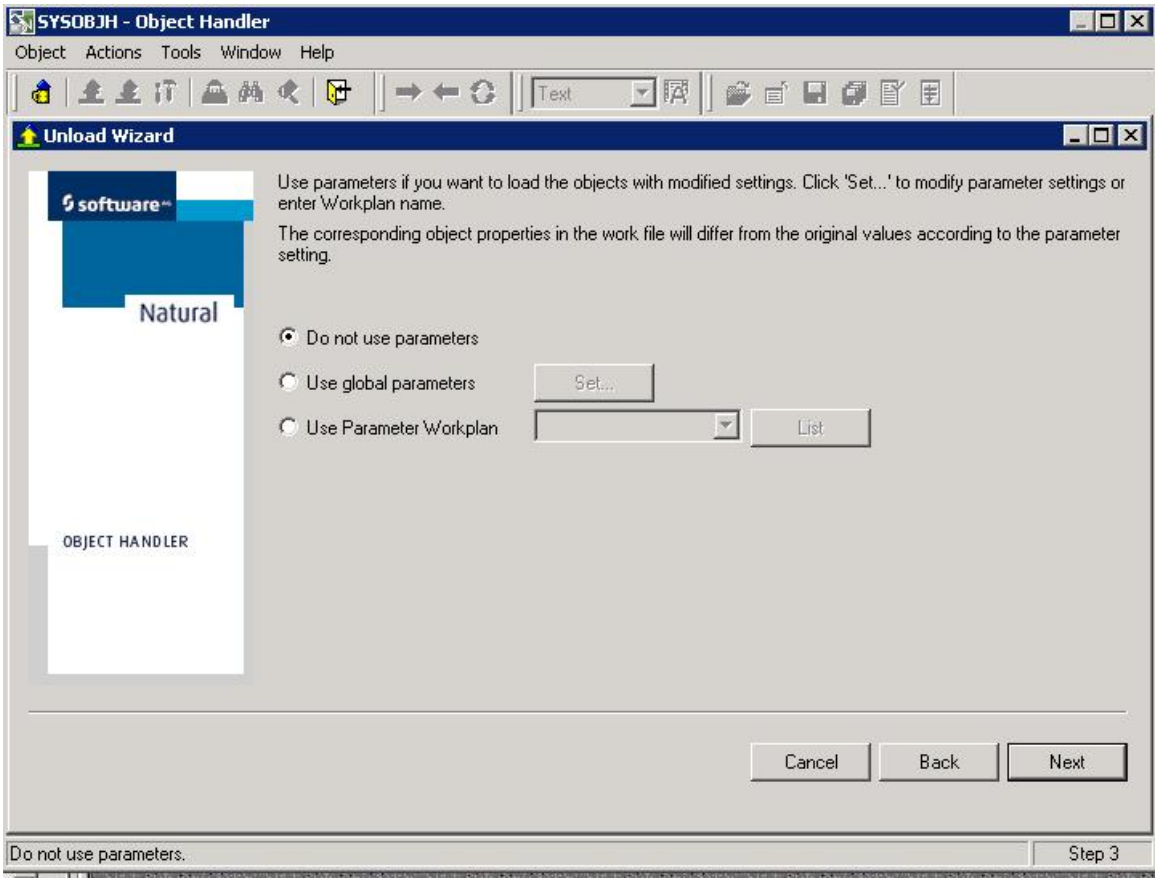
3. Click the **Unload objects into Natural work file(s)** radio button. Click **Next.**



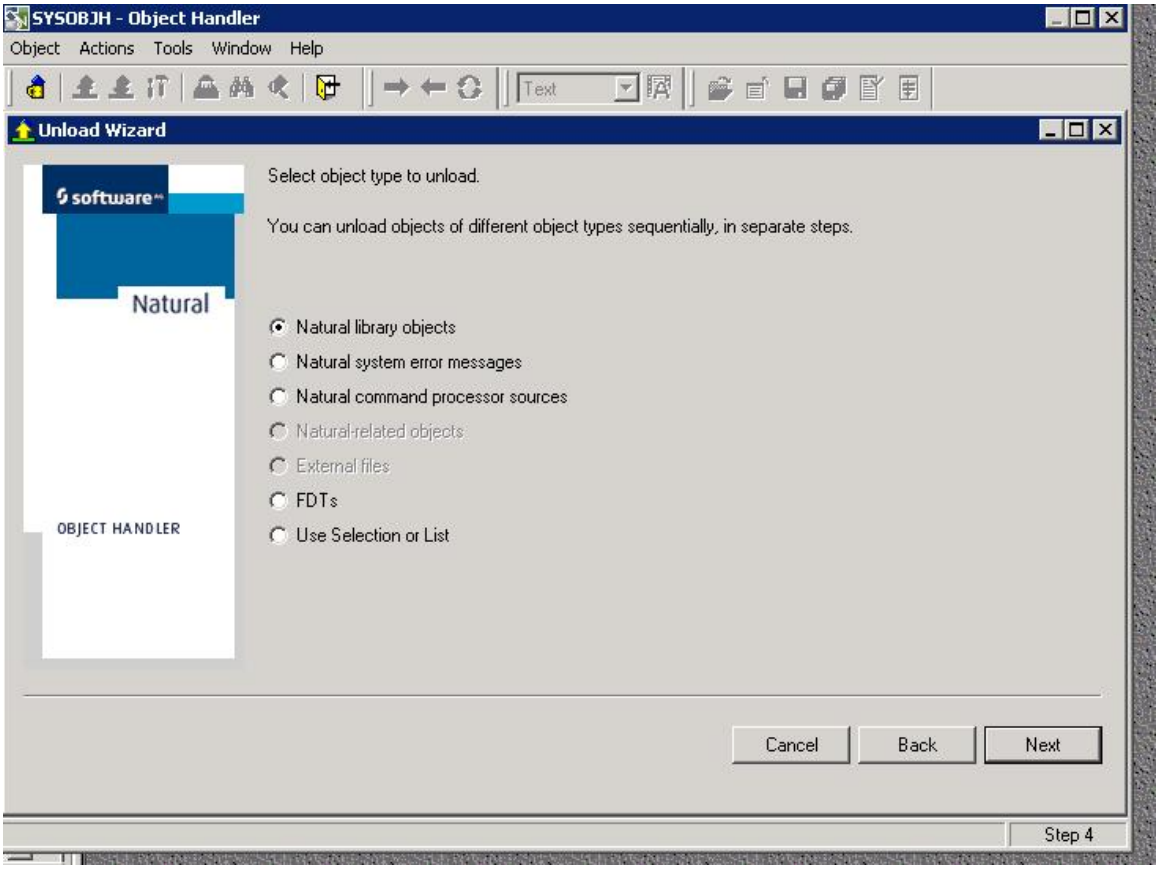
4. In the next window of the **Unload Wizard,** select the **Transfer format** check box, and then enter the location to which you want the files to be unloaded in the **Unload file** text box. If necessary, you can use the **Browse** button.
5. Select the **Use default options** radio button, and then click **Next.**



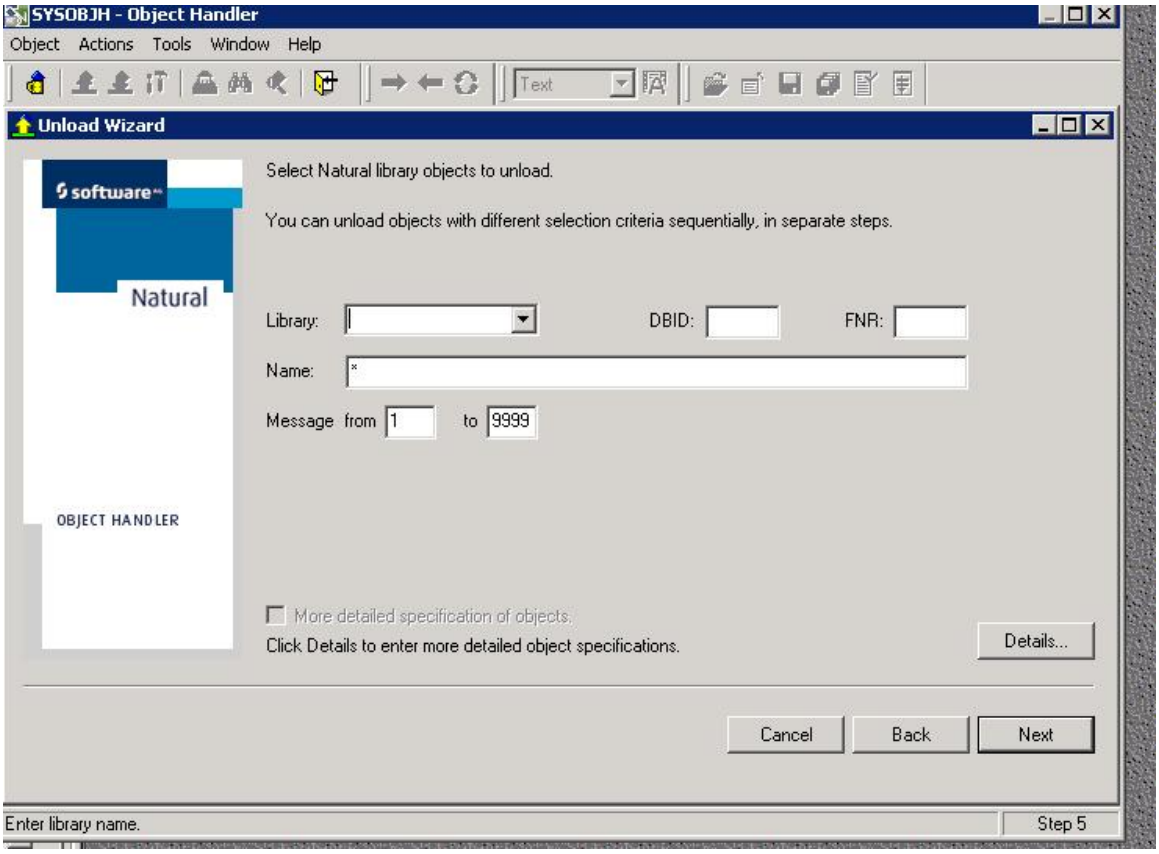
6. In the next window of the Unload Wizard, select the **Do not use parameters** radio button, and then click **Next.**



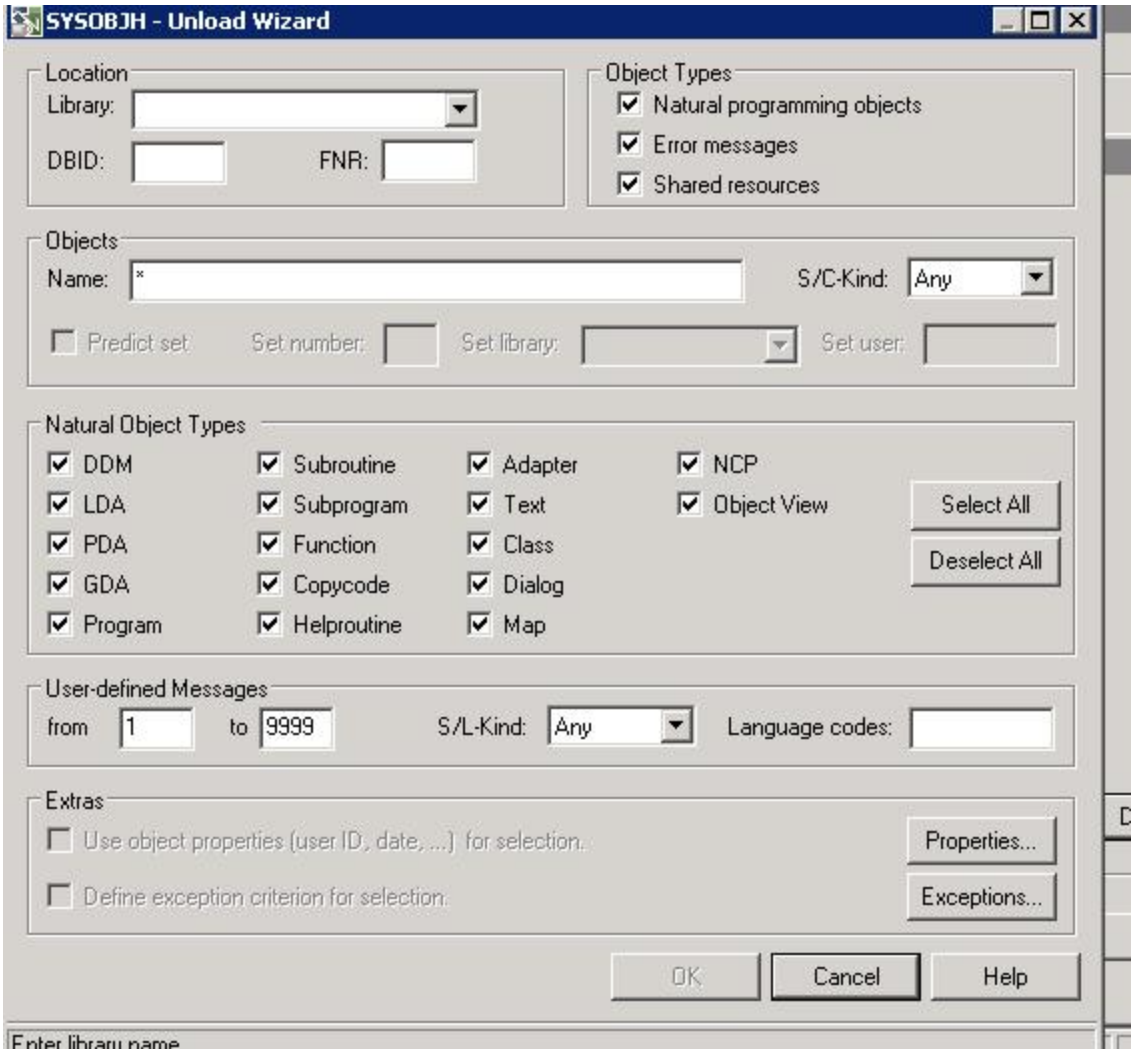
7. In the next window of the **Unload Wizard,** select the **Natural library object** radio button, and then click **Next.**



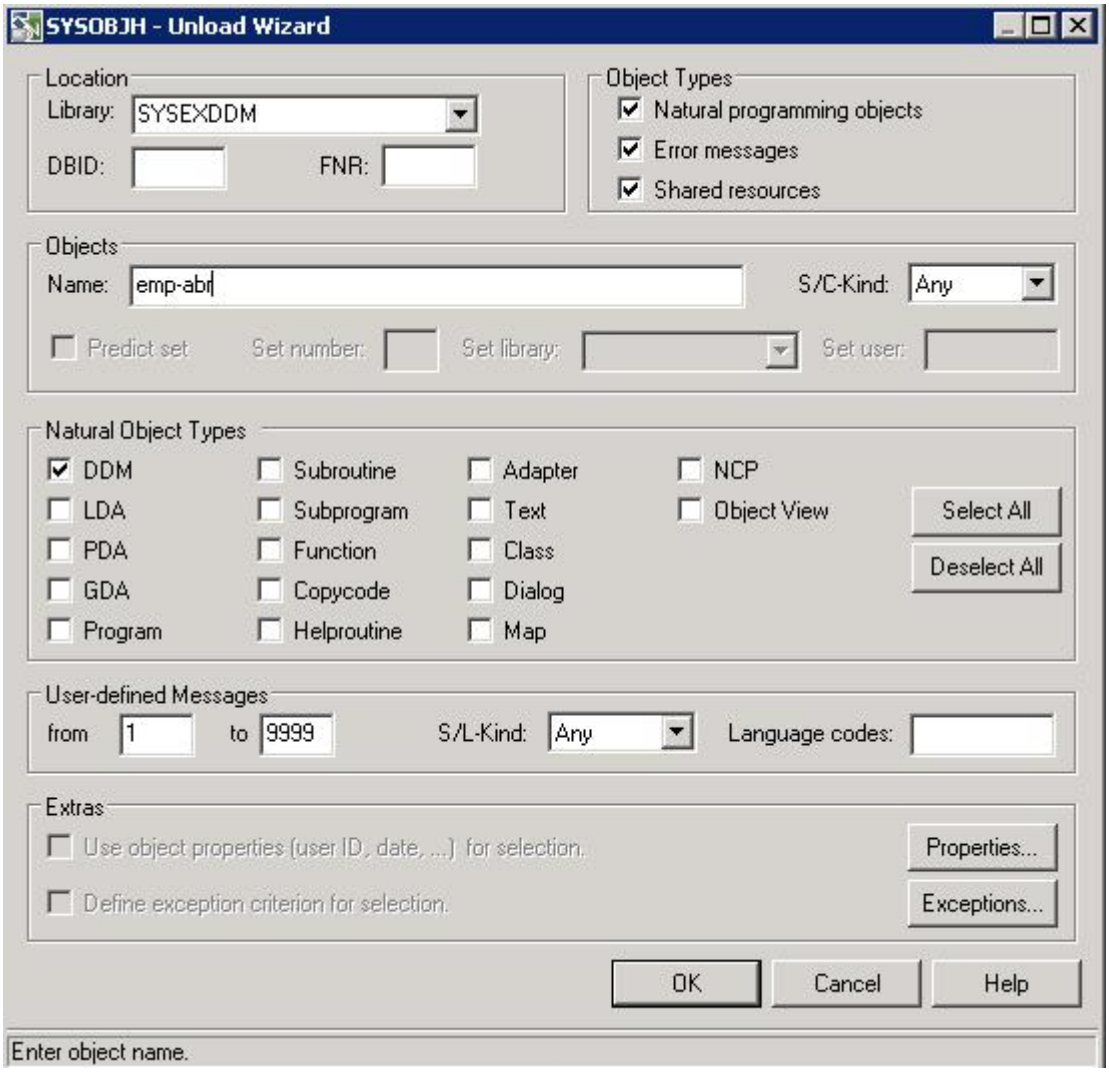
8. In the next window of the **Unload Wizard,** select the library name that contains your DDM definitions, for example, **SYSEXDDM,** from the **Library** list box, and then click the **Details** button. You may also enter a specific DBID and/or a specific FNR. If you leave these blank, all valid DDMs are returned.



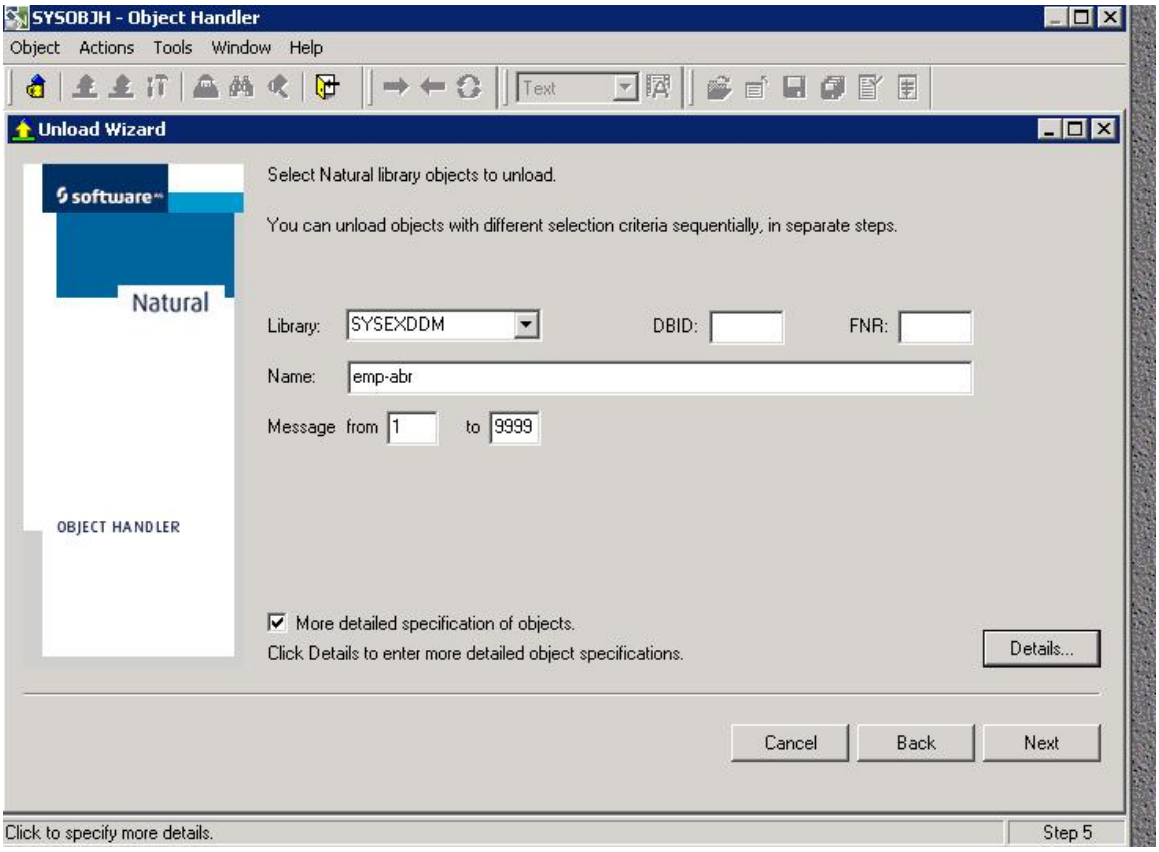
9. In the **SYSOBJH Unload Wizard** window**,** use the **Deselect All** button to deselect all of the choices listed under **Natural Object Types,** and then select only **DDM.** Click the **OK** button to return to the Unload Wizard.



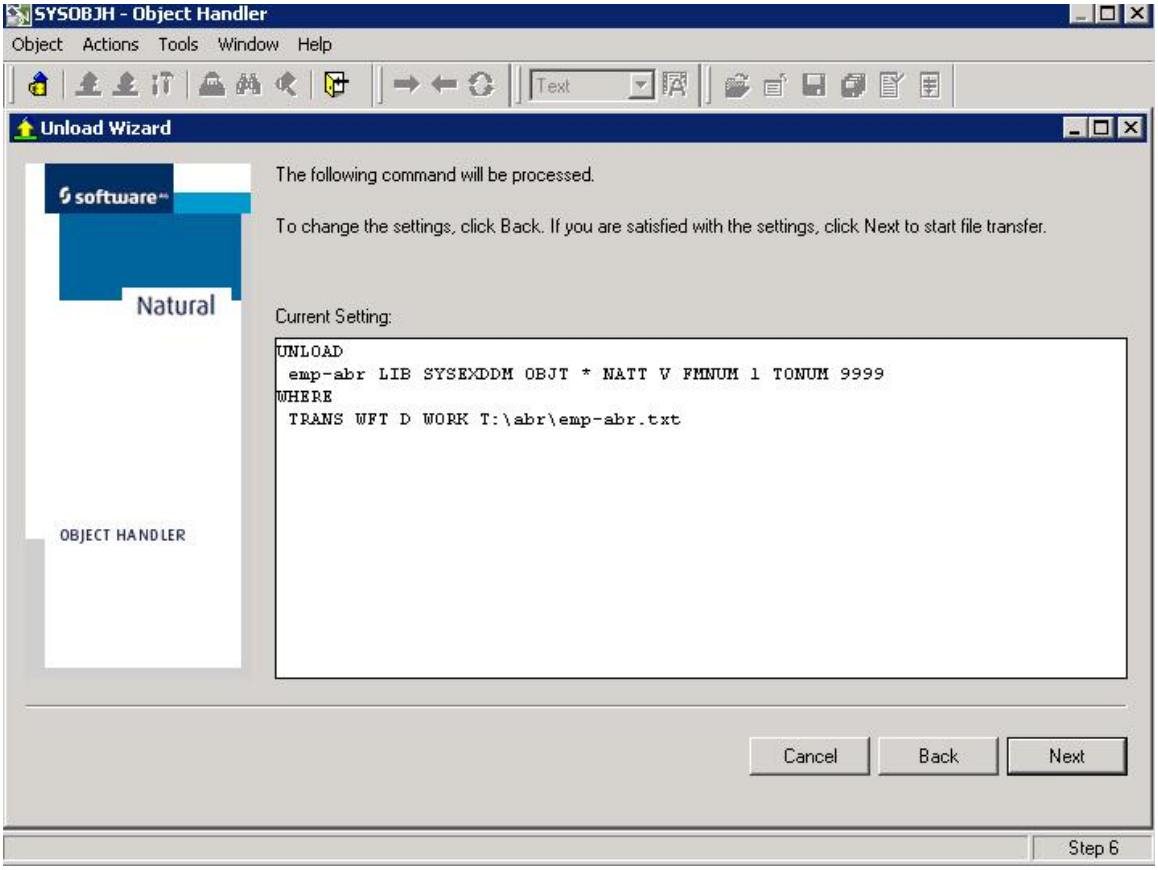
10. **Enter an object name (DDM) that you wish to create an extract for.** Click OK.



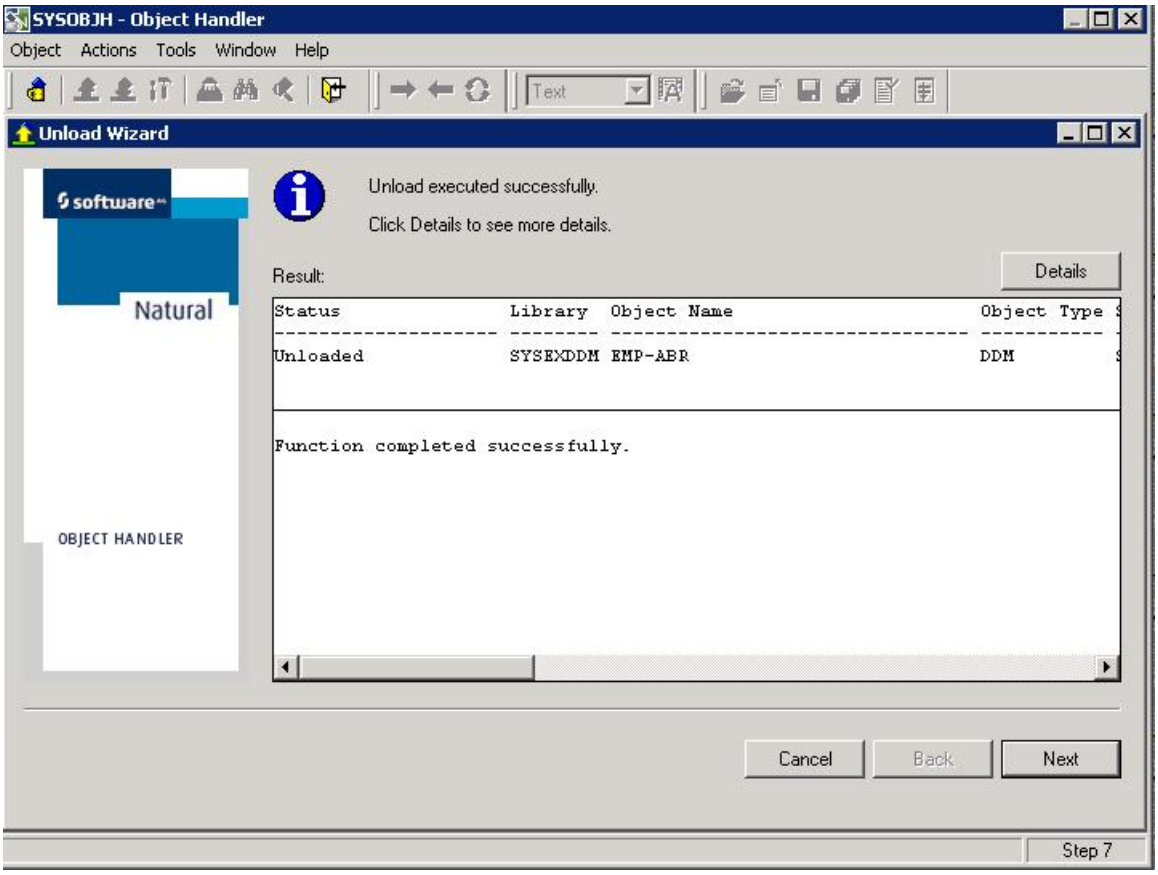
11. Click **Next**.



12. Unload information appears in the next window of the **Unload Wizard.** Click **Next**.



13. A message appears stating the status of the unload. If it is successful, a file is created on your system under the path and file name specified. Click the **Next** button to complete the unload, and then close the application.


SYSOBJH - Object Handler
Object Actions Tools Window Help
Text
Unload Wizard
software
Natural
OBJECT HANDLER
Unload executed successfully.
Click Details to see more details.
Result:
Details
Status Library Object Name Object Type
Unloaded SYSEXDDM EMP-ABR DDM
Function completed successfully.
Cancel
Back
Next
Step 7

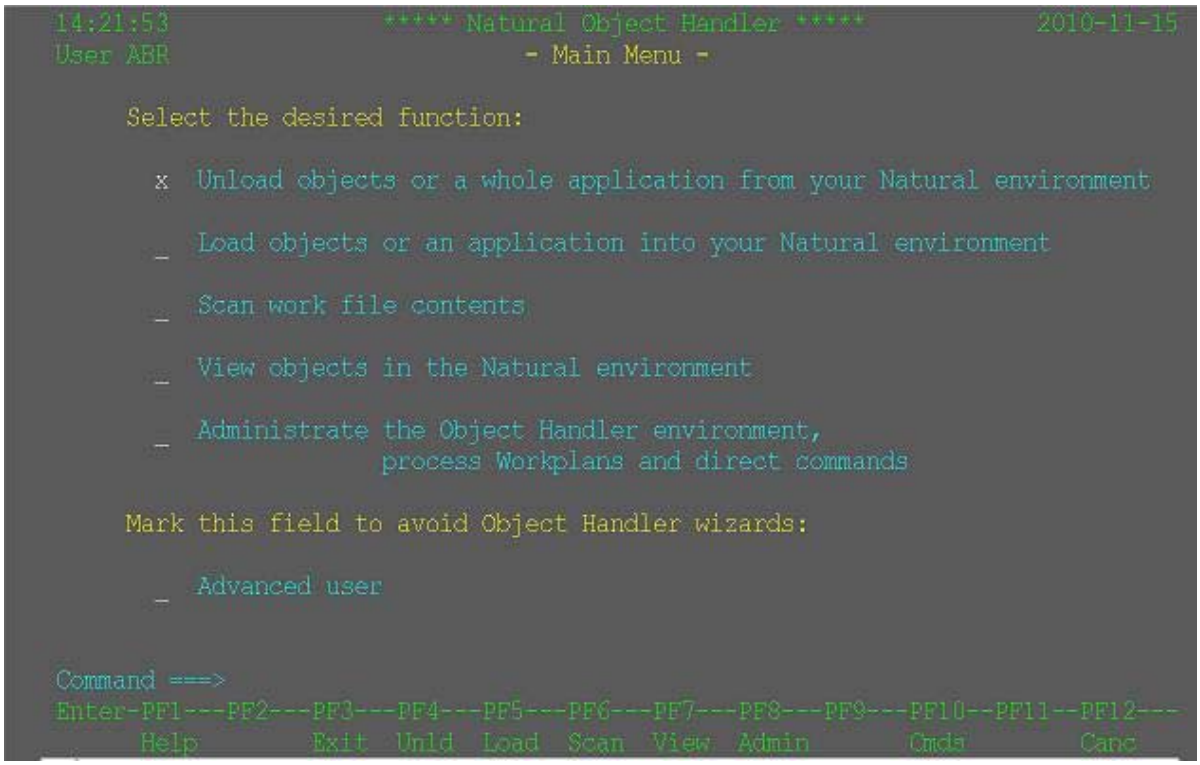
Mainframe

To generate a Mainframe SYSOBJ extract file

Before continuing with import procedures for CONNX for Adabas, it is necessary to first import the Natural DDM files, using the Natural Object Handler.

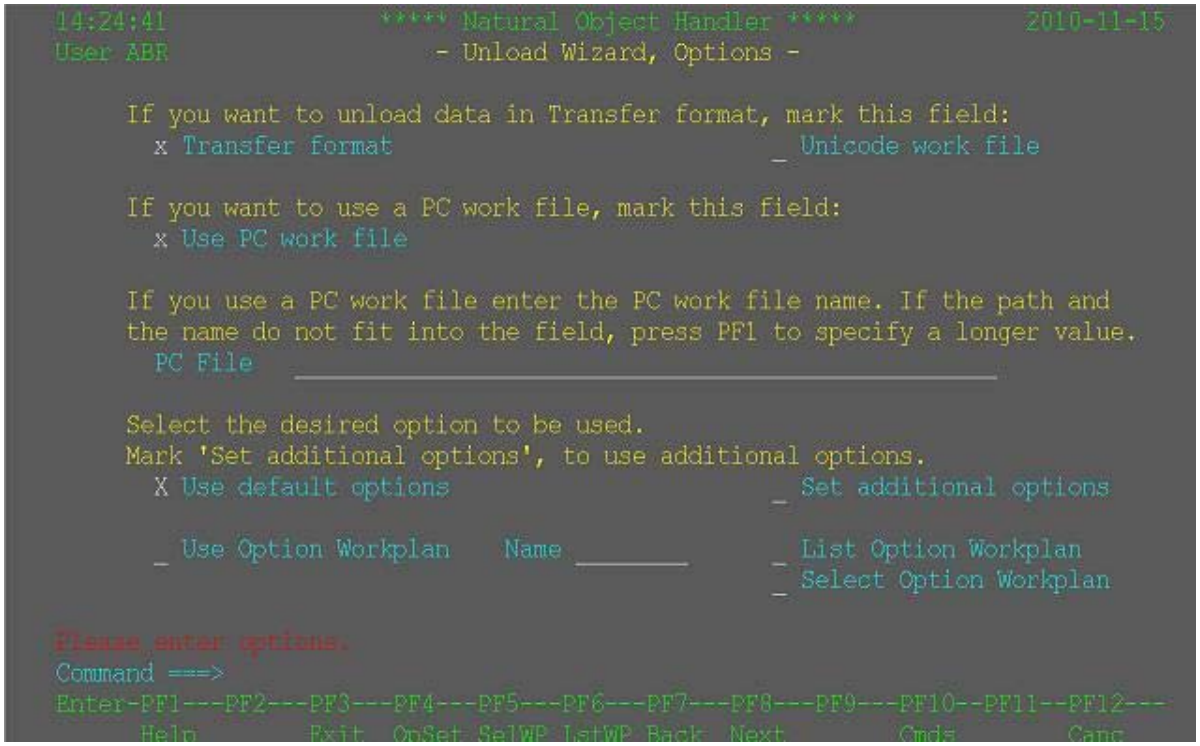
The following steps highlight how to obtain the extract file:

1. In Natural, select the Natural Object Handler tool, and select the option to Unload objects from your Natural Environment.

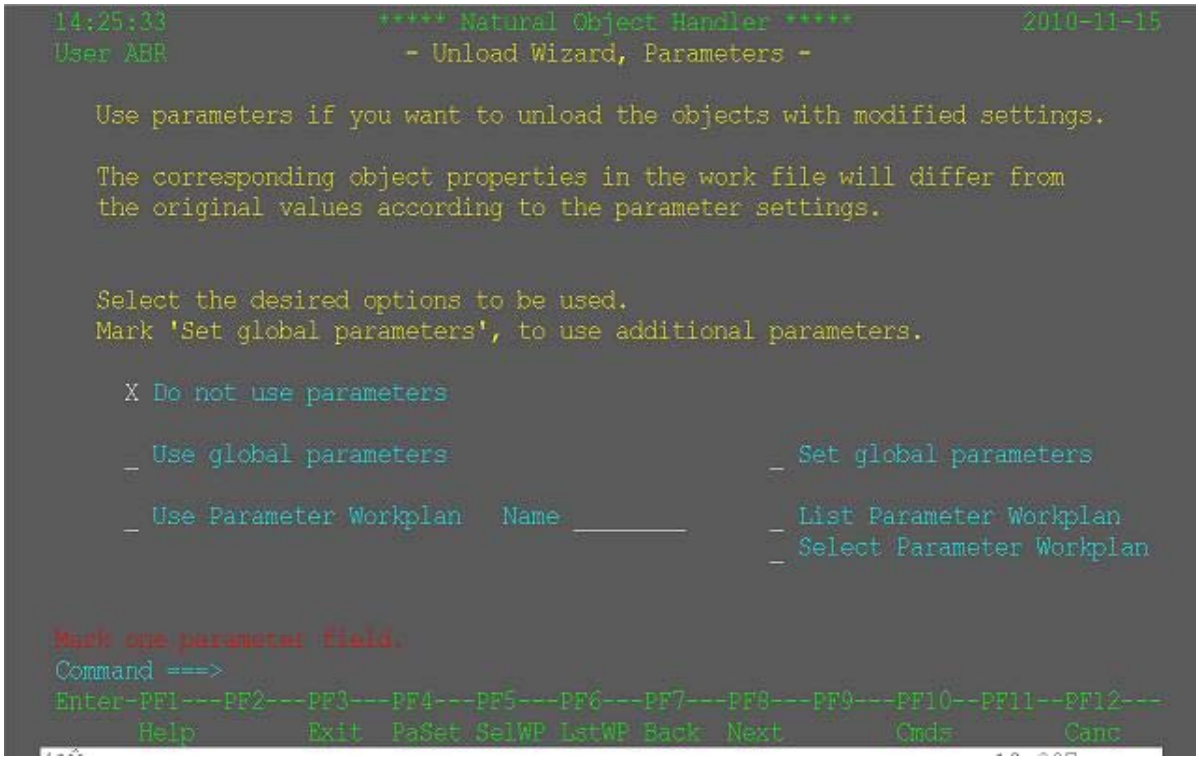


2. The Unload Wizard is then displayed. You must then select the option to unload in your Natural Work Files.

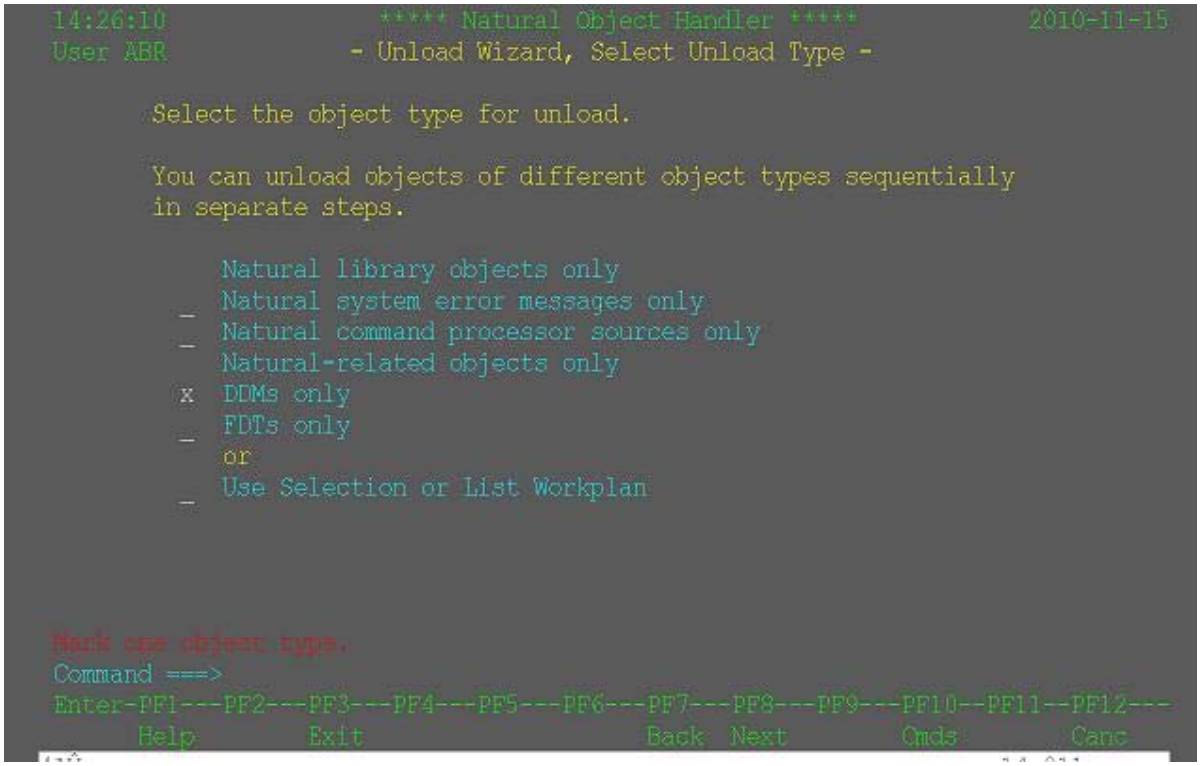
3. You will then be prompted to select Transfer Format, and PC work file. This will enable file to be saved to a PC. The PC option is only valid if Entire Collection is installed.



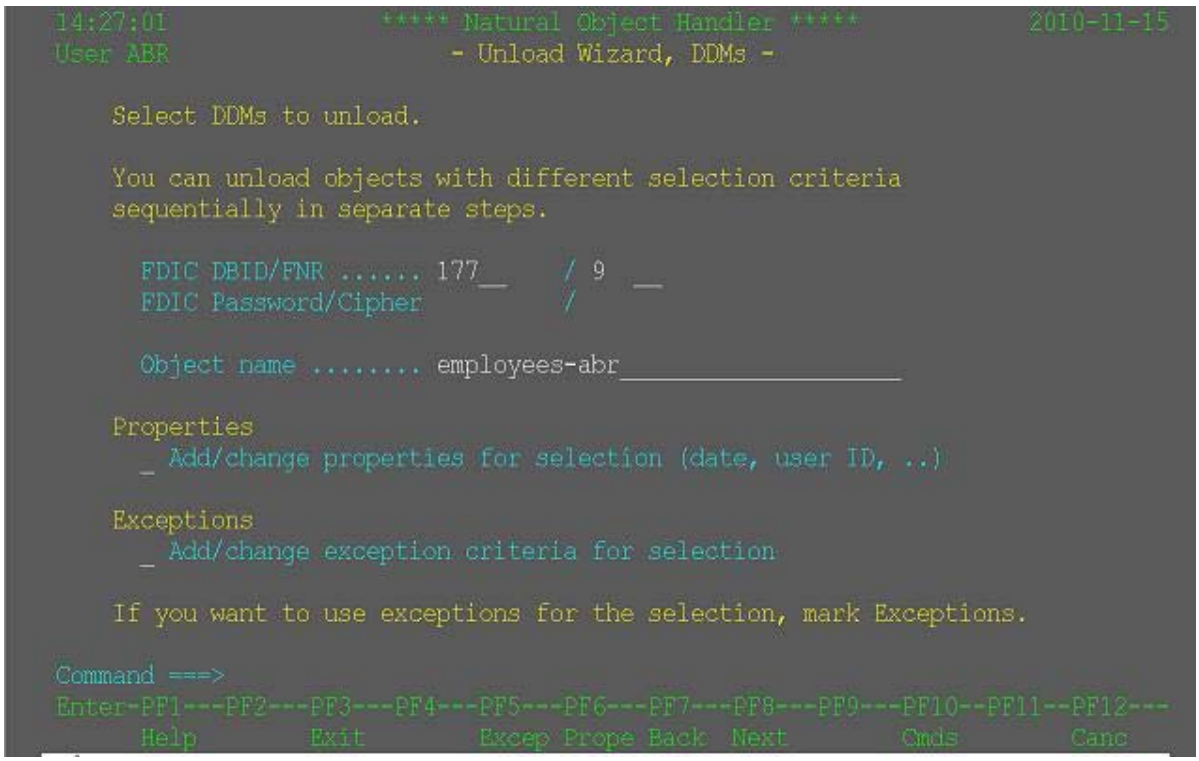
4. Select "Do not use Parameters"



5. You will then unload the DDM's.

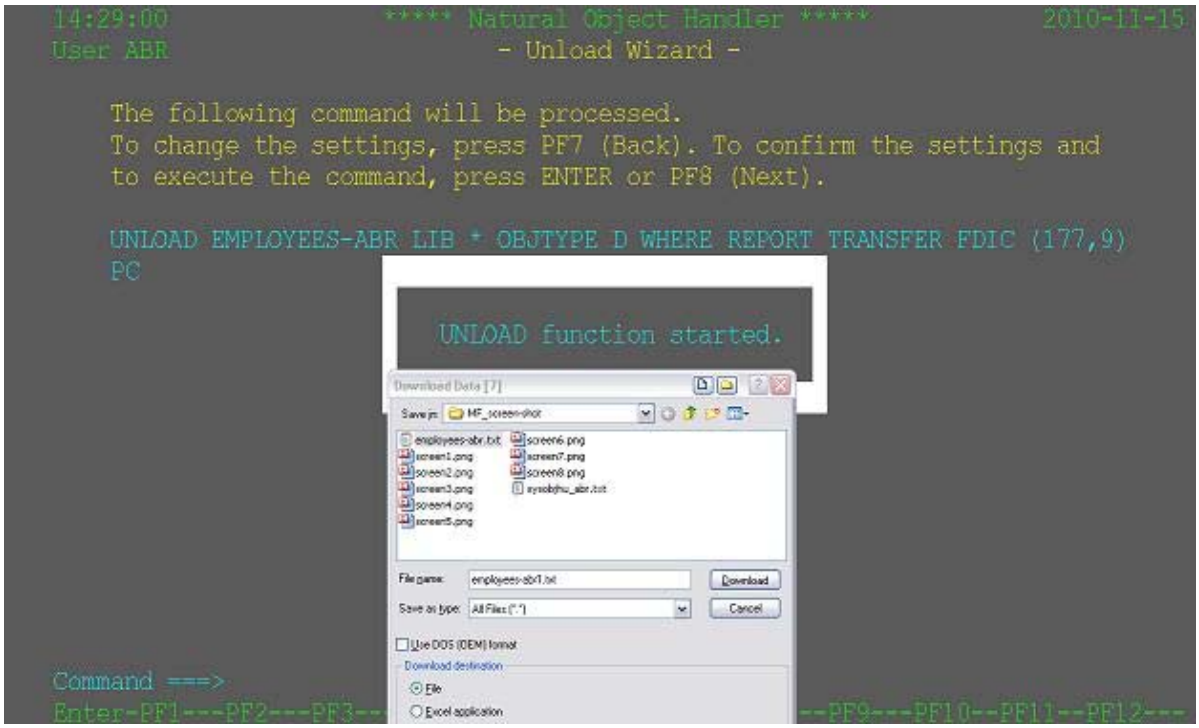


6. You will then select the specific DDM to unload.



7. The Utility will then display the SYSOBJ command it will process. You will select Enter to proceed.

8. After selecting Enter, it will ask where you wish to store the extract file on the PC. This extract file is the input file used to import Natural DDM's into CONNX.

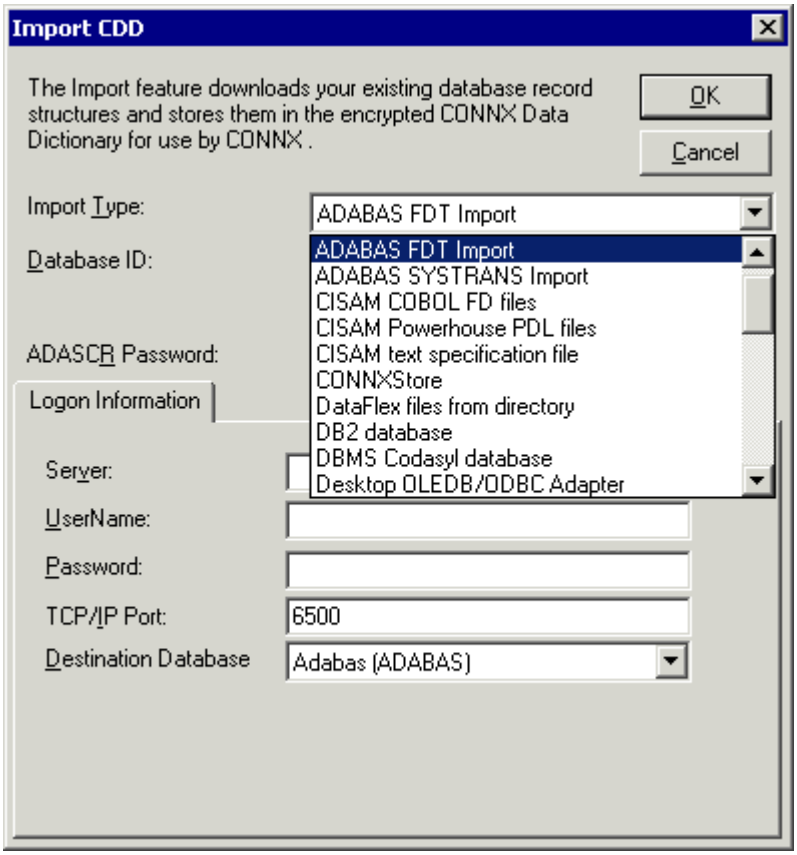


9. After selecting Enter, the extract will be processed, and a completion message will be displayed.

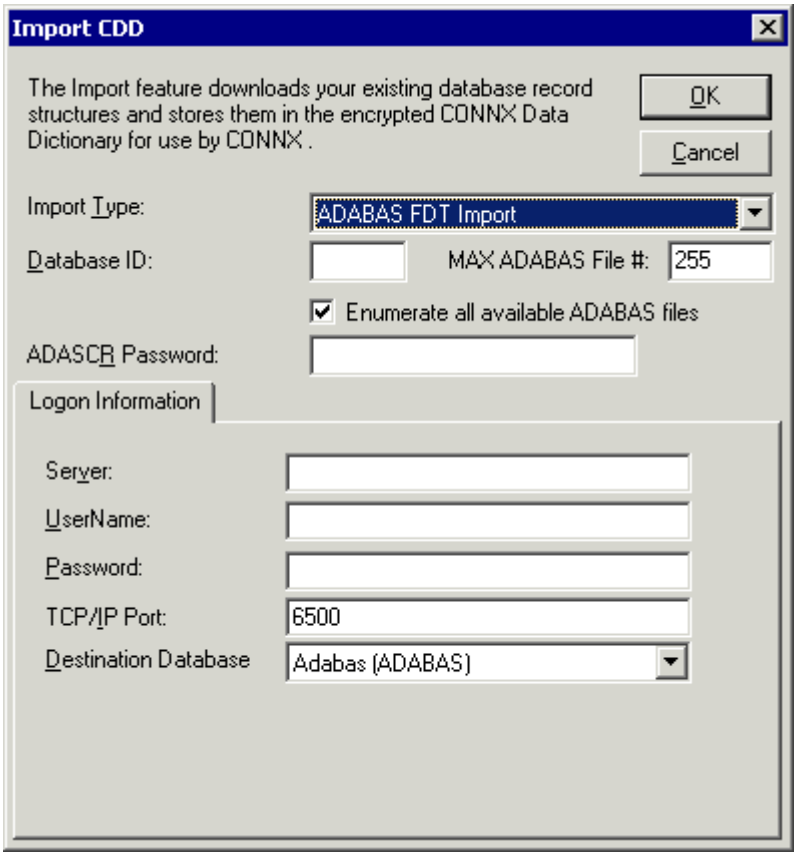
Import from Adabas Files

To import Adabas FDT (read field definitions)

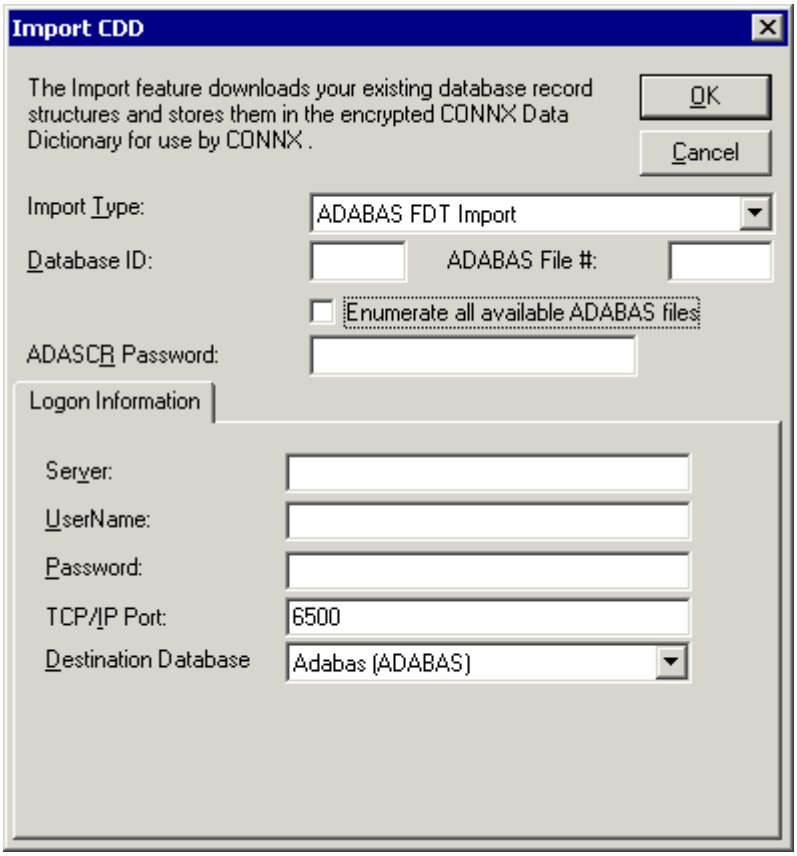
1. Start your Adabas database.
2. Click the **Import** button in the CONNX Data Dictionary Manager window.
3. The Import CDD dialog box appears.
4. Select **Adabas FDT Import** in the **Import Type** list box.



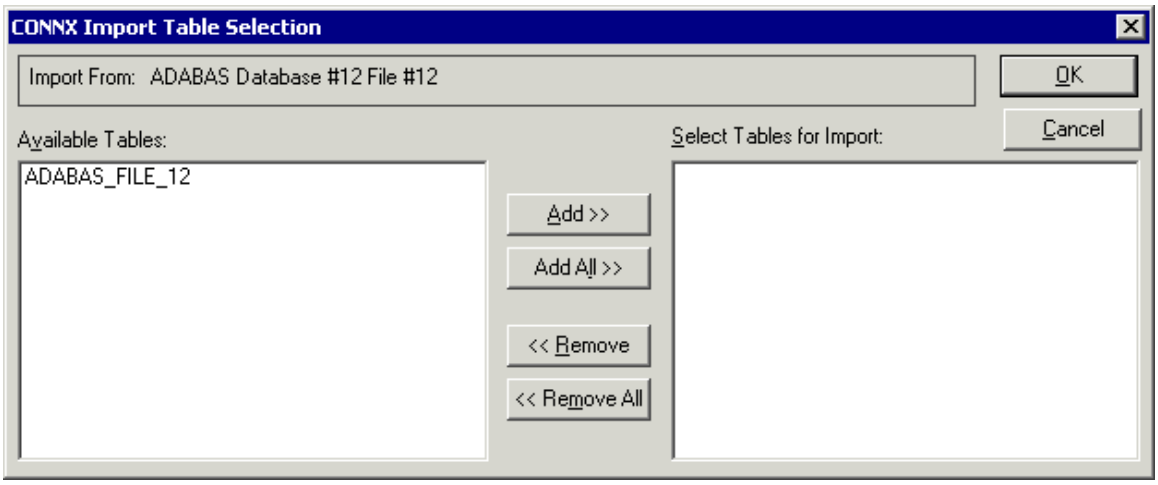
5. To import multiple files, enter a **Database ID** number. The **Enumerate all available Adabas files** check box should be checked. Enter a maximum Adabas file number in the **MAX ADABAS File #** text box.



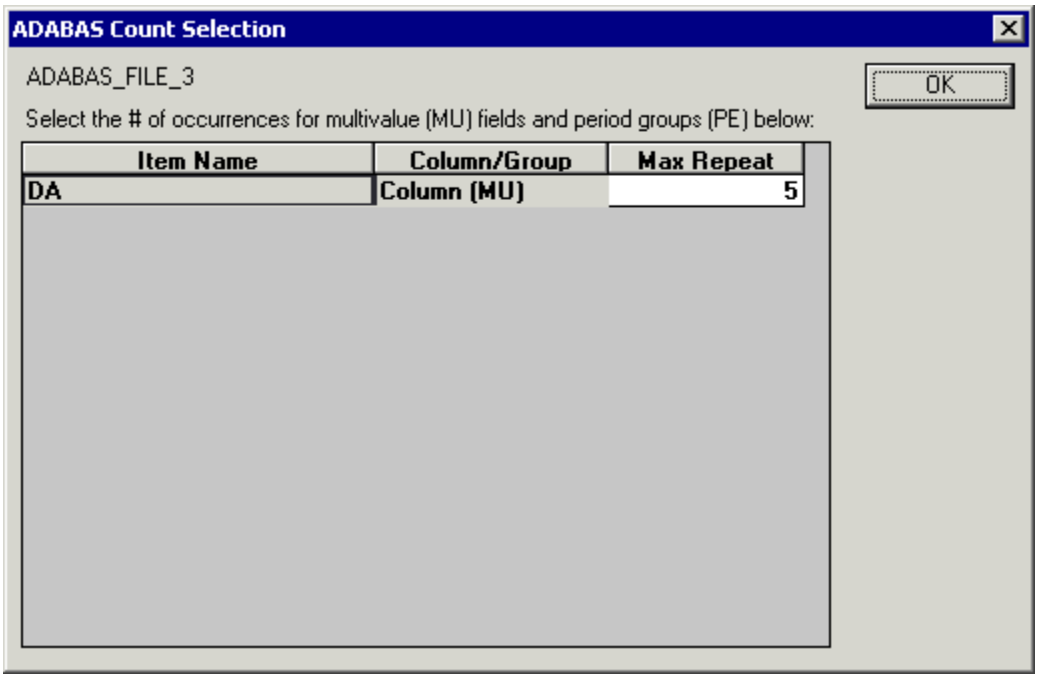
6. To import a single file, uncheck the check box. The label on the **MAX ADABAS File #** text box changes to **Adabas File #.** Enter the file number in the text boxt.



7. Enter an **ADASCR password,** if you are using the ADASCR security method.
8. If the database you are importing from is located on the same Windows machine that you are importing to, enter "localhost" in the **Server** text box. For all other instances, enter the TCP/IP address or server name in the **Server** text box. The CONNX Listener Task attempts to access the given server. If the server is unavailable or cannot be located, the following message appears: "The CONNX Listener process (CNXRUN##_MAIN) is not running on the system."
9. Enter a CONNX user name and password.
10. Click the **OK** button.
11. The **CONNX Import Table Selection** dialog box appears. Select each file to import, and then click the **Add** or **Add All** button.



12. The **Adabas Count Selection** dialog box appears. Enter the number of maximum occurrences of each field under **Max Repeat,** and then click the **OK** button. **Note:** The maximum value that can be entered in the Max Repeat field is 191.



13. Save the CDD by selecting the **File** menu and then clicking **Save.**

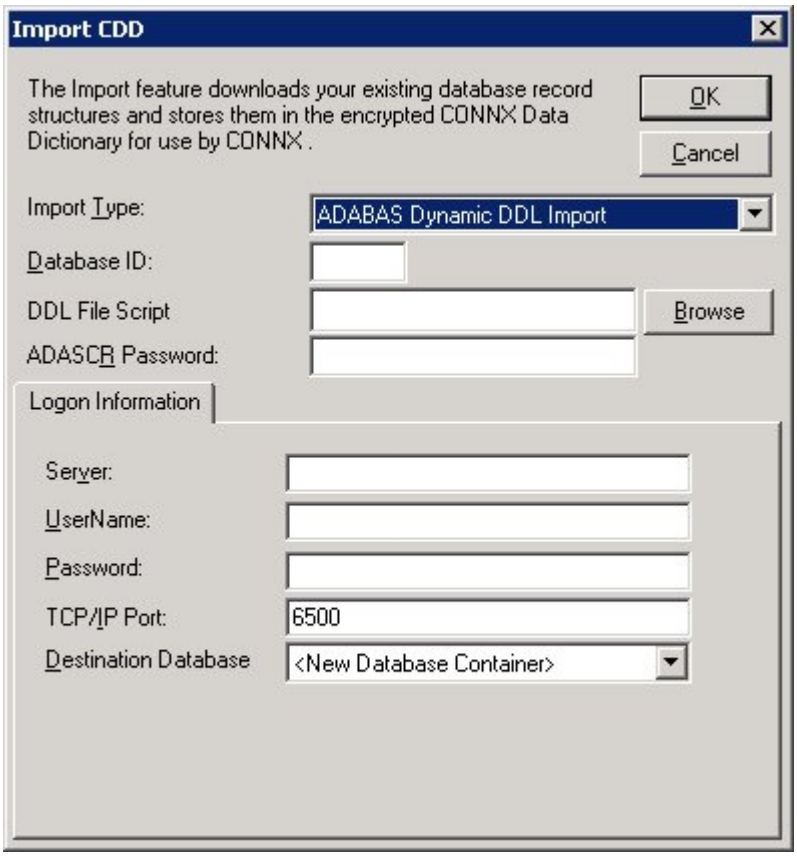
Import from Dynamic DDL

To import from Dynamic DDL files

CONNX provides an option to import metadata from scripts containing two of Software AG's specific SQL commands: CREATE TABLE DESCRIPTION and CREATE CLUSTER DESCRIPTION.

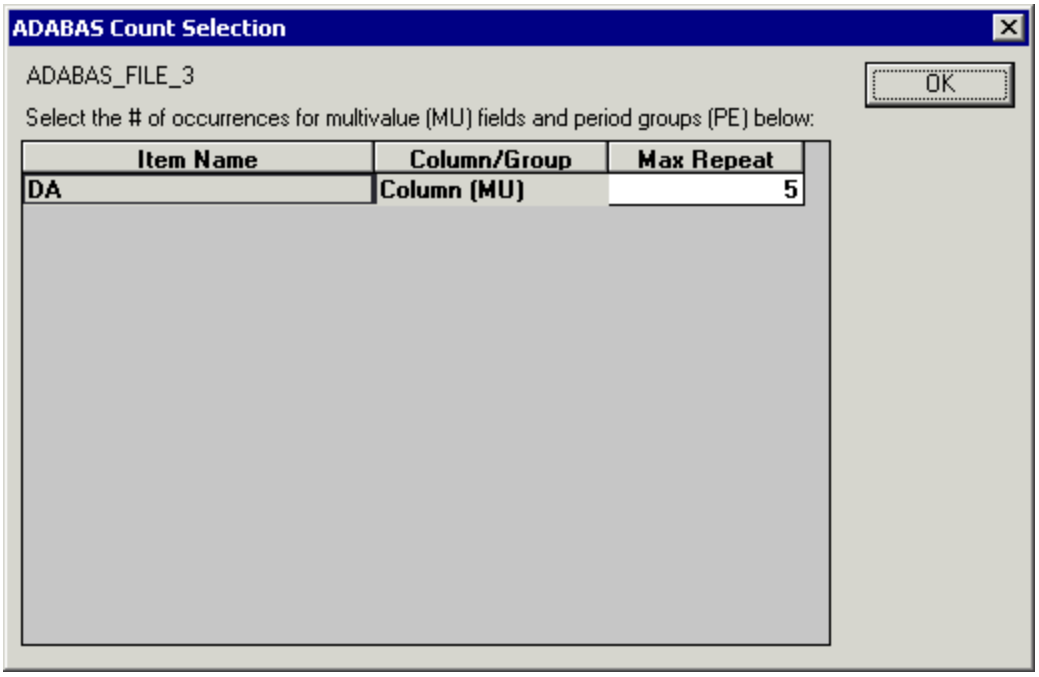
*Note: If you wish to retain hyphens during a **Windows** import, you must first set the KeepHyphens variable.*

1. Start your Adabas database.
2. Click the **Import** button in the CONNX Data Dictionary Manager window.
3. The Import CDD dialog box appears.



4. Select **Adabas Script DDL Import** files in the **Import Type** list box.
5. To import multiple files, enter a **Database ID** number.
6. Add the **DDL File Script** number, or use the **Browse** button to locate the file.
7. Enter an **ADASCR password,** if you are using the ADASCR security method.
8. If the database you are importing from is located on the same Windows machine that you are importing to, enter "localhost" in the **Server** text box. For all other instances, enter the TCP/IP address or server name in the **Server** text box. The CONNX Listener Task attempts to access the given server. If the server is unavailable or cannot be located, the following message appears: "The CONNX Listener process (CNXRUN##_MAIN) is not running on the system."
9. Enter a CONNX user name and password.
10. Click the **OK** button.

11. CONNX searches for Master tables containing the various subtables. The **CONNX Import Table Selection** dialog box appears. Select a table, and then click the **Add** button.
12. The **Adabas Count Selection** dialog box appears. Enter the number of maximum occurrences of each field under **Max Repeat,** and then click the **OK** button.



13. Save the CDD by selecting the **File** menu and then clicking **Save.** The imported tables appear in the CONNX Data Dictionary Manager window.

A script from the sample Employees file appears as follows. CONNX creates the various subtables as well as the relationships between the files.:

```
#########################################################################

#  ASG Version    : x.x.x.x

#  Date/Time      : 04/01/04 10:50:38

#  File           : 12:11

#########################################################################

create cluster description CLUSTER_11

database number 12

file number 11

(
```

```
 create table description EMPLOYEES
(
    COL_SEQNO_0                         seqno(0)  not null
   ,PERSONNEL_ID
      shortname 'AA'
   ,FIRST_NAME
      shortname 'AC'
   ,NAME
      shortname 'AE'
   ,MIDDLE_I
      shortname 'AD'
   ,MAR_STAT
      shortname 'AF'
   ,SEX
      shortname 'AG'
   ,BIRTH
      numeric (6, 0)
      shortname 'AH'
   ,CITY
      shortname 'AJ'
   ,ZIP
      shortname 'AK'
   ,COUNTRY
      shortname 'AL'
   ,AREA_CODE
```

```
            shortname 'AN'
         ,PHONE
            shortname 'AM'
         ,DEPT
            shortname 'AO'
         ,JOB_TITLE
            shortname 'AP'
         ,LEAVE_DUE
            shortname 'AU'
         ,LEAVE_TAKEN
            shortname 'AV'
         ,primary key ( COL_SEQNO_0)
   # Number of columns for this table:   17.
      )

      ,create table description EMPLOYEES_ADDRESS_LINE
      (

          COL_SEQNO_0                      seqno(0)  not null
         ,COL_SEQNO_1                      seqno(1)  not null
         ,ADDRESS_LINE
            shortname 'AI'
         ,foreign key ( COL_SEQNO_0) references EMPLOYEES
         ,primary key ( COL_SEQNO_0, COL_SEQNO_1)
   # Number of columns for this table:    3.
      )
```

```
   ,create table description EMPLOYEES_INCOME
   (

       COL_SEQNO_0                      seqno(0)  not null
      ,COL_SEQNO_1                      seqno(1)  not null
      ,CURR_CODE
         shortname 'AR'
      ,SALARY
         shortname 'AS'
      ,foreign key ( COL_SEQNO_0) references EMPLOYEES
      ,primary key ( COL_SEQNO_0, COL_SEQNO_1)
# Number of columns for this table:    4.
   )

   ,create table description EMPLOYEES_BONUS
   (

       COL_SEQNO_0                      seqno(0)  not null
      ,COL_SEQNO_1                      seqno(1)  not null
      ,COL_SEQNO_2                      seqno(2)  not null
      ,BONUS
         shortname 'AT'
      ,foreign key
         (COL_SEQNO_0, COL_SEQNO_1)
           references EMPLOYEES_INCOME
      ,primary key ( COL_SEQNO_0, COL_SEQNO_1, COL_SEQNO_2)
# Number of columns for this table:    4.
```

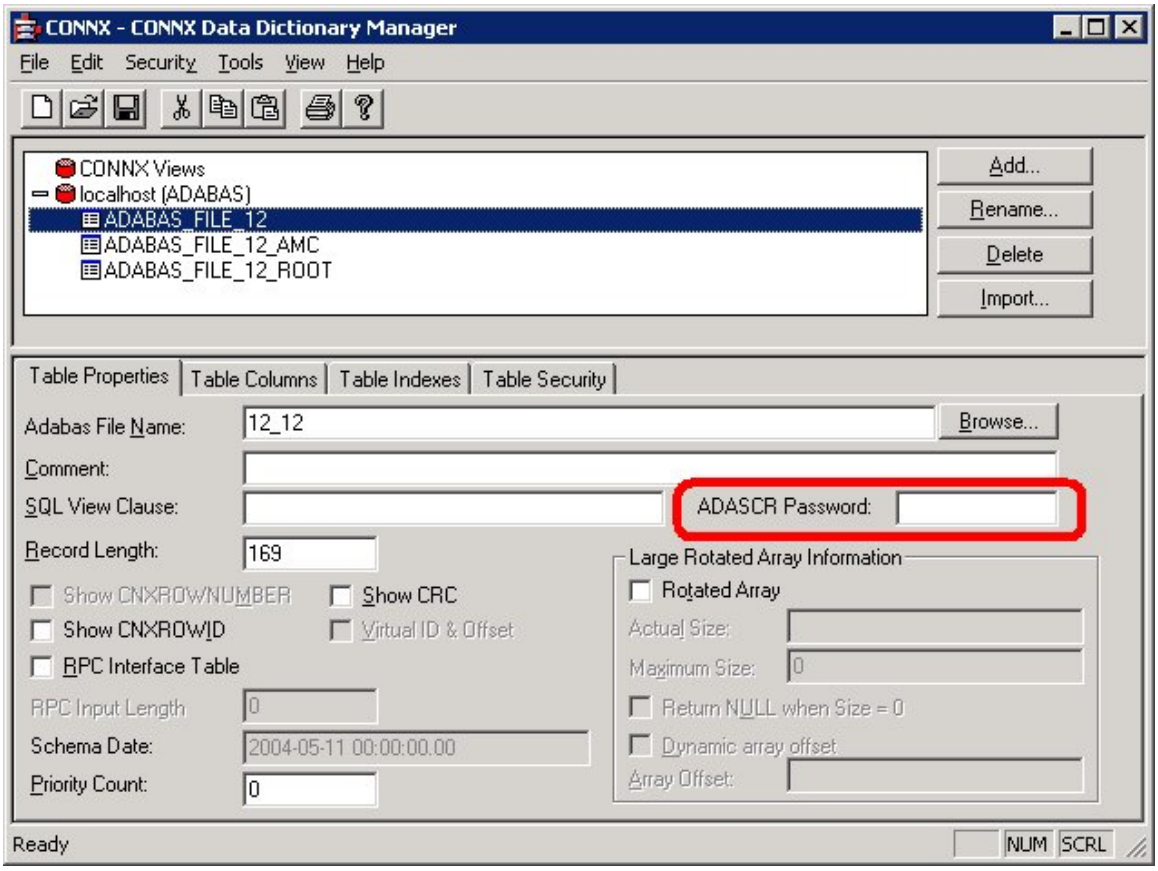
```
   )

   ,create table description EMPLOYEES_LEAVE_BOOKED

   (

       COL_SEQNO_0                     seqno(0)  not null

      ,COL_SEQNO_1                     seqno(1)  not null

      ,LEAVE_START

         shortname 'AX'

      ,LEAVE_END

         shortname 'AY'

      ,foreign key ( COL_SEQNO_0) references EMPLOYEES

      ,primary key ( COL_SEQNO_0, COL_SEQNO_1)

# Number of columns for this table:    4.

   )

   ,create table description EMPLOYEES_LANG

   (

       COL_SEQNO_0                     seqno(0)  not null

      ,COL_SEQNO_1                     seqno(1)  not null

      ,LANG

         shortname 'AZ'

      ,foreign key ( COL_SEQNO_0) references EMPLOYEES

      ,primary key ( COL_SEQNO_0, COL_SEQNO_1)

# Number of columns for this table:    3.

   )

);
```

Adabas ADASCR Security

Using ADASCR Security with CONNX

There are currently two ways to specify an ADASCR password for a given file through CONNX:

- You can specify a global ADASCR password in the CONNX Data Dictionary that can be used by all users. Administrators can then further restrict access to the file using CONNX security on the Table Security tab. For more information on the many CONNX security features, see CONNX Security Overview.



- At runtime, administrators can provide a password for any Adabas file using the following extended CONNX syntax:

  {fn setadapassword <table alias> , <password>}

For example, to specify a password for the CUSTOMERS_ADABAS table, issue the following SQL statement:

```
SELECT * FROM adabas_windows.dbo.CUSTOMERS_ADABAS {fn setadapassword
CUSTOMERS_ADABAS, PASSWORD}
```

If you have defined a global password in a CONNX Data Dictionary, the password specified with the SQL statement overrides the global password.

Code Pages

CONNX supports CODEPAGE and SBCCSID settings. The various code pages are stored in c:\connx32\sbtabs (if the installed database supports code pages). Code pages can be recognized by the R2D extension, where the first four hex numbers of the file represent the server SBCCSID, and the last four hex numbers represent the Client (or Windows) SBCCSID translation. The default code page setting in CONNX is 37 (United States EBCDIC).

The SBCCSID can be changed using the SBCCSID configuration setting.

The following list includes some widely used code page settings:

37- United States - EBCDIC

273 Austria, Germany - EBCDIC

278 Finland, Sweden - EBCDIC

285 United Kingdom - EBCDIC

For a complete set of code pages that run on IBM mainframes, go to:

http://www-03.ibm.com/systems/i/software/globalization/codepages.html

For the Windows client machine, the client-based SBCCSID is often obtained when CONNX performs a Windows call that sets the proper code page. If you wish to override the default value, specify the SBCCSID in the CONNX section of the registry.

You can also set the code page for individual columns of a table by using the CONNX Configuration Manager, and specifying it to a value that is appropriate. The code page is specified in the Table Columns tab for the table in the Code Page column.

If the database is located on a server machine, the code page will be obtained from that system's operating system. Please note: Normally, you should not have to do anything. The code page will be set automatically at import time. If you need to change the code page either by changing the code page on the server or by changing the client SBCCSID value, you may need to re-import all the tables in the CDD if you wish the new code page value to be set in all the tables.

C-ISAM, DISAM, and Micro Focus Imports

CONNX uses a local database definition file (CDD) that contains metadata relating to the various data sources it can access. The metadata is transferred to the CDD file by invoking the CONNX Data Dictionary tool, and specifying one of the following import options:

- **C-ISAM, DISAM, and Micro Focus Text File Imports**
- **C-ISAM, DISAM, and Micro Focus Manual Imports**
- **C-ISAM, DISAM, and Micro Focus COBOL FD**

CONNX supports C-ISAM, DISAM, and Micro Focus databases version 7 and above.

Important: *It should be noted that C-ISAM and DISAM performs all transactions at the Process level and that any actions taken in separate threads are all pooled into the single Process transaction. Consequently, transactions are not recommended in instances where multiple users are connecting through a JDBC server.*

Related Topics

- Creating C-ISAM, DISAM, or Micro Focus Databases Manually
- Creating CDD entries manually
- C-ISAM View Text File Import Specification

C-ISAM, DISAM, and Micro Focus Manual Imports

Creating C-ISAM, DISAM, or Micro Focus Databases Manually

In CONNX, a C-ISAM, DISAM, or Micro Focus database represents a collection of C-ISAM, DISAM, or Micro Focus files on either a Unix server or a Windows system. With the Add Database Connection feature, you can manually create new database connections to VMS servers on a given port. After the database has been created, you can use the CONNX import to add tables to the database, or you can manually create new table entries.

Related Topics

- Creating CDD entries manually
- To build a C-ISAM, DISAM, or Micro Focus database manually
- To create a C-ISAM, DISAM, or Micro Focus table entry manually
- To add columns to a C-ISAM, DISAM, or Micro Focus file entry
- To refresh an index

Creating CDD entries manually

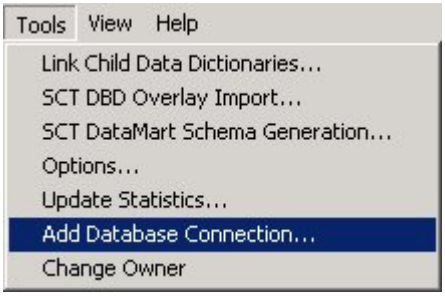
CONNX Data Dictionaries for C-ISAM, DISAM, or Micro Focus tables can be created manually. Once a location is established for the type of database files in the CONNX Data Dictionary Manager dialog box, create new entries by using the Add button. Text boxes in the lower CDD pane define table properties and other features of the table, including levels of security. The data file name must be specified on the Table Properties tab in the CONNX Data Dictionary Manager window.

Related Topics

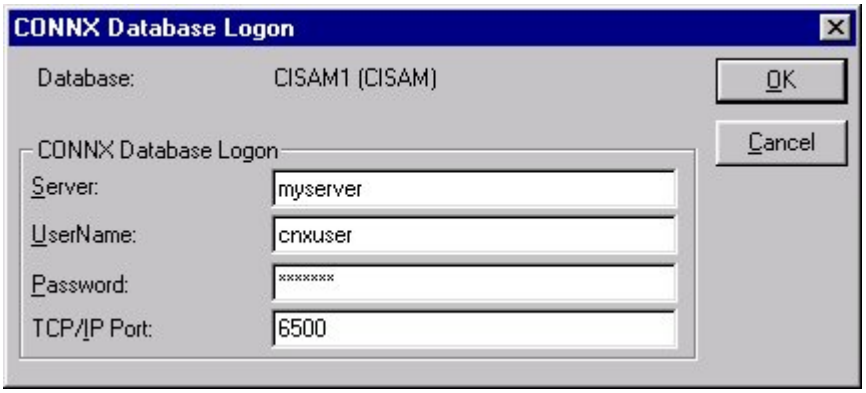
- Creating C-ISAM, DISAM, or Micro Focus Databases Manually
- To build a C-ISAM, DISAM, or Micro Focus database manually
- To create a C-ISAM, DISAM, or Micro Focus table entry manually
- To add columns to a C-ISAM, DISAM, or Micro Focus file entry
- To refresh an index

To build a C-ISAM, DISAM, or Micro Focus database manually

1. Select **Add Database Connection** on the **Tools** menu in the CONNX Data Dictionary Manager window.



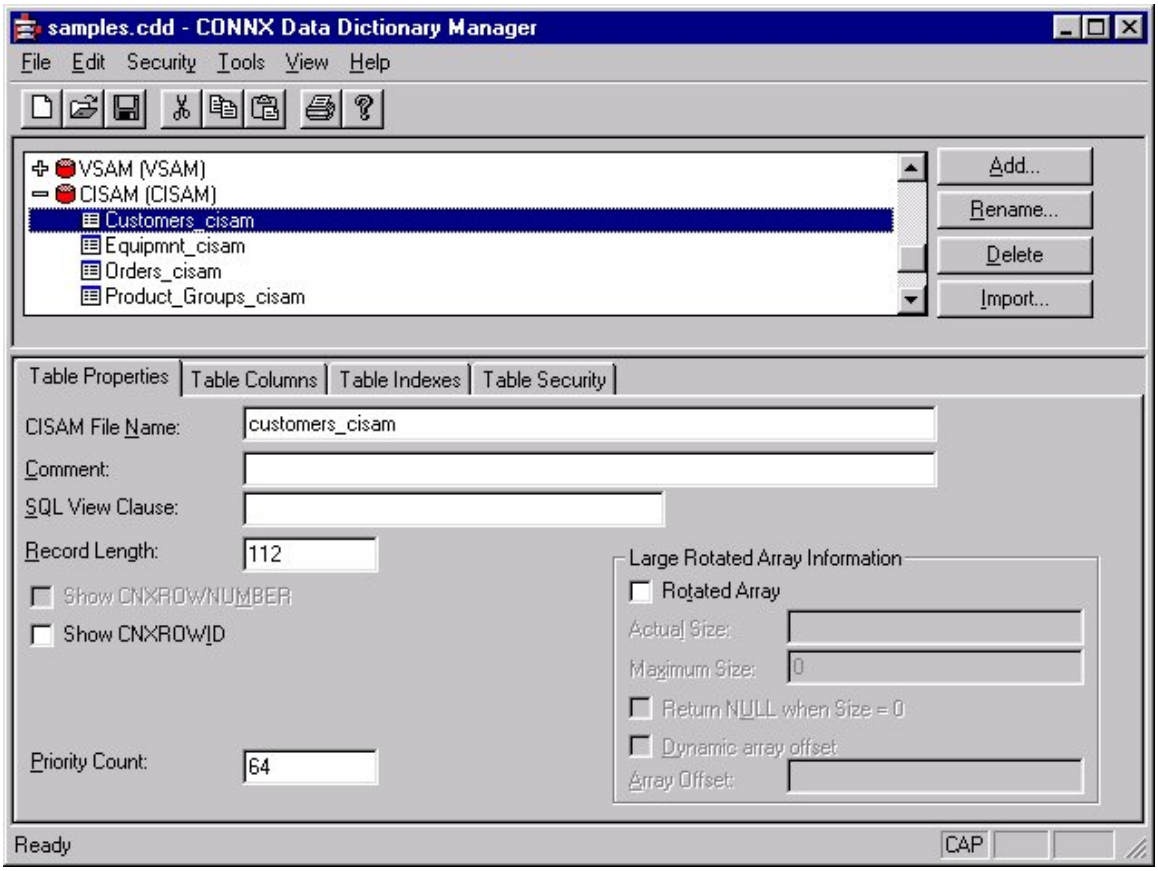
2. The **Enter the logical name of the new database** dialog box appears.
3. Type a name for the database in the **Database Name** text box.
4. Select **C-ISAM, DISAM, or Micro Focus** as the type of database to create in the **Database Type** text box.
5. If the database you are importing from is located on the same Windows machine that you are importing to, enter "localhost" in the **Server** text box. For all other instances, enter the TCP/IP address or server name in the **Server** text box. The CONNX Listener Task attempts to access the given server. If the server is unavailable or cannot be located, the following message appears: "The CONNX Listener process (CNXRUN##_MAIN) is not running on the system."
6. The **CONNX Database Logon** dialog box appears with the server name and TCPIP port number entered as defaults. For Windows systems, enter "localhost" as the server name.



7. Type a C-ISAM, DISAM, or Micro Focus user name in the **User Name** text box.
8. Type a C-ISAM, DISAM, or Micro Focus password in the **Password** text box.
9. Click the **OK** button.
10. The new database is added to the list of available databases in the CONNX Data Dictionary Manager window. Each C-ISAM, DISAM, or Micro Focus database listed in the CONNX Data Dictionary Manager window can be associated with a different server or, for Windows, a local host machine.

To create a C-ISAM, DISAM, or Micro Focus table entry manually

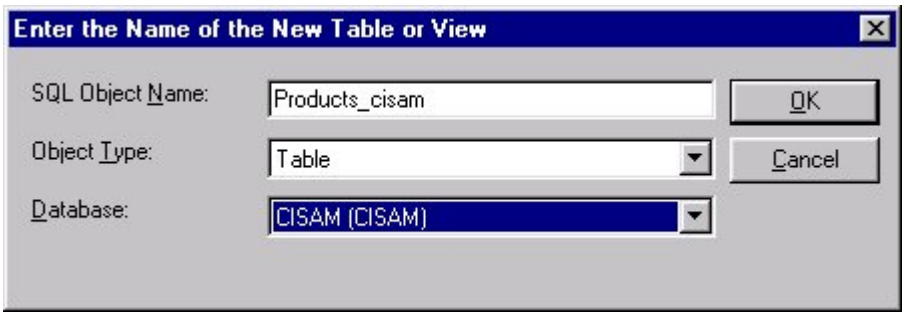
1. Click the **Add** button in the CONNX Data Dictionary Manager window.



2. The **Enter the Name of the New Table or View** dialog box appears. Type the **SQL Object Name.**
    3. Must be unique.
    4. Valid CONNX table names cannot contain spaces or begin with a number.
    5. Maximum length is 50 characters

3. Select **Table** as the type of object to create in the **Object Type** list box. This option determines whether the object is a view or a table definition.

4. Select **C-ISAM, DISAM, or Micro Focus** as the type of database in which to create an object in the **Database** list box. The option specifies the type of database in which the table is located. Valid database types for manual entry are C-ISAM, DISAM, Micro Focus, DataFlex, and VSAM.



5. Click the **OK** button.

6. In the **C-ISAM, DISAM, or Micro Focus File Name** text box in the CONNX Data Dictionary Manager window, type the physical VMS path for the C-ISAM, DISAM, or cro Focus data file. The maximum length of the filename is 255 characters. The C-ISAM, DISAM, or cro Focus file name may also include VAX logicals. If a logical is used in the file name, it is important that the logical is still defined when logging onto the VMS system as a network process.

Note: If you are running CONNX for C-ISAM, DISAM, or cro Focus within a Unix/Linux environment, enter ${<myenvironmentvariable>} in the file path to expand an environment variable.

7. In the **Comment** text box, type up to 64 characters of descriptive text.

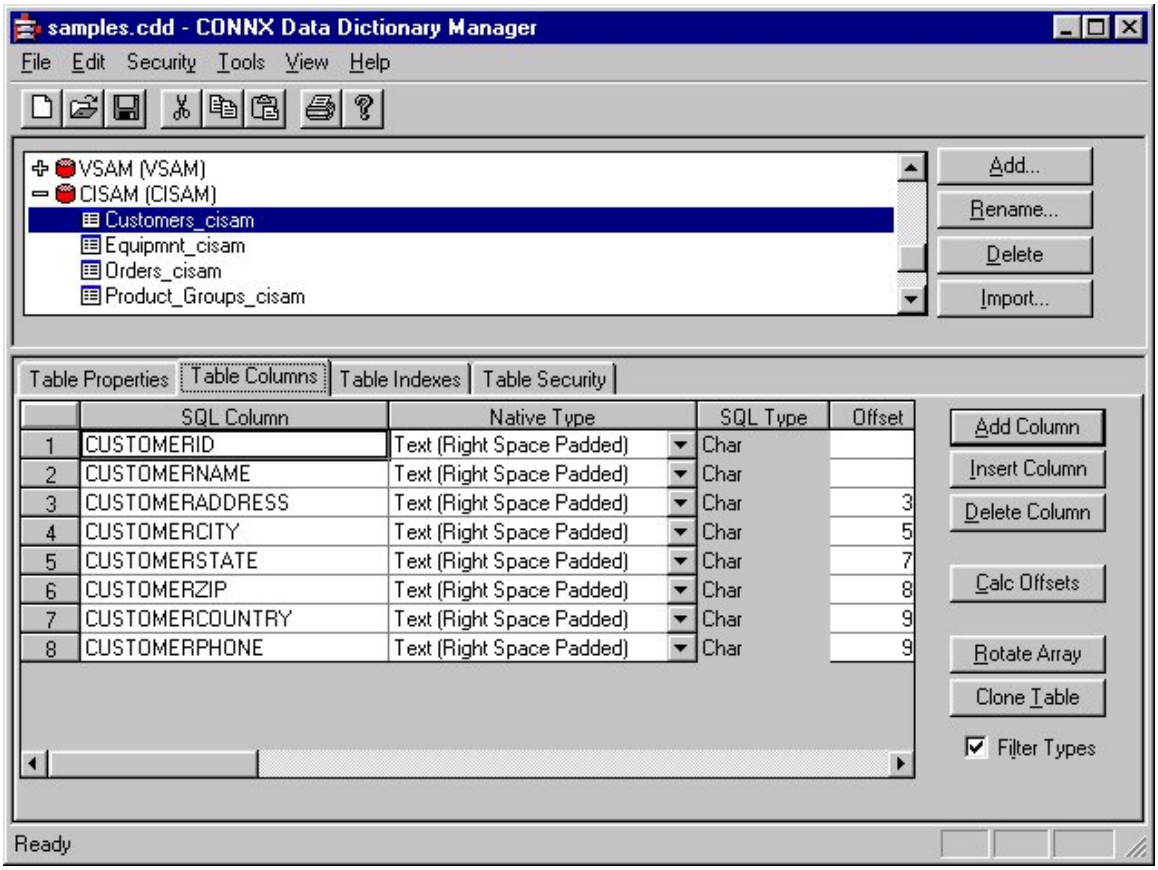
8. In the **SQL View Clause** text box, you may type any valid SQL expression. This text box is used to limit the type of records returned from the table. Maximum length is 128 characters. See SQL View Clause Text Box.

9. Record Length is automatically pulled from the data file.

10. The Priority Count text box displays a comparison of the relative size of the tables used in join optimization. The numbers may not match the actual number of records in the C-ISAM, DISAM, or Micro Focus file. This value is automatically pulled from the data file.

To add columns to a C-ISAM, DISAM, or Micro Focus file entry

1. Select the table to which columns will be added in the CONNX Data Dictionary Manager window.
2. Click the **Table Columns** tab.



3. Click the **Add Column** button. The cursor moves to a new row in the table.
4. Under **SQL Column,** type the name of the new column.
    5. A valid column name must be unique within the table.
    6. The CONNX column name may be different from the SQL column name. Rename columns or use alias column names.
    7. Valid column names cannot contain spaces or begin with numbers.
    8. Maximum length is 30 characters.

Note: *If you do not have a C-ISAM, DISAM, or Micro Focus database created for your Unix server or Windows system, a database is automatically created when you import. If you used the server or Windows IP address in the Server text box, the name of the new container appears as "C-ISAM", "DISAM", or "Micro Focus" followed by the IP address.*

5. Under **Native Type,** select a data type from the list box. See C-ISAM, DISAM, and Micro Focus Data Types. CONNX determines the SQL data type.

6. Click the **Calc Offsets** button to automatically enter the offset value in the **Offsets** column. Do not use the **Calc Offsets** button if your record contains redefined fields.

7. The remaining five fields display statistics related to the data: Length, Decimal Places, Scale, Array Offset, and Comment. The data in these fields can be modified, depending on the data type. A description of each field is included in the following table:

CONNX CDD Table Columns Tab Fields

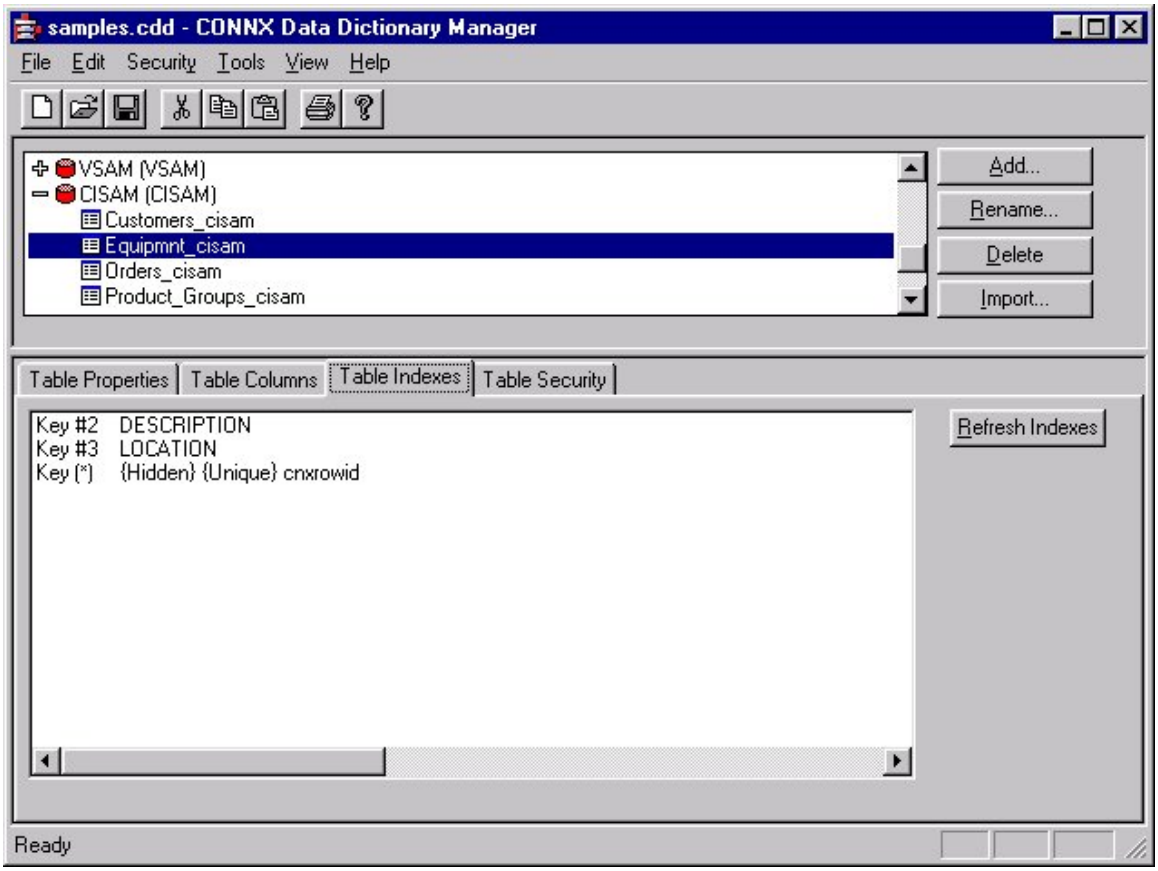
Field	Definition
Offset	Indicates the starting point of a specific field in a table. Length = number of bytes in field.
Length	Physical length in bytes of the column.
Precision	Number of implied decimal places.
Scale	Number of places to move the decimal point in a numeric field. A scale of -2 will convert the number 345.67 to 3.4567, and a scale of 2 will convert the number 345.67 to 34567.
Array	Used to determine the size of one element of an array when used with the Rotated Array Option. Refer to "Using the Rotated Array Assistant".
Comment	Used to provide up to 64 characters of descriptive text.

8. Click the **Tables Indexes** tab, and then click the **Refresh Indexes** button to refresh the indexes defined for the .dat C-ISAM, DISAM, and Micro Focus file.

To refresh an index

The Table Indexes tab displays the indexes in the table. CONNX displays this list automatically. To refresh the indexes:

1. Click the **Table Indexes** tab in the CONNX Data Dictionary Manager window. (CONNX maintains its indexes automatically.) Table indexes may not initially appear.
2. Click the **Refresh Indexes** button to refresh.



Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

C-ISAM, DISAM, and Micro Focus COBOL Imports

To import from C-ISAM, DISAM, and Micro Focus, DISAM, and Micro Focus COBOL FD (File Definition) files

1. Click the **Import** button in the CONNX CDD Windows Application window.
2. The **Import CDD** dialog box appears. Select **C-ISAM, DISAM,** or **Micro Focus COBOL FD files** in the **Import Type** list box.
3. Type a COBOL FD path and file name in the **Enter a C-ISAM, DISAM, or Micro Focus COBOL FD file name** text box, for example, **/home/mydir/customer.fd**
4. Enter the following information in the **Import CDD** dialog box:

1. Type the server name or IP address, a user name, and password in the corresponding text boxes on the **Logon Information** tab. For Windows systems, enter "localhost" as the server name.
2. Port 6500 is listed in the **TCP/IP Port** text box by default. Any change made to the port setting in this text box becomes a permanent change to the port setting of the imported database. See "To edit the C-ISAM, DISAM, and Micro Focus server startup procedure" in the CONNX Installation Guide for information on changing the port setting on the server.
3. Select a **Destination Database** for the imported tables. See Adding a Database Connection for more information.

5. Click the **OK** button. The CONNX Data Dictionary Manager window appears.

6. COBOL FD import specifications do not contain the C-ISAM, DISAM, and Micro Focus data file name, therefore, the C-ISAM, DISAM, and Micro Focus data file must be specified manually after the import is complete. Because the C-ISAM, DISAM, and Micro Focus file name is unknown, the indexes must also be refreshed.

7. From the list of available tables, select each table to import and follow these steps:

1. Click the **Table Properties** tab in the CONNX Data Dictionary Manager window.
2. Click the **Table Indexes** tab in the CONNX Data Dictionary Manager window, and then click the **Refresh Indexes** button. A message saying the indexes are successfully refreshed appears.
3. Save the CDD by clicking **Save** on the **File** menu.

C-ISAM, DISAM, and Micro Focus Text File Imports

C-ISAM, DISAM, or Micro Focus Text File Imports

Because C-ISAM, DISAM, and Micro Focus do not contain information (metadata) on how fields are stored within a record, it is necessary to define the record layout for each C-ISAM, DISAM, or Micro Focus file. CONNX requires the user to create a specification file that will contain these definitions. The specification file can be created with any editor, but is required to be located on the machine where the C-ISAM, DISAM, or Micro Focus files are located.

The C-ISAM, DISAM, or Micro Focus Text File Import Specification enables the following imports to take place:

- **CONNX C-ISAM, DISAM, or Micro Focus Table Imports**
- **CONNX View Imports**

Related Topics

- C-ISAM, DISAM, or Micro Focus View Text File Import Specifications
- C-ISAM, DISAM, or Micro Focus Table Imports
- To import tables or views from a C-ISAM, DISAM, or Micro Focus text file import specification
- To import from C-ISAM, DISAM, or Micro Focus COBOL FD files

C-ISAM, DISAM, and Micro Focus Table Imports

The first line of each record layout should be as follows:

CONNXTABLE, <CONNX Table Name>, <C-ISAM, DISAM, and Micro Focus File Name>,<Record Length>, <Optional SQL Filter Clause>

Note: *The optional SQL Filter Clause is not used in this example.*

Example:

CONNXTABLE, CustomerTable, /home/cisamfiles/customer, 64

CONNX Table Name

User can specify a table name that will be referred to by CONNX.

C-ISAM, DISAM, and Micro Focus File Name

Refers to the physical location and name of C-ISAM, DISAM, and Micro Focus file.

Record Length

Length of the C-ISAM, DISAM, and Micro Focus record.

Optional SQL Filter Clause

Enables filtering of retrieved records.

Example:

SalesAmt > 500.00

Only records that have a sales amount greater than $500. are retrieved.

Column Definition lines must be added after each table definition line, as described above. The syntax for the Column Definition lines is as follows and further defined in the table below:

C-ISAM, DISAM, and Micro Focus Column Specification

<column name>, <column length>, <column offset>, <column type>, <column scale>, <column base>, <column fraction>, <column comment>

C-ISAM, DISAM, and Micro Focus Text File Import Syntax and Description

Syntax	Description
<column name>	Name of the column.
<column offset>	Offset of the column. First column should always be 0.
<column type>	CHAR, INT, LONG, DOUBLE, DECIMAL are valid values.
<column scale>	Must be 0.

<column base>	Must be 0.
<column fraction>	Number of significant fractional values to show. For example, 3.333 would be 3, while 3.14165 would be 5.
<column comment>	Comments field.

Important: *When creating a C-ISAM, DISAM, and Micro Focus text file, it is recommended that you use the column scale syntax rather than the column fraction syntax.*

The following is an example of a C-ISAM, DISAM, or Micro Focus text file specification for **/home/cisam.txt:**

```
CONNXTABLE, employee, /home/employee, 29
ssnum,9,0,CHAR,0,0,0
name,9,9,CHAR,0,0,0
age,2,18,INT,0,0,0
attpct,4,20,FLOAT,0,0,0
salary,4,24,LONG,0,0,0
dectype,4,28,DECIMAL,0,0,0
```

C-ISAM, DISAM, or Micro Focus VIEW Text File Import Specification

The C-ISAM VIEW text file import specification can be used to populate your CDD with predefined CONNX Views using the C-ISAM Text Import Option.

The VIEW text file import specification layout is described below.

The first line of each view layout must contain CONNXVIEW and the view object name as follows:

```
CONNXVIEW, <VIEWOBJECTNAME>
```

The subsequent lines of each view layout must contain the SQL Select statement:

```
SELECT ...
```

The last or footer line in each view must contain:

```
ENDVIEW
```

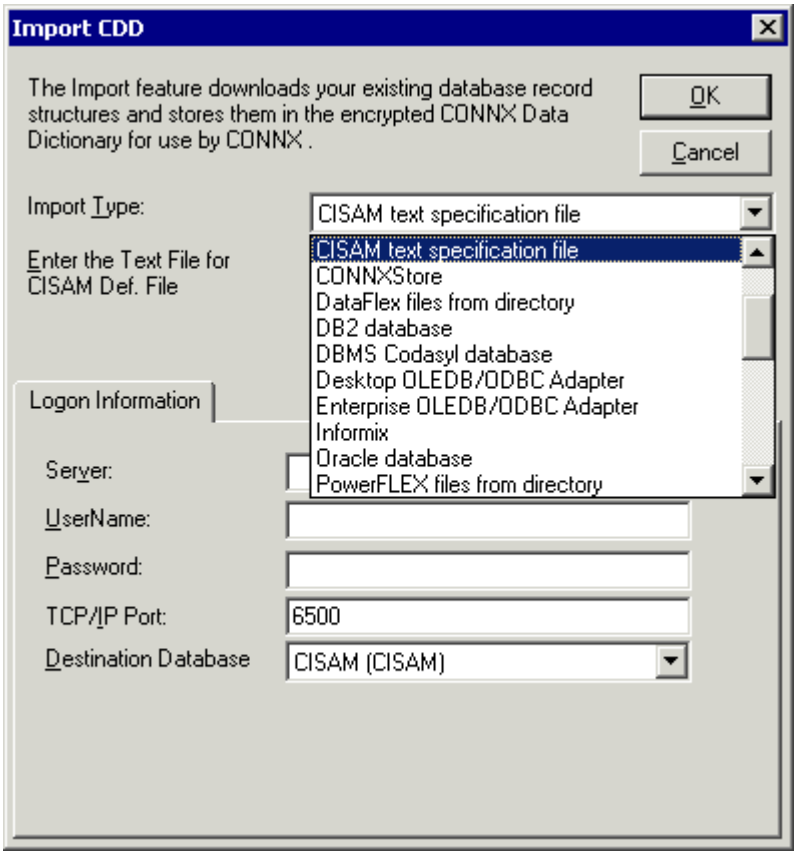
One import text file may contain multiple views, each starting with the same header line shown above and followed by a SELECT statement and a footer line with the word ENDVIEW.

The following is an example of a C-ISAM VIEW import file:

```
CONNXVIEW, NWORDERS
/*This view was requested by Johnathon Jones on 3/1/2001.  He executes
 this view daily to see orders for the Northwest Territory. */

SELECT
ORDERS_C-ISAM.orderid as 'Order' /* Order Number
*/,
ORDERS_C-ISAM.customerid as 'Cust Id' /* Customer Identification */,
CUSTOMERS_C-ISAM.customername as 'Name' /* Name of Customer*/,
CUSTOMERS_C-ISAM.customerstate as 'ST' /* State Ordered by */,
ORDERS_C-ISAM.orderdate as 'Ord Date' /* Date Ordered */,
ORDERS_C-ISAM.productid as 'Product' /* Product number */,
PRODUCTS_C-ISAM.productname as 'Description' /* Product Description */,
ORDERS_C-ISAM.productquantity as 'Qty' /* order quantity */,
PRODUCTS_C-ISAM.productprice as 'Price' /* price per unit */,
(ORDERS_C-ISAM.productquantity * PRODUCTS_C-ISAM.productprice) as 'Ext
 Price' /* Calculate extended price) */
 FROM ORDERS_C-ISAM, CUSTOMERS_C-ISAM, PRODUCTS_C-ISAM /* Tables
 included in view */
WHERE ORDERS_C-ISAM.customerid=CUSTOMERS_C-ISAM.customerid
AND
ORDERS_C-ISAM.productid=PRODUCTS_C-ISAM.productid  AND
CUSTOMERS_C-ISAM.customerstate in ('WA', 'OR', 'MT', 'ID', 'CA') /* Join
 tables together and select only Northwest states */
ENDVIEW
```

To import tables or views from a C-ISAM, DISAM, or Micro Focus text file import specification

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select your platform type from the **Import Type** list box.



3. Type the full C-ISAM, DISAM, or Micro Focus path for the text file you created, for example, **/home/cisamimp.txt**

1. Type the **server name** or **IP address, a user name,** and **password** in the corresponding text boxes on the **Logon Information** tab. For Windows systems, enter "localhost" as the server name.
2. Port 6500 is listed in the text box by default. Port 6500 is listed in the TCPIP Port text box by default. To change the port number, you must stop the server and restart it with the correct port number. See "To edit the C-ISAM, DISAM, or Micro Focus server startup procedure" in the CONNX Installation Guide which can be found either online, on the CONNX CD-ROM, or in hard copy for more information. Log in to the server and then type **connxserver START nnnn** where **nnnn** is the new port number. Any change made to the port setting in this text box becomes a permanent change to the port setting of the imported database.
3. Select a **Destination Database** for the imported tables. See Adding a Database Connection for more information.
4. Click the **OK** button.

4. Save the CDD by selecting the **File** menu and then clicking **Save.**

Micro Focus Sequential Files

To access a sequential file in Micro Focus

Create a file with the same name as the root file but with a .cnx extension. Place the file in the same directory as the data file on the UNIX server. The text file must contain at least three lines; the fourth line is optional.

1. Line 1 should contain one of the following three words: "sequential", "line", or "relative", depending on the file type;
2. Line 2 should contain the maximum record length;
3. Line 3 should contain the minimum record length, and;
4. Line 4 is optional, but, if used, can signify either fixed or variable length. If the length is variable, the line should contain the word "variable".

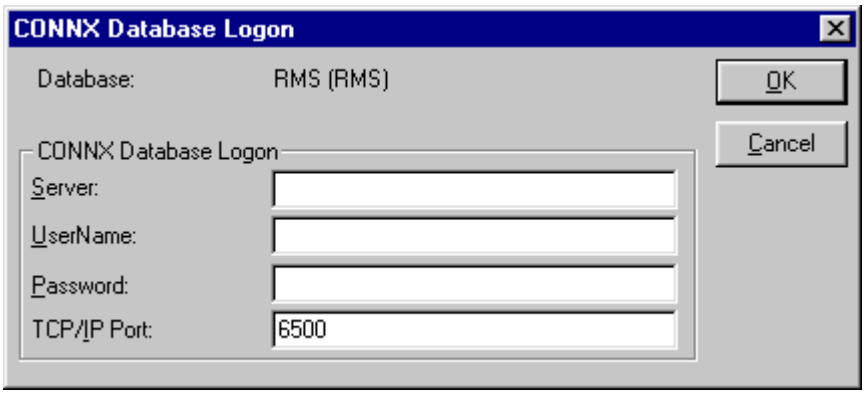
To import from a VAX or Alpha CDD repository

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **RMS VAX or Alpha CDD** in the **Import Type** list box.
3. Type the dictionary path for the CDD record definitions, and then click **OK.** When importing from a VAX or Alpha CDD, the full CDD record name must be specified, for example, **cdd$top.products.customer_record**

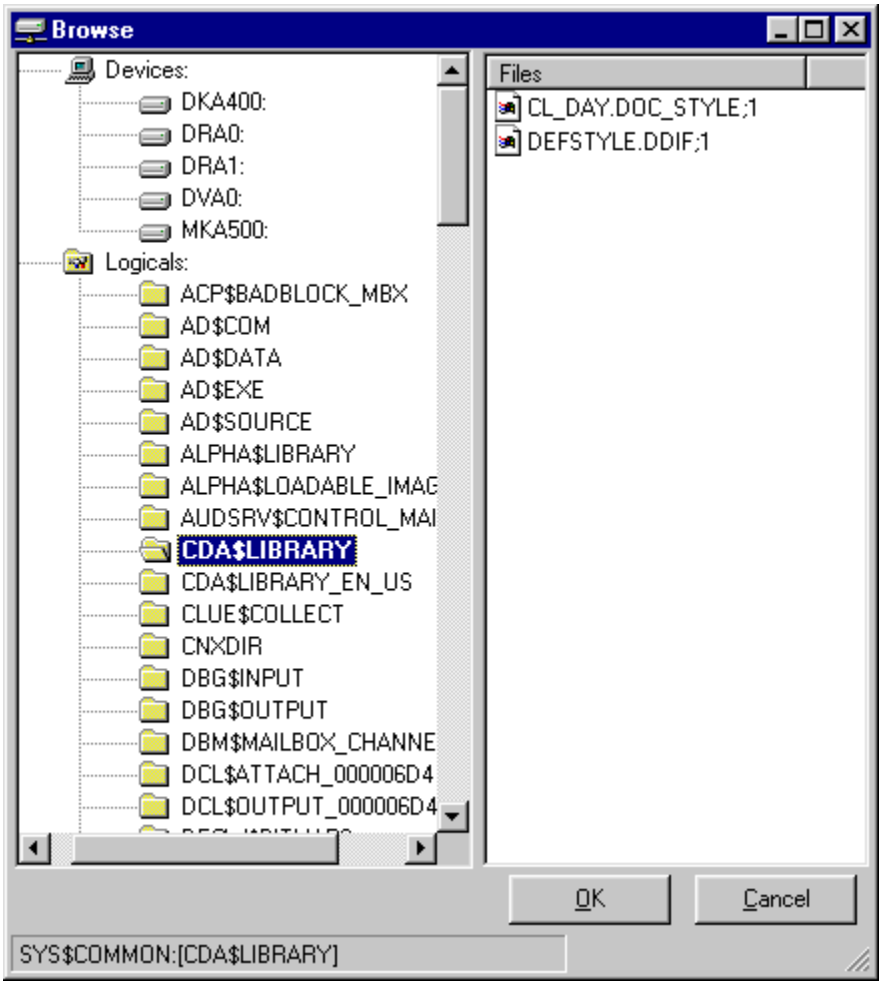
   Note: *Wildcard characters may be used in the dictionary path.*

   Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

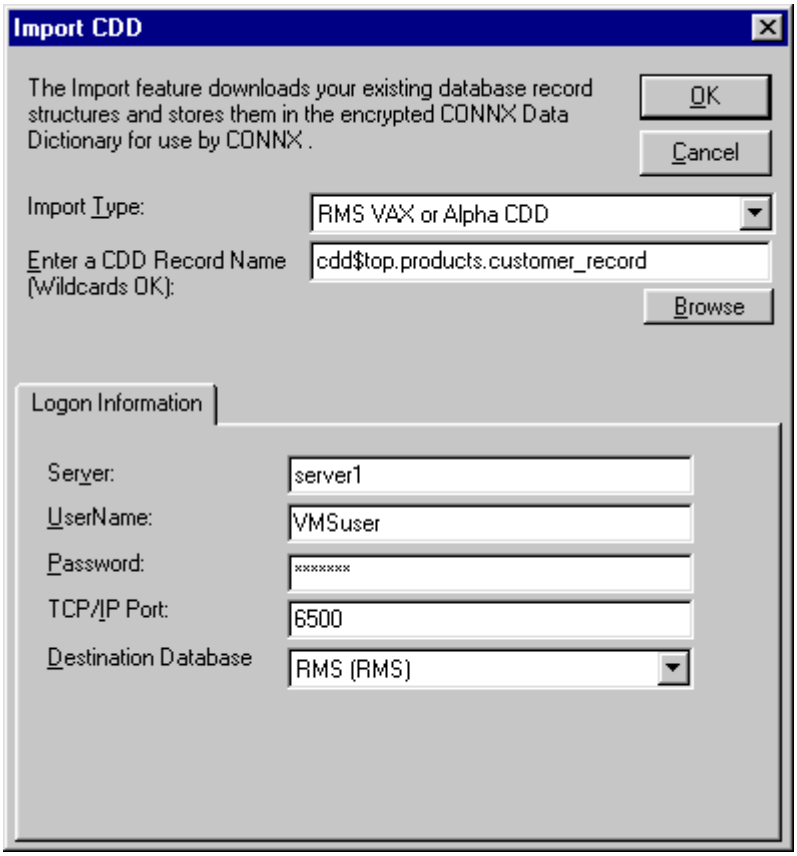
   Note: *Multiple files can be specified in the text box by separating each file name with a comma. The allowable limit is 255 characters.*
4. You can also use the **Browse** button below the text box to locate files to import. If you do not need to use the **Browse** button, proceed to Step 6.
5. Click the **Browse** button. If you are not connected to a VMS server, the **CONNX Database Login** dialog box appears.



1. Type the server name or IP address, a user name, and password in the corresponding text boxes on the **Logon Information** tab.
2. Port 6500 is listed in the **TCPIP Port** text box by default.
3. Click the **OK** button. The **Browse** dialog box appears.
4. Select the file(s) in the **Browse** dialog box, and then click the **OK** button to return to the **Import CDD** dialog box.



6. Enter the following information in the **Import CDD** dialog box:
    1. Type the server name or IP address, a user name, and password in the corresponding text boxes on the **Logon Information** tab.
    2. Port 6500 is listed in the **TCP/IP Port** text box by default. Any change made to the port setting in this text box becomes a permanent change to the port setting of the imported database. See "To edit the OpenVMS Site-Specific Startup Command Procedure" in the CONNX Installation Guide for information on changing the port setting on the server.
    3. Select the destination database from the **Destination Database** list box.
7. Click the **OK** button.



8. VAX or Alpha CDD specifications do not contain the RMS data file name, therefore, the RMS data file must be specified manually or by using the **Browse** button on the **Table Properties** tab in the CONNX Data Dictionary Manager window after the import is complete. See To use the CONNX Browse button. Because the RMS file name is unknown, the indexes must also be refreshed.
See To view an index for information on refreshing indexes.

9. From the list of available tables, select each table to import and follow these steps:
    1. Click the **Table Properties** tab in the CONNX Data Dictionary Manager window.
    2. Type the data file name and path in the **RMS file name** text box.
    3. Click the **Table Indexes** tab in the CONNX Data Dictionary Manager window, and then click the **Refresh Indexes** button. A message saying the indexes are successfully refreshed appears.
10. Save the CDD by clicking **Save** on the **File** menu.

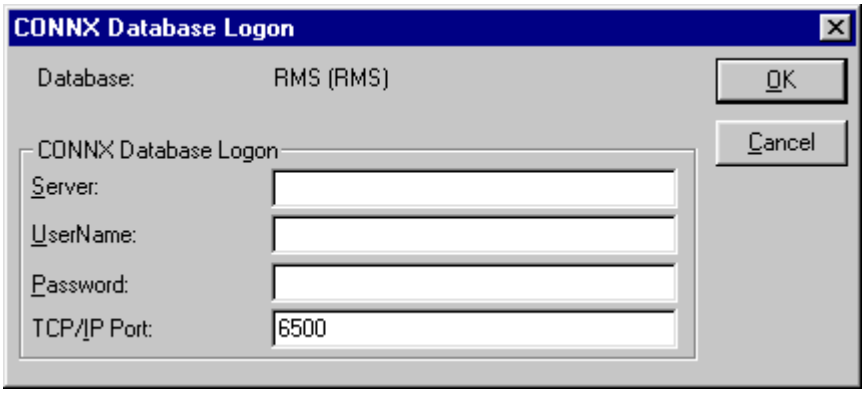
To import from Powerhouse PDL (Powerhouse Definition Language) files (RMS only)

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **RMS Powerhouse PDL** files in the **Import Type** list box.
3. Type a Powerhouse PDL file name in the **Enter a Powerhouse PDL file name** text box. Specify the full file name for the Powerhouse PDL, for example, DKA600:[MYDIR]QSHOGEN.PDL

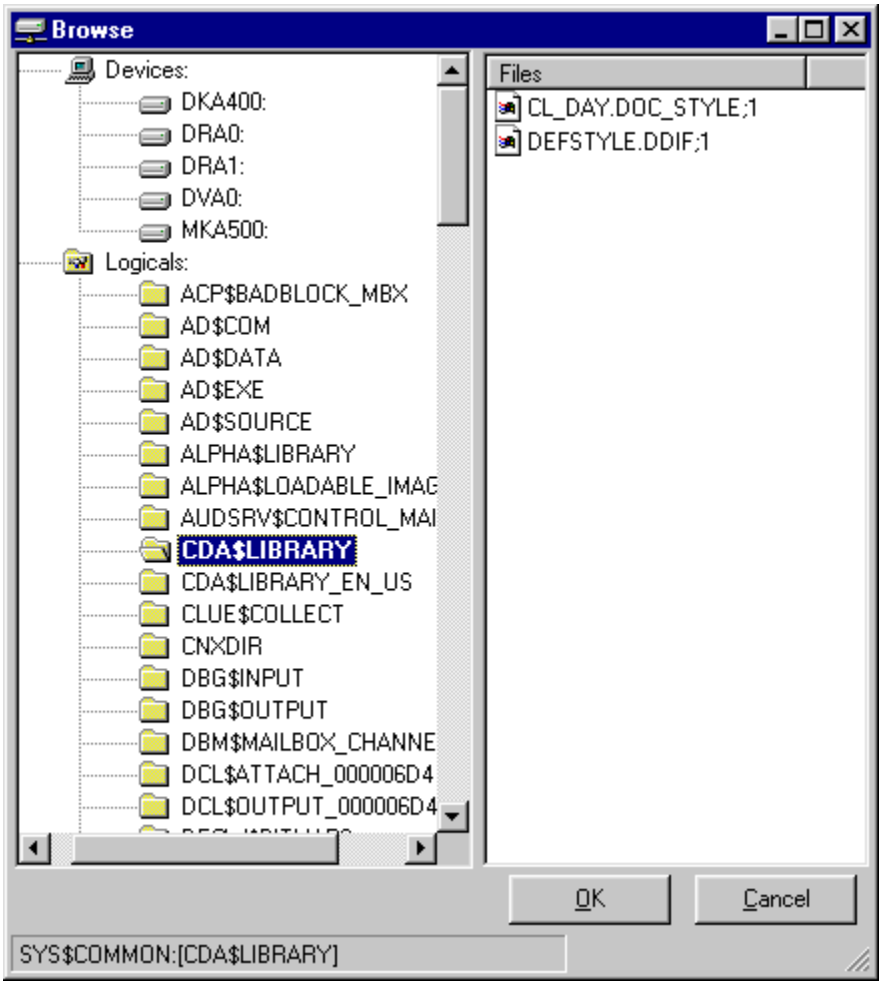
   Note: *Wildcard characters may be used in the dictionary path.*

   Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

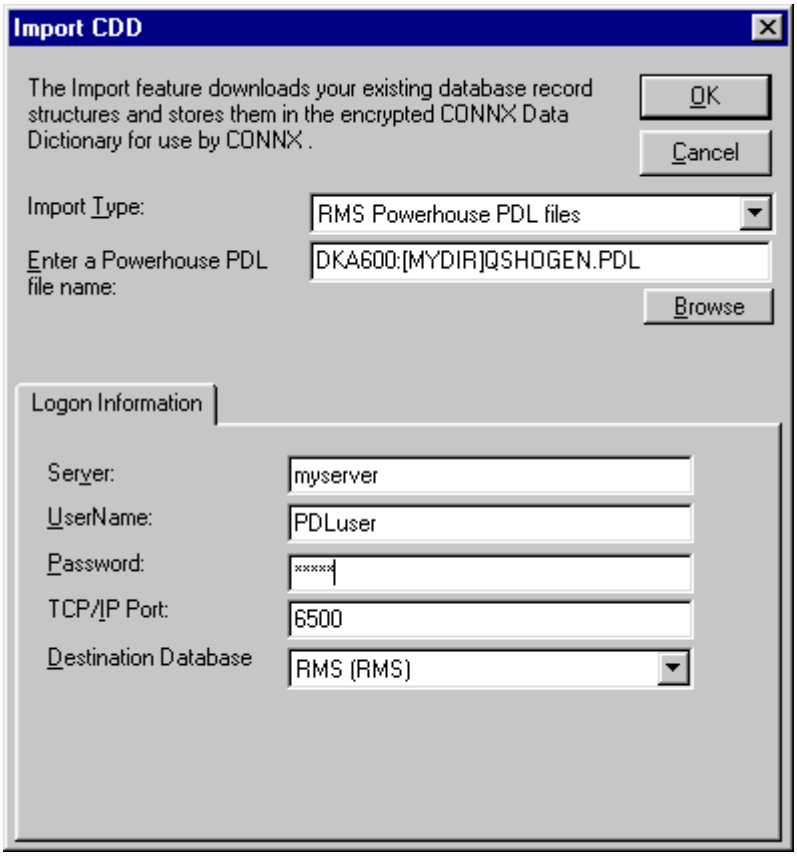
   Note: *Multiple files can be specified in the text box by separating each file name with a comma. The allowable limit is 255 characters.*
4. You can also use the **Browse** button below the text box to locate files to import. If you do not need to use the **Browse** button, proceed to Step 6.
5. Click the **Browse** button. If you are not connected to a VMS server, the **CONNX Database Login** dialog box appears.



1. Type the server name or IP address, a user name, and password in the corresponding text boxes.
2. Port 6500 is listed in the **TCPIP Port** text box by default.
3. Click the **OK** button. The **Browse** dialog box appears.
4. Select the file(s) in the **Browse** dialog box, and then click the **OK** button to return to the **Import CDD** dialog box.



6. Enter the following information in the **Import CDD** dialog box:
    1. Type the server name or IP address, a user name, and password in the corresponding text boxes on the **Logon Information** tab.
    2. Port 6500 is listed in the **TCP/IP Port** text box by default. Any change made to the port setting in this text box becomes a permanent change to the port setting of the imported database. See "To edit the OpenVMS Site-Specific Startup Command Procedure" in the CONNX Installation Guide for information on changing the port setting on the server.
    3. Select the destination database from the **Destination Database** list box.
7. Click the **OK** button.



8. Powerhouse PDL import specifications do not contain the RMS data file name, therefore, the RMS data file must be specified manually or by using the **Browse** button on the **Table Properties** tab in the CONNX Data Dictionary Manager window after the import is complete. See To use the CONNX Browse button. Because the RMS file name is unknown, the indexes must also be refreshed. See To view an index for information on refreshing indexes.

9. From the list of available tables, select each table to import and follow these steps:

    1. Click the **Table Properties** tab in the CONNX Data Dictionary Manager window.
    2. Type the data file name and path in the **RMS file name** text box.
    3. Click the **Table Indexes** tab in the CONNX Data Dictionary Manager window, and then click the **Refresh Indexes** button. A message saying the indexes are successfully refreshed appears.
    4. Save the CDD by clicking **Save** on the **File** menu.

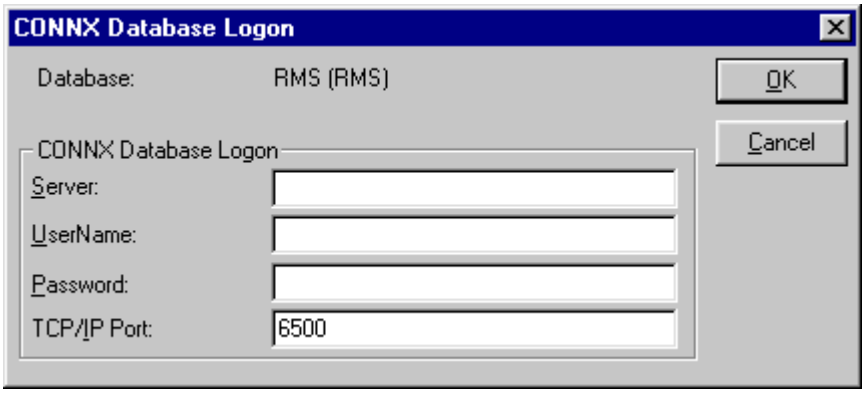
To import from RMS COBOL FD (File Definition) files (RMS only)

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **RMS COBOL FD files** in the **Import Type** list box.
3. Type a COBOL FD path and file name in the Enter a COBOL FD file name text box. Specify the full file name for the COBOL FD file, for example, DKA600:[MYDIR]CUSTOMER.FD

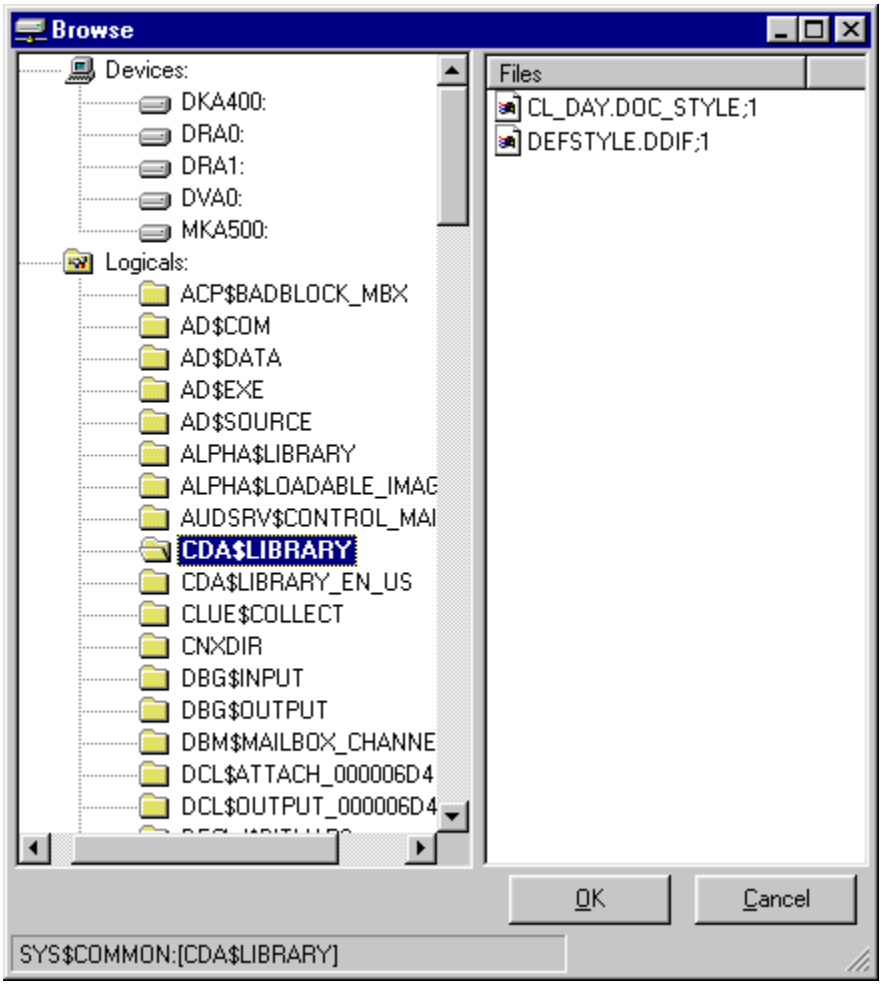
   Note: *Wildcard characters may be used in the dictionary path.*

   Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

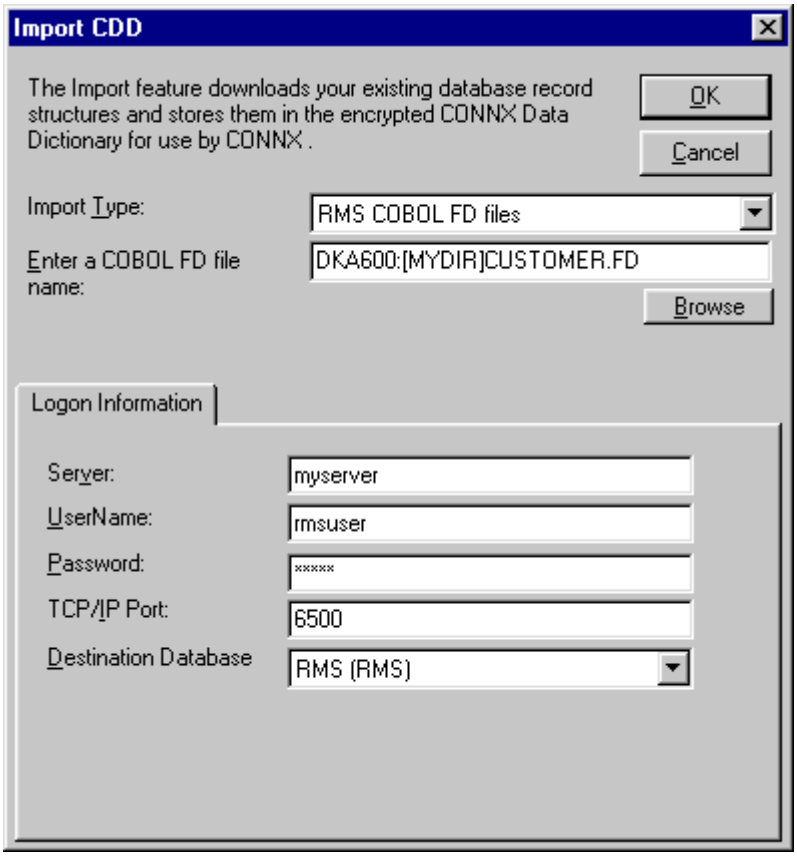
   Note: *Multiple files can be specified in the text box by separating each file name with a comma. The allowable limit is 255 characters.*
4. You can also use the **Browse** button below the text box to locate files to import. If you do not need to use the **Browse** button, proceed to Step 6.
5. Click the **Browse** button. If you are not connected to a VMS server, the **CONNX Database Login** dialog box appears.



1. Type the server name or IP address, a user name, and password in the corresponding text boxes.
2. Port 6500 is listed in the **TCPIP Port** text box by default.
3. Click the **OK** button. The **Browse** dialog box appears.
4. Select the file(s) in the **Browse** dialog box, and then click the **OK** button to return to the **Import CDD** dialog box.



6. Enter the following information in the **Import CDD** dialog box:
    1. Type the server name or IP address, a user name, and password in the corresponding text boxes on the **Logon Information** tab.
    2. Port 6500 is listed in the **TCP/IP Port** text box by default. Any change made to the port setting in this text box becomes a permanent change to the port setting of the imported database. See "To edit the OpenVMS Site-Specific Startup Command Procedure" in the CONNX Installation Guide for information on changing the port setting on the server.
    3. Select the destination database from the **Destination Database** list box. See Adding a database connection for more information.
7. Click the **OK** button.



8. COBOL FD import specifications do not contain the RMS data file name, therefore, the RMS data file must be specified manually or by using the **Browse** button on the **Table Properties** tab in the CONNX Data Dictionary Manager window after the import is complete. See To use the CONNX Browse button. Because the RMS file name is unknown, the indexes must also be refreshed. See To view an index for information on refreshing indexes.

9. From the list of available tables, select each table to import and follow these steps:
    1. Click the **Table Properties** tab in the CONNX Data Dictionary Manager window.
    2. Type the data file name and path in the **RMS file name** text box.
    3. Click the **Table Indexes** tab in the CONNX Data Dictionary Manager window, and then click the **Refresh Indexes** button. A message saying the indexes are successfully refreshed appears.
    4. Save the CDD by clicking **Save** on the **File** menu.

To import from RMS SCT COBOL FD (File Definition) files

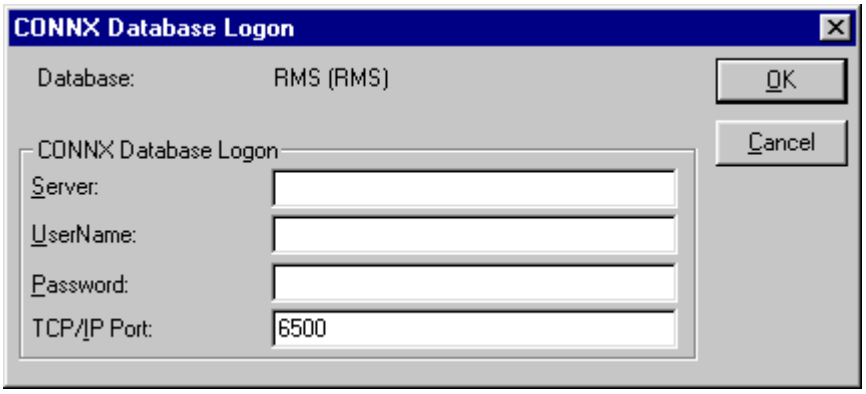
This procedure is site-specific for SCT customers. If you do not use the standard SCT logicals, refer to the SCTLOGICAL import setting. See also CONNX and SCT Import Rules.

1. Click the **Import** button in the CONNX CDD Windows Application window.
2. The **Import CDD** dialog box appears. Select **RMS SCT COBOL FD files** in the **Import Type** list box.
3. Type an SCT COBOL FD path and file name in the **Enter an SCT COBOL FD file name** text box. Specify the full filename for the SCT COBOL FD file, for example, **SI$SOURCE:ACADRC.LIB**

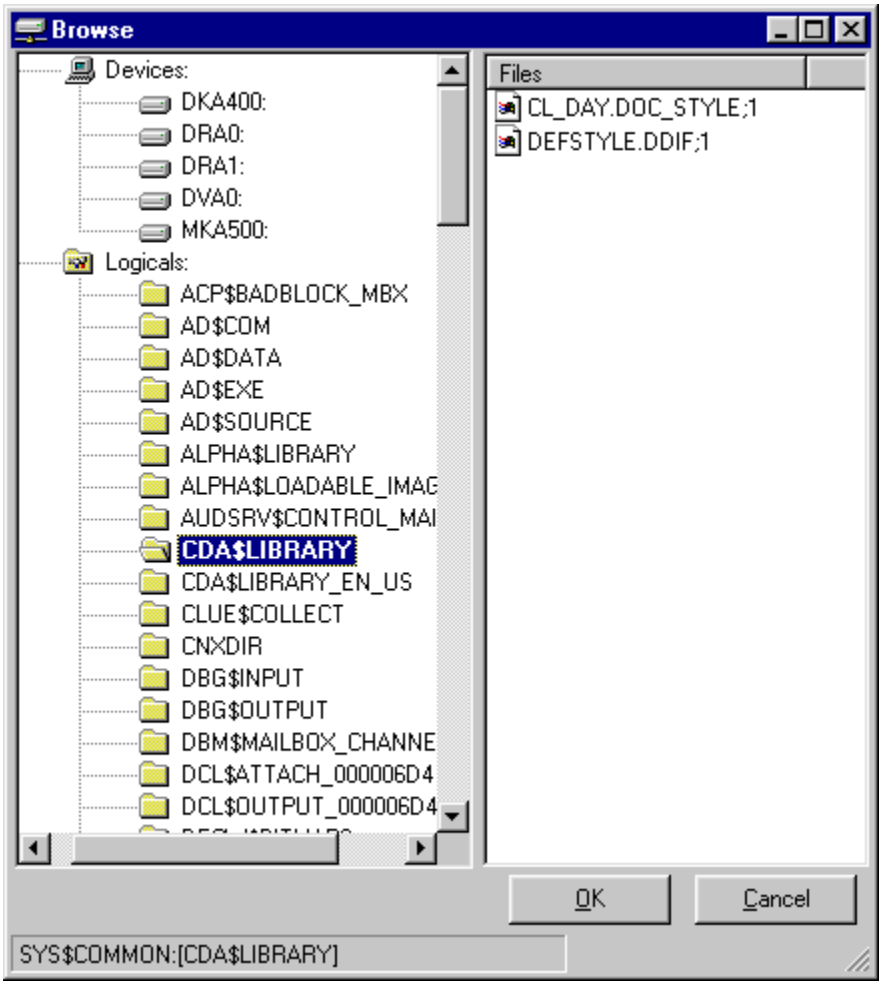
   Note: *Use the following wildcard syntax to import multiple SCT COBOL FD files from the SIS module: SI$SOURCE:*RC.LIB*

   Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

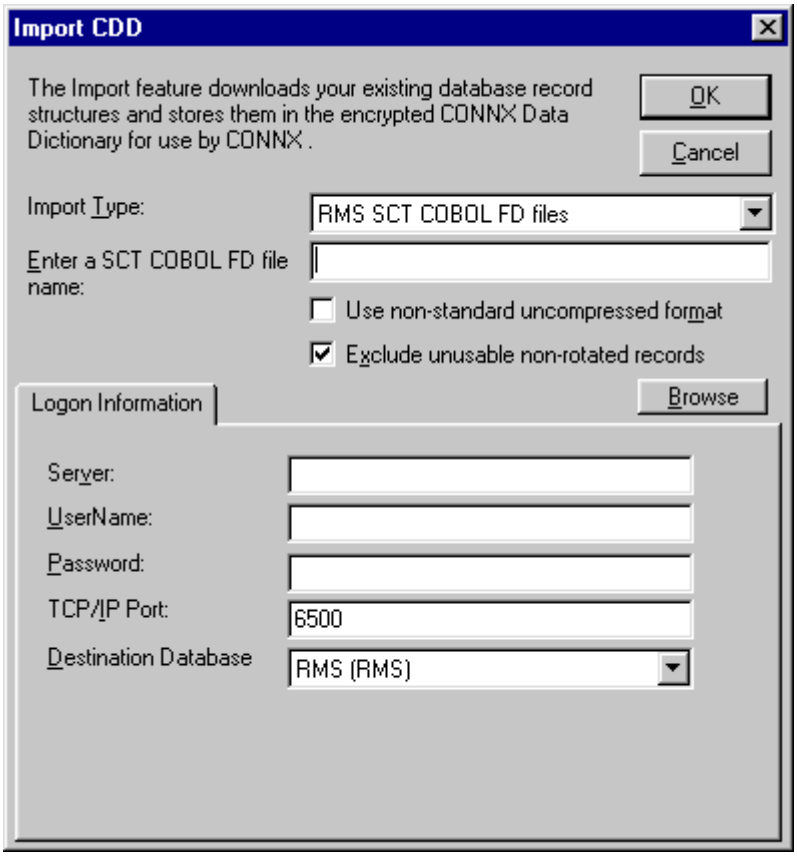
   Note: *Multiple files can be specified in the Enter an SCT COBOL FD file name text box by separating each file name with a comma. The allowable limit is 255 characters.*
4. You can also use the **Browse** button below the text box to locate files to import. If you do not need to use the **Browse** button, proceed to Step 6.
5. Click the **Browse** button. If you are not connected to a VMS server, the **CONNX Database Login** dialog box appears.



1. Type the server name or IP address, a user name, and password in the corresponding text boxes.
2. Port 6500 is listed in the **TCPIP Port** text box by default.
3. Click the **OK** button. The **Browse** dialog box appears.
4. Select the file(s) in the **Browse** dialog box, and then click the **OK** button to return to the **Import CDD** dialog box.



6. Enter the following information in the **Import CDD** dialog box:
    1. Type the server name or IP address, a user name, and password in the corresponding text boxes on the **Logon Information** tab.
    2. Port 6500 is listed in the **TCP/IP Port** text box by default. Any change made to the port setting in this text box becomes a permanent change to the port setting of the imported database. See "To edit the OpenVMS Site-Specific Startup Command Procedure" in the CONNX Installation Guide for information on changing the port setting on the server.
    3. Select the destination database from the **Destination Database** list box. See Adding a database connection for more information.
7. Click the **OK** button.



8. The RMS data files name is automatically entered in the table properties using the standard SCT data logicals, for example, the standard SCT data logicals, for example, SI$DATA:ADFILE.DAT. See CONNX and SCT Import Rules for more information on the import logic.

9. Save the CDD by clicking **Save** on the **File** menu.

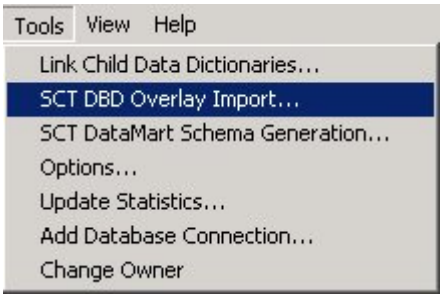
Note: *Use the following wildcard syntax to import multiple SCT COBOL FD files from the SIS module: SI$SOURCE:*RC.LIB*

To perform an RMS SCT DBD (Database Definition) Overlay Import

Once imported, you can overlay existing COBOL field names with the field names in your DBD. You can also use this overlay feature to add comments that correlate to the help screens in your application. Additionally, the comments contain the SCT mnemonic for each field, making it easy to locate a desired field using the CONNX Find feature.

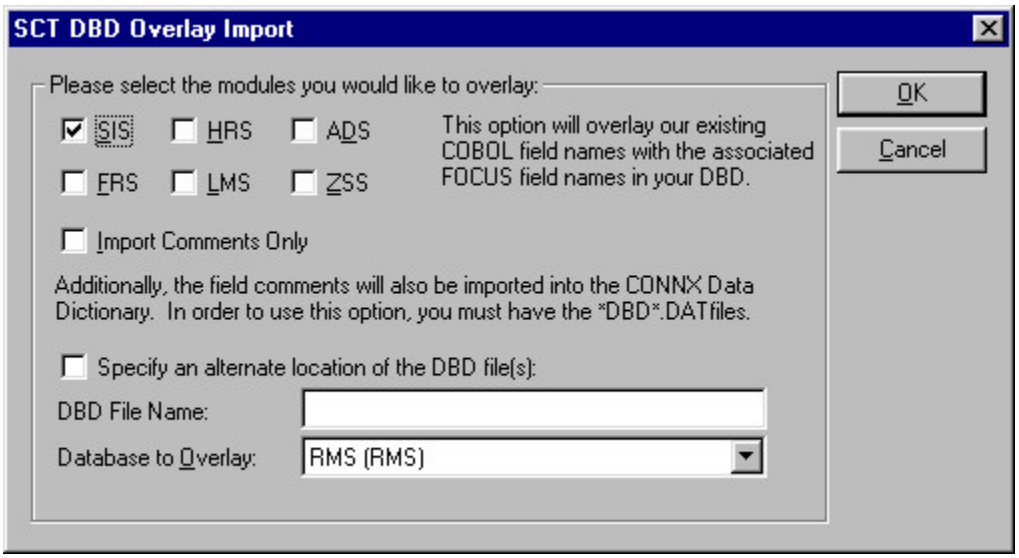
For more information, see Overlay Conventions.

1. Be sure you have a backup copy of your CDD before performing an overlay.
2. On the **Tools** menu in the CONNX Data Dictionary Manager window, select **SCT DBD Overlay Import.**
3. The **SCT DBD Overlay Import** dialog box appears. Select the check box in front of the modules you want to overlay. Select the **Import Comments Only** check box to add only the DBD comments currently in your CDD. Importing comments only adds the mnemonic file and field identifier and any comments existing for each field in the DBD file.



By default, CONNX searches for "DBD".DAT in the $data directory for the module being overlaid. If applicable, select the **Specify an alternate location of the DBD file** check box and then type the location in the text box below. Wildcard characters may be used in the file path.

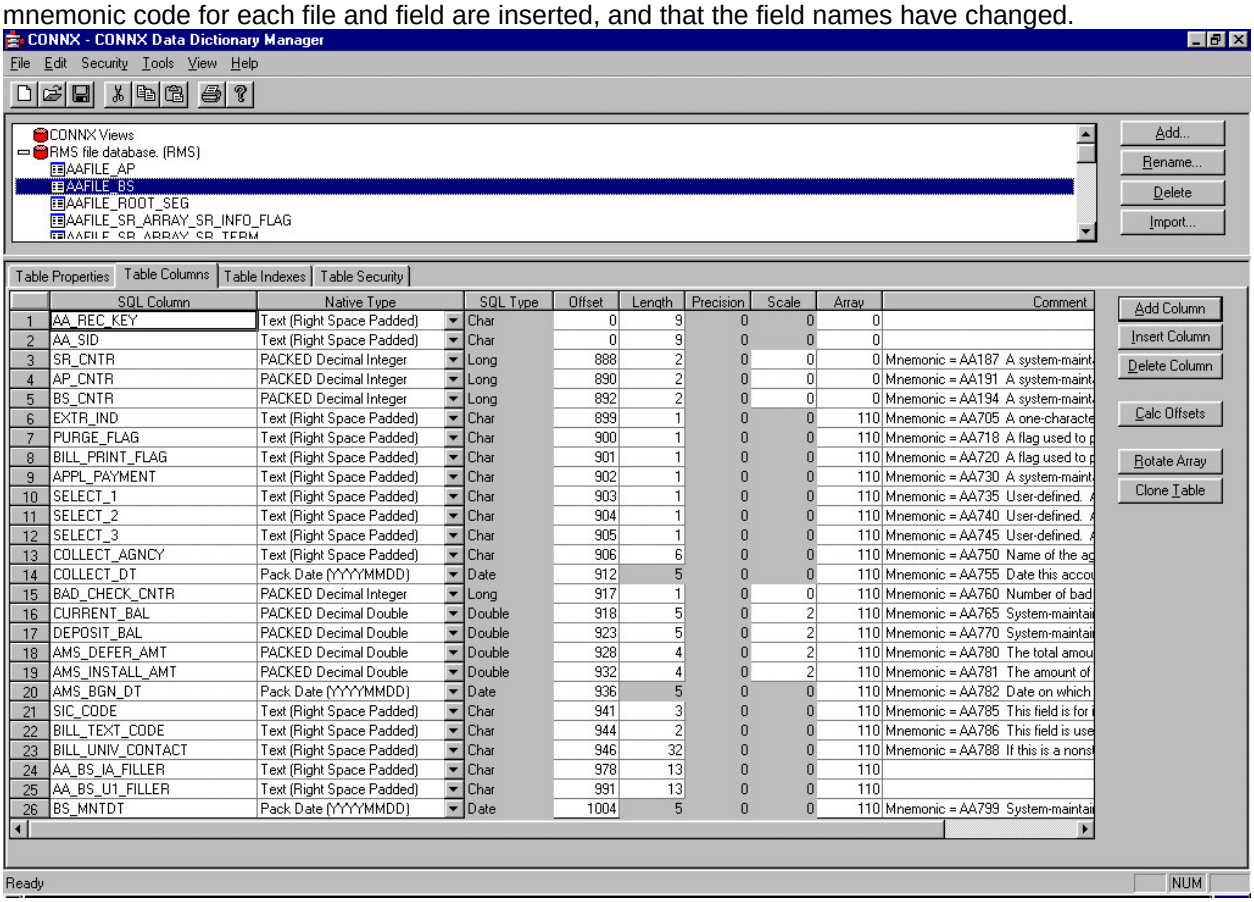
4. Select the RMS database to overlay in the **Database to Overlay** text box.



5. Click the **OK** button.
6. Type the server name or IP address, a user name, and password in the corresponding text boxes on the Logon Information tab.

1. Port 6500 is listed in the TCPIP Port text box by default. Any change made to the port setting in this text box becomes a permanent change to the port setting of the imported database. See "To Edit the OpenVMS Site-Specific Startup Command Procedure" in the CONNX Installation Guide for information about changing the port setting on the server.
2. Note that the SIS module is used for this example.
3. The conversion to DBD field names may take a few minutes, depending on the size of the database.

7. Once the tables are converted, select a table from the list in the upper pane. The COBOL field names in the selected modules are converted to DBD field names. Note that comments containing a mnemonic code for each file and field are inserted, and that the field names have changed.



Related Topic

CONNX and SCT Import Rules

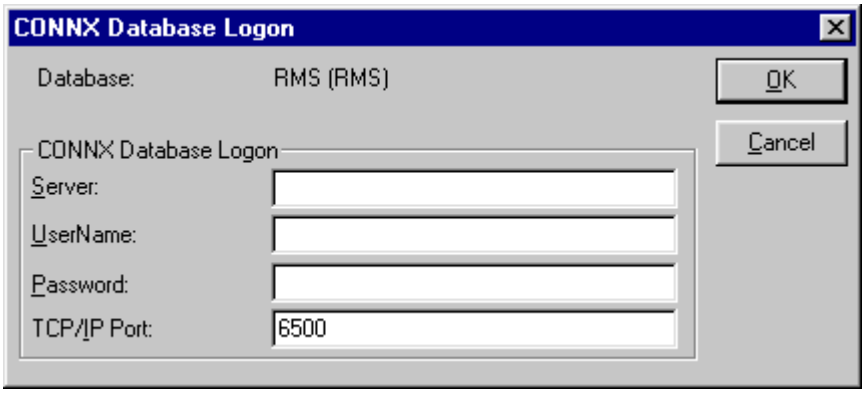
To import an existing DIBOL table definition (RMS only)

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **RMS DIBOL files** in the **Import Type** list box.
3. Type a dictionary path for the DIBOL record definitions in the **Enter a DIBOL file name** text box, for example, **DKA600:[MYDIR]CUSTOMER.DBL**

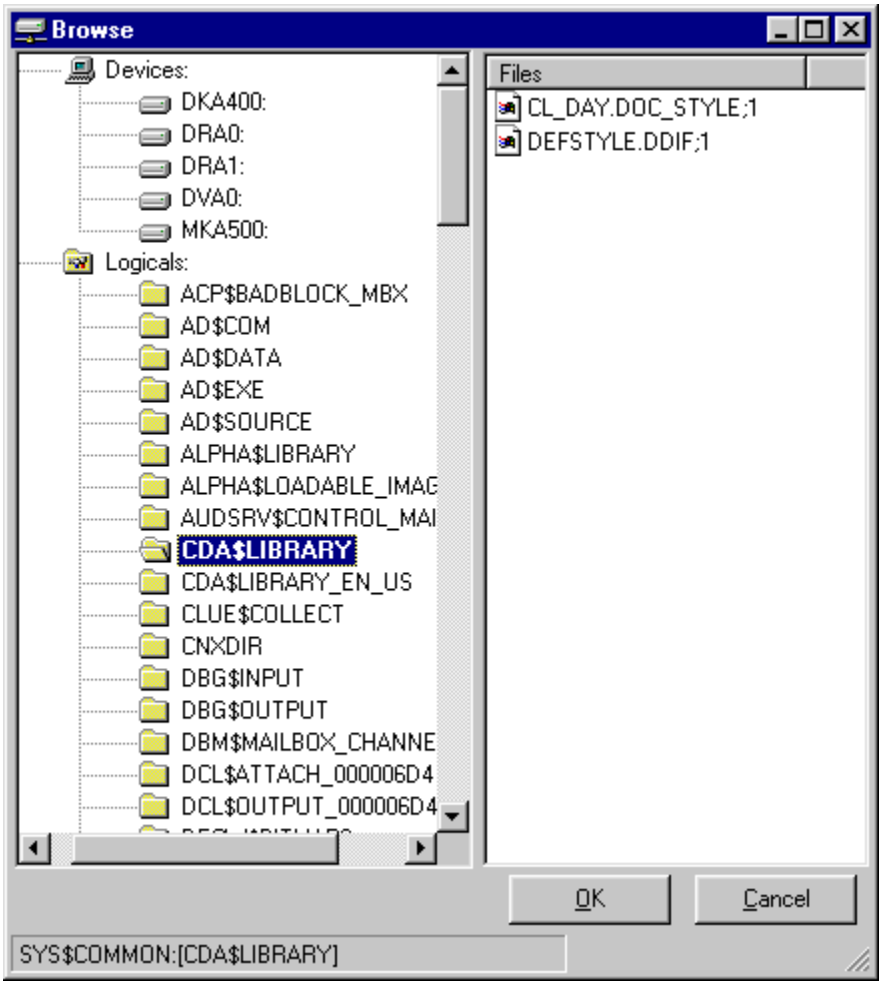
   Note: *Wildcard characters may be used in the dictionary path.*

   Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

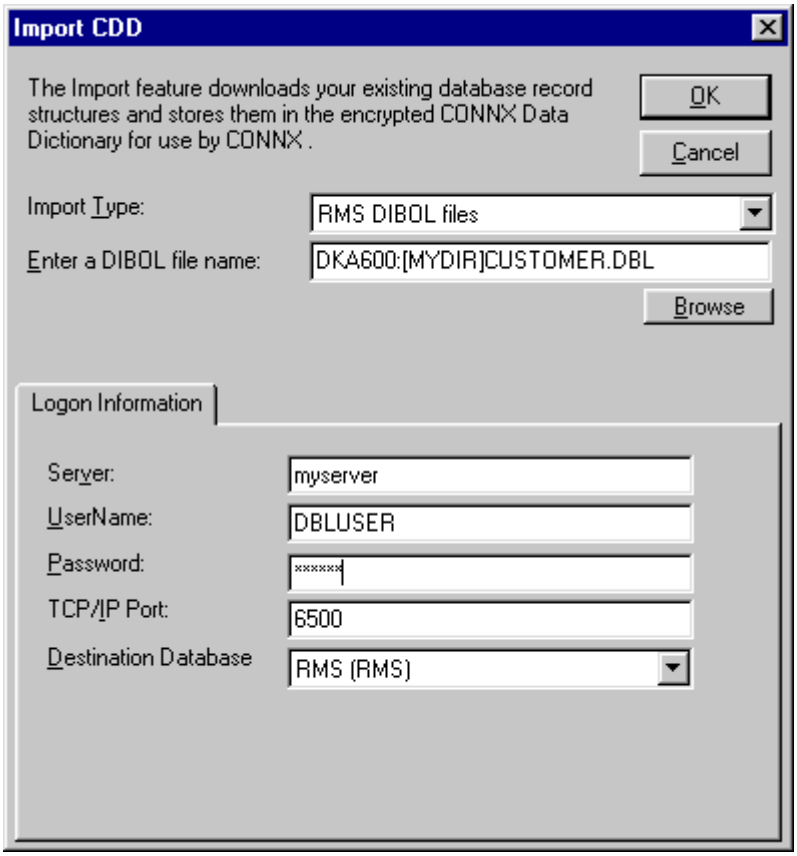
   Note: *Multiple files can be specified in the text box by separating each file name with a comma. The allowable limit is 255 characters.*
4. You can also use the **Browse** button below the text box to locate files to import. If you do not ned to use the **Browse** button, proceed to Step 6.
5. Click the **Browse** button. If you are not connected to a VMS server, the **CONNX Database Login** dialog box appears.



1. Type the server name or IP address, a user name, and password in the corresponding text boxes.
2. Port 6500 is listed in the **TCPIP Port** text box by default.
3. Click the **OK** button. The **Browse** dialog box appears.
4. Select the file(s) in the **Browse** dialog box, and then click the **OK** button to return to the **Import CDD** dialog box.



6. Enter the following information in the **Import CDD** dialog box:
    1. Type the server name or IP address, a user name, and password in the corresponding text boxes on the **Logon Information** tab.
    2. Port 6500 is listed in the **TCP/IP Port** text box by default. Any change made to the port setting in this text box becomes a permanent change to the port setting of the imported database. See "To edit the OpenVMS Site-Specific Startup Command Procedure" in the CONNX Installation Guide for information on changing the port setting on the server.
    3. Select the destination database from the **Destination Database** list box. See Adding a database connection for more information.
7. Click the **OK** button.



8. DIBOL import specifications do not contain the RMS data file name, therefore, the RMS data file must be specified manually or by using the **Browse** button on the **Table Properties** tab in the CONNX Data Dictionary Manager window after the import is complete. See To use the CONNX Browse button. Because the RMS file name is unknown, the indexes must also be refreshed. See To view an index for information on refreshing indexes.

From the list of available tables, select each table to import and follow these steps:

1. Click the **Table Properties** tab in the CONNX Data Dictionary Manager window.
2. Type the data file name and path in the **RMS file name** text box.
3. Click the **Table Indexes** tab in the CONNX Data Dictionary Manager window, and then click the **Refresh Indexes** button. A message saying the indexes are successfully refreshed appears.
4. Save the CDD by clicking **Save** on the **File** menu.

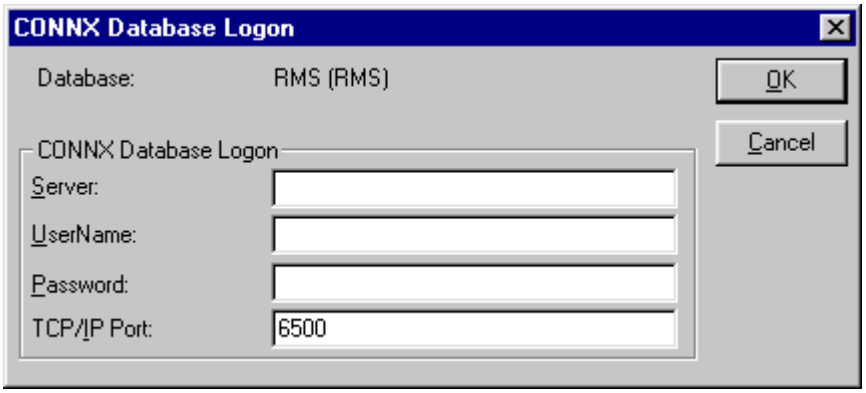
To import existing formatted DDL (Data Definition Language) table definitions

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **RMS DIBOL files** in the **Import Type** list box.
3. Type a dictionary path for the DDL record definition in the **Enter a DDL file name** text box, for example, **DKA600:[MYDIR]CUSTOMER.DDL**

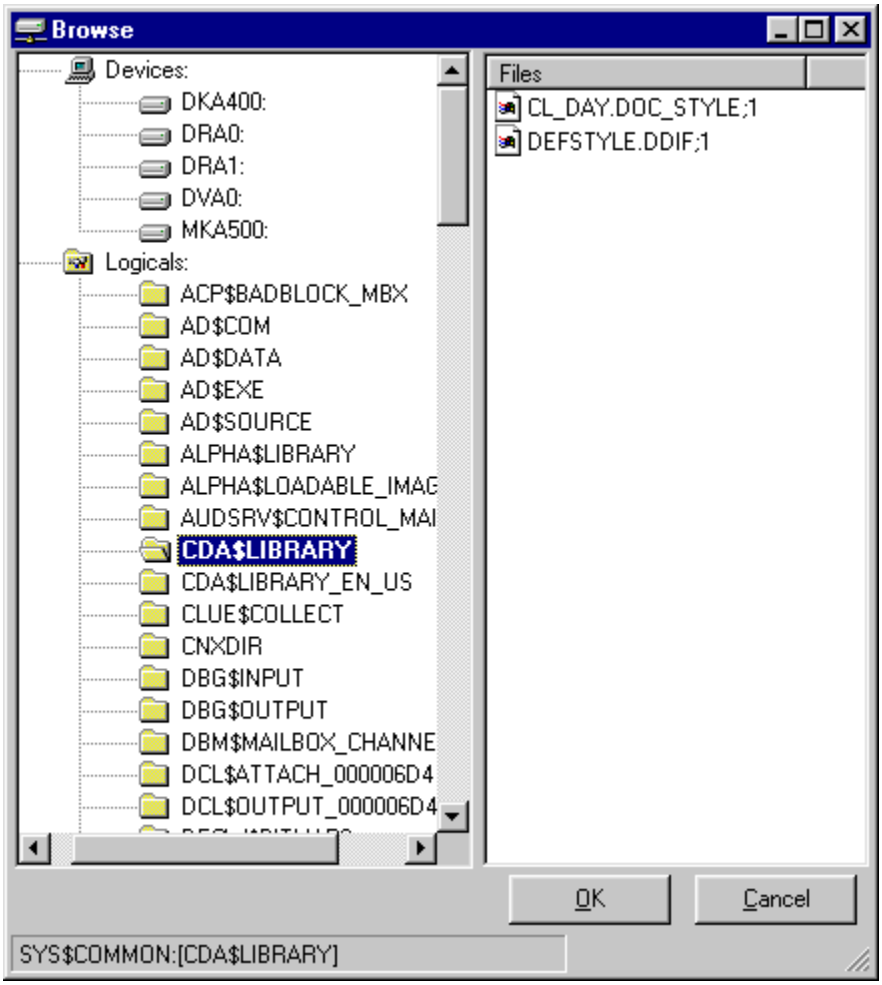
   Note: *Wildcard characters may be used in the dictionary path.*

   Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

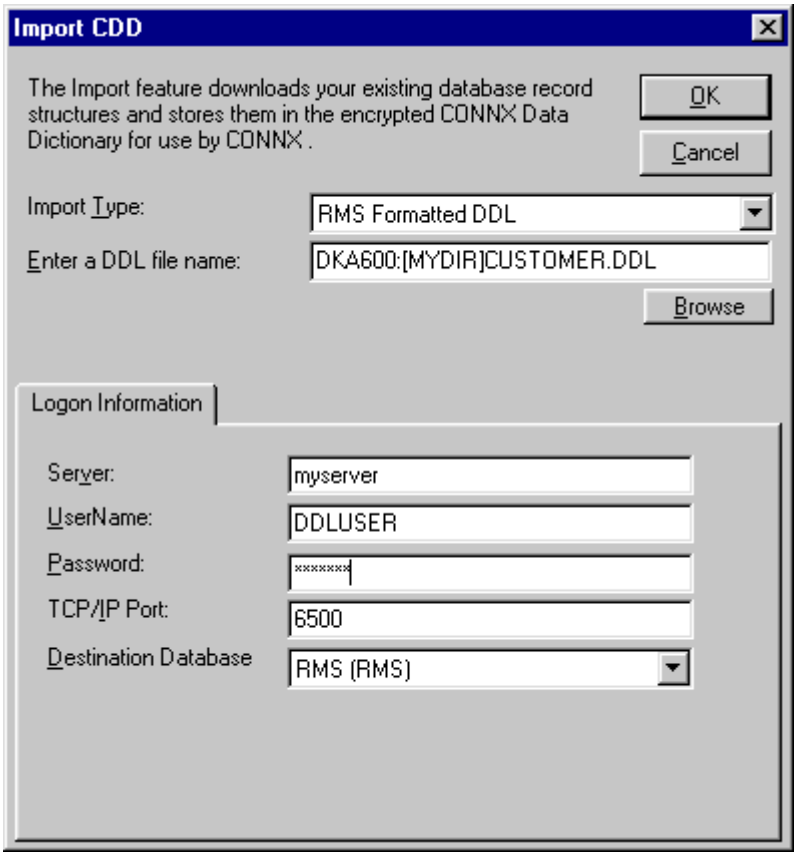
   Note: *Multiple files can be specified in the text box by separating each file name with a comma. The allowable limit is 255 characters.*
4. You can also use the **Browse** button below the text box to locate files to import. If you do not need to use the **Browse** button, proceed to Step 6.
5. Click the **Browse** button. If you are not connected to a VMS server, the **CONNX Database Login** dialog box appears.



1. Type the server name or IP address, a user name, and password in the corresponding text boxes.
2. Port 6500 is listed in the **TCPIP Port** text box by default.
3. Click the **OK** button. The **Browse** dialog box appears.
4. Select the file(s) in the **Browse** dialog box, and then click the **OK** button to return to the **Import CDD** dialog box.



6. Enter the following information in the **Import CDD** dialog box:
    1. Type the server name or IP address, a user name, and password in the corresponding text boxes on the **Logon Information** tab.
    2. Port 6500 is listed in the **TCP/IP Port** text box by default. Any change made to the port setting in this text box becomes a permanent change to the port setting of the imported database. See "To edit the OpenVMS Site-Specific Startup Command Procedure" in the CONNX Installation Guide for information on changing the port setting on the server.
    3. Select the destination database from the **Destination Database** list box. See Adding a database connection for more information.
7. Click the **OK** button.



8. Formatted DDL import specifications do not contain the RMS data file name, therefore, the RMS data file must be specified manually or by using the **Browse** button on the **Table Properties** tab in the CONNX Data Dictionary Manager window after the import is complete. See To use the CONNX Browse button. Because the RMS file name is unknown, the indexes must also be refreshed. See To view an index for information on refreshing indexes.

From the list of available tables, select each table to import and follow these steps:

1. Click the **Table Properties** tab in the CONNX Data Dictionary Manager window.
2. Type the data file name and path in the **RMS file name** text box.
3. Click the **Table Indexes** tab in the CONNX Data Dictionary Manager window, and then click the **Refresh Indexes** button. A message saying the indexes are successfully refreshed appears.
4. Save the CDD by clicking **Save** on the **File** menu.

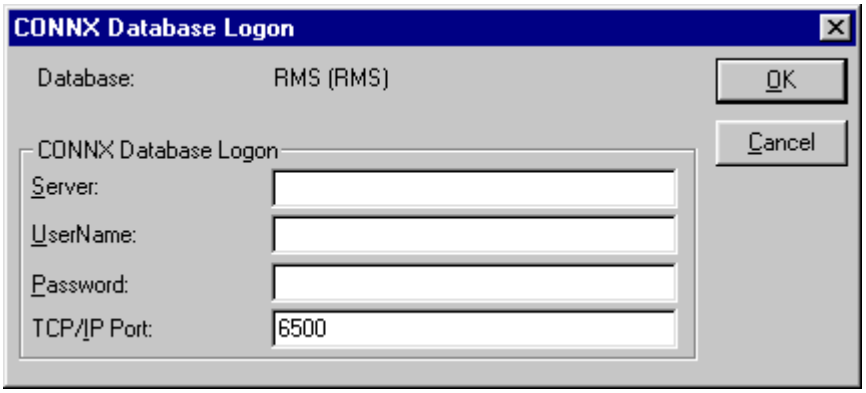
To import from RMS VAX Basic MAP files

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **RMS VAX Basic files** in the **Import Type** list box.
3. Type a VAX Basic path and file name in the **Enter a VAX Basic file name** text box, for example, **DKA600:[MYDIR]CUSTOMER_MAP.BS**

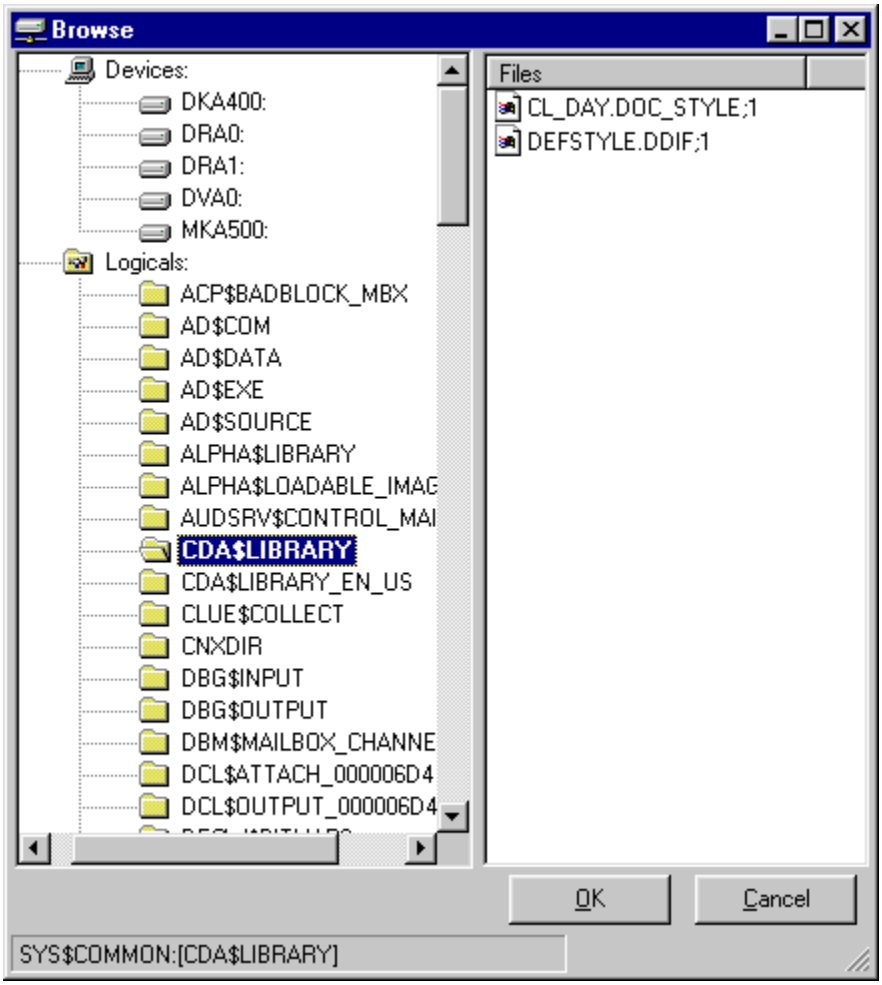
   Note: *Wildcard characters may be used in the dictionary path.*

   Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

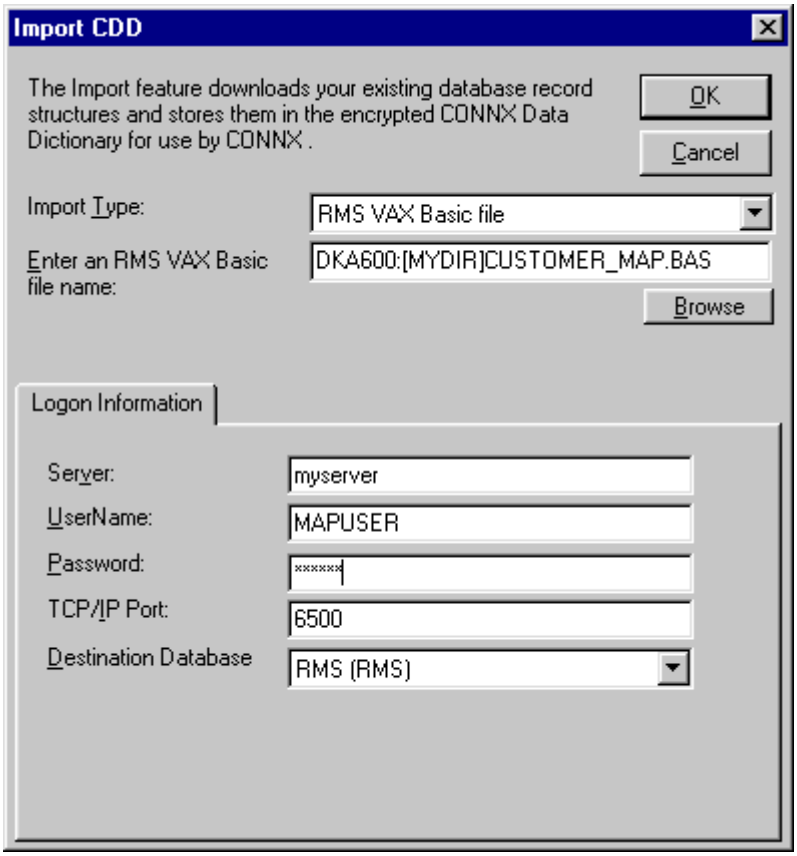
   Note: *Multiple files can be specified in the text box by separating each file name with a comma. The allowable limit is 255 characters.*
4. You can also use the **Browse** button below the text box to locate files to import. If you do not need to use the **Browse** button, proceed to Step 6.
5. Click the **Browse** button. If you are not connected to a VMS server, the **CONNX Database Login** dialog box appears.



1. Type the server name or IP address, a user name, and password in the corresponding text boxes.
2. Port 6500 is listed in the **TCPIP Port** text box by default.
3. Click the **OK** button. The **Browse** dialog box appears.
4. Select the file(s) in the **Browse** dialog box, and then click the **OK** button to return to the **Import CDD** dialog box.



6. Enter the following information in the **Import CDD** dialog box:
    1. Type the server name or IP address, a user name, and password in the corresponding text boxes on the **Logon Information** tab.
    2. Port 6500 is listed in the **TCP/IP Port** text box by default.
    3. Select the destination database from the **Destination Database** list box. See Adding a database connection for more information.
7. Click the **OK** button.



8. VAX Basic import specifications do not contain the RMS data file name, therefore, the RMS data file must be specified manually or by using the **Browse** button on the **Table Properties** tab in the CONNX Data Dictionary Manager window after the import is complete. See To use the CONNX Browse button. Because the RMS file name is unknown, the indexes must also be refreshed.
See To view an index for information on refreshing indexes.

From the list of available tables, select each table to import and follow these steps:

1. Click the **Table Properties** tab in the CONNX Data Dictionary Manager window.
2. Type the data file name and path in the **RMS file name** text box.
3. Click the **Table Indexes** tab in the CONNX Data Dictionary Manager window, and then click the **Refresh Indexes** button. A message saying the indexes are successfully refreshed appears.
4. Save the CDD by clicking **Save** on the **File** menu.

RMS Text Files

RMS Text File Import Specification

The RMS text file import specification should be used only if your record layouts **are not** COBOL FD, DIBOL, Formatted DDL, Powerhouse PDL, BASIC, or VAX or Alpha CDD format.

If you have many record layouts to import from a non-standard format, it is possible to convert them into the CONNX text file import format. This will require that you write an application to convert your existing record layouts to format specification. If you have only a few small record layouts, it may be faster to manually enter them into a new or existing CONNX Data Dictionary instead of using the text file import.

The text file import specification is described below.

The first line of each record layout should be as follows:

CONNXTABLE, <TableName>, <RMS File Name>,<Record Length>, <SQL View Clause>

Note: Inclusion of a SQL View Clause is optional.

One import text file may contain multiple record layouts, each starting with the same header line shown above.

Each subsequent line in the file represents a column in the record layout. The format for each line is as follows:

<column name>, <column length>, <column offset>, <column type>, <column scale>, <column base>, <column fraction>, <column comment>

RMS Text File Import Syntax and Description

Syntax	Description
<column name>	Name of the column.
<column length>	Length of the column.
<column offset>	Offset of the column.
<column type>	Code for the data type of the column.
<column scale>	Scale of the column (power of 10). A scale of -2 converts 4.3 to .043. A scale of 2 converts 4.3 to 430.
<column base>	Reserved. Must be 0.
<column fraction>	Fraction of the column (negative power of 10). A fraction of -2 converts 4.3 to .043. A fraction of +2 converts 4.3 to 430.
<column comment>	Comments field.

The following is an example of an RMS import file. It includes an optional SQL View Clause. Inserting a view clause limits the result set. This sample view clause returns only rows where the Company field is not blank. The SQL View Clause text box is located on the Table Properties tab in the CONNX Data Dictionary Manager window.

CONNXTABLE, CompanyTable, COMPTABLE.DAT, 64, Company <> ''

Company, 30, 0, 1, 0, 0, 0, This is the Company Field.

Title, 10, 30, 1, 0, 0, 0, This is the Title Field.

Name, 20, 40, 1, 0, 0, 0, This is the Name Field.

Age, 4, 60, 14, 0 ,0 ,2, This is the Age Field.

Important: *When creating an RMS text file, it is recommended that you use the column scale syntax rather than the column fraction syntax.*

Related Topics

To import from an RMS text file import specification

SQL View Clause text box

RMS View Text File Import Specification

The RMS View text file import specification can be used to populate your CDD with predefined CONNX Views using the RMS Text Import Option.

The VIEW text file import specification layout is described below.

The first line of each view layout must contain **CONNXVIEW** and the view object name as follows:

```
CONNXVIEW, <VIEWOBJECTNAME>
```

Subsequent lines contain the SQL Select statement.

```
Select. . .
```

The LAST or footer line in each view must contain:

```
ENDVIEW
```

One import text file may contain multiple views, each starting with the same header line shown above, followed by a **SELECT** statement and a footer line with the word **ENDVIEW**.

The following is an example of an RMS VIEW import file.

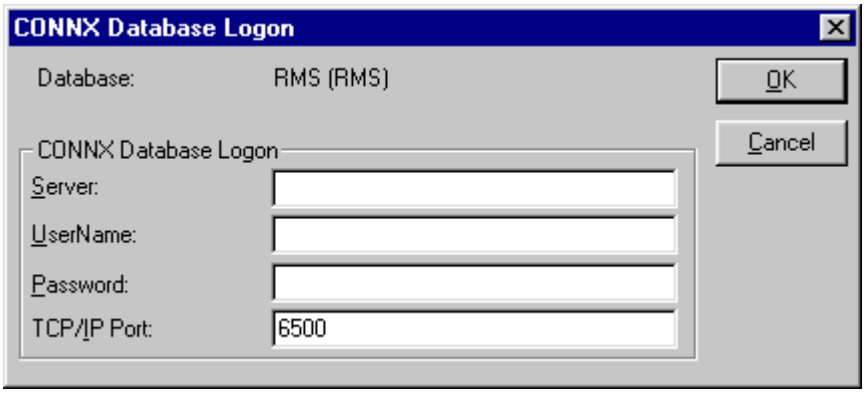
```
CONNXVIEW, NWORDERS
/*This view was requested by Johnathon Jones on 3/1/2001. He executes
   this view daily to see orders for the Northwest Territory. */
SELECT
ORDERS_RMS.orderid as 'Order' /* Order Number */,
ORDERS_RMS.customerid as 'Cust Id' /* Customer Identification */,
CUSTOMERS_RMS.customername as 'Name' /* Name of Customer*/,
CUSTOMERS_RMS.customerstate as 'ST' /* State Ordered by */,
ORDERS_RMS.orderdate as 'Ord Date' /* Date Ordered */,
ORDERS_RMS.productid as 'Product' /* Product number */,
PRODUCTS_RMS.productname as 'Description' /* Product Description */,
ORDERS_RMS.productquantity as 'Qty' /* order quantity */,
PRODUCTS_RMS.productprice as 'Price' /* price per unit */,
(ORDERS_RMS.productquantity * PRODUCTS_RMS.productprice) as 'Ext Price'
   /* Calculate extended price) */
FROM ORDERS_RMS, CUSTOMERS_RMS, PRODUCTS_RMS /* Tables included in view
   */
WHERE ORDERS_RMS.customerid=CUSTOMERS_RMS.customerid AND
ORDERS_RMS.productid=PRODUCTS_RMS.productid and
   CUSTOMERS_RMS.customerstate in ('WA', 'OR', 'MT', 'ID', 'CA') /*
   Join tables together and select only Northwest states */
ENDVIEW
```

To import tables or views from an RMS text file import specification

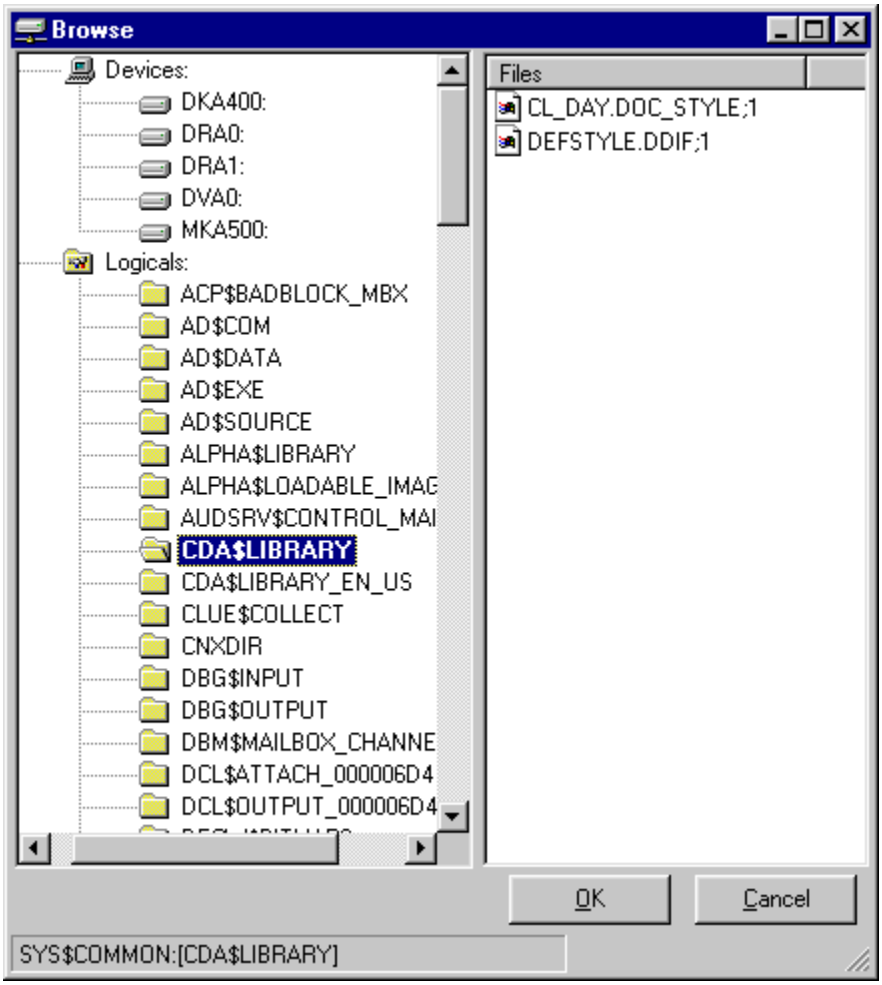
1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **RMS Text Specification File** in the **Import Type** list box.
3. Type the full RMS path for the text file you created, for example, **DKA600:[MYDIR]MYIMPORT.TXT**

   Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

   Note: *Multiple files can be specified in the text box by separating each file name with a comma. The allowable limit is 255 characters.*
4. You can also use the **Browse** button below the text box to locate files to import. If you do not need to use the **Browse** button, proceed to Step 6.
5. Click the **Browse** button. If you are not connected to a VMS server, the **CONNX Database Login** dialog box appears.



1. Type the server name or IP address, a user name, and password in the corresponding text boxes.
2. Port 6500 is listed in the **TCPIP Port** text box by default.
3. Click the **OK** button. The **Browse** dialog box appears.
4. Select the file(s) in the **Browse** dialog box, and then click the **OK** button to return to the **Import CDD** dialog box.



6. Enter the following information in the **Import CDD** dialog box:
    1. Type the server name or IP address, a user name, and password in the corresponding text boxes on the **Logon Information** tab.
    2. Port 6500 is listed in the **TCP/IP Port** text box by default.
    3. Select the destination database from the **Destination Database** list box. See Adding a database connection for more information.
7. Click the **OK** button.

Related Topics

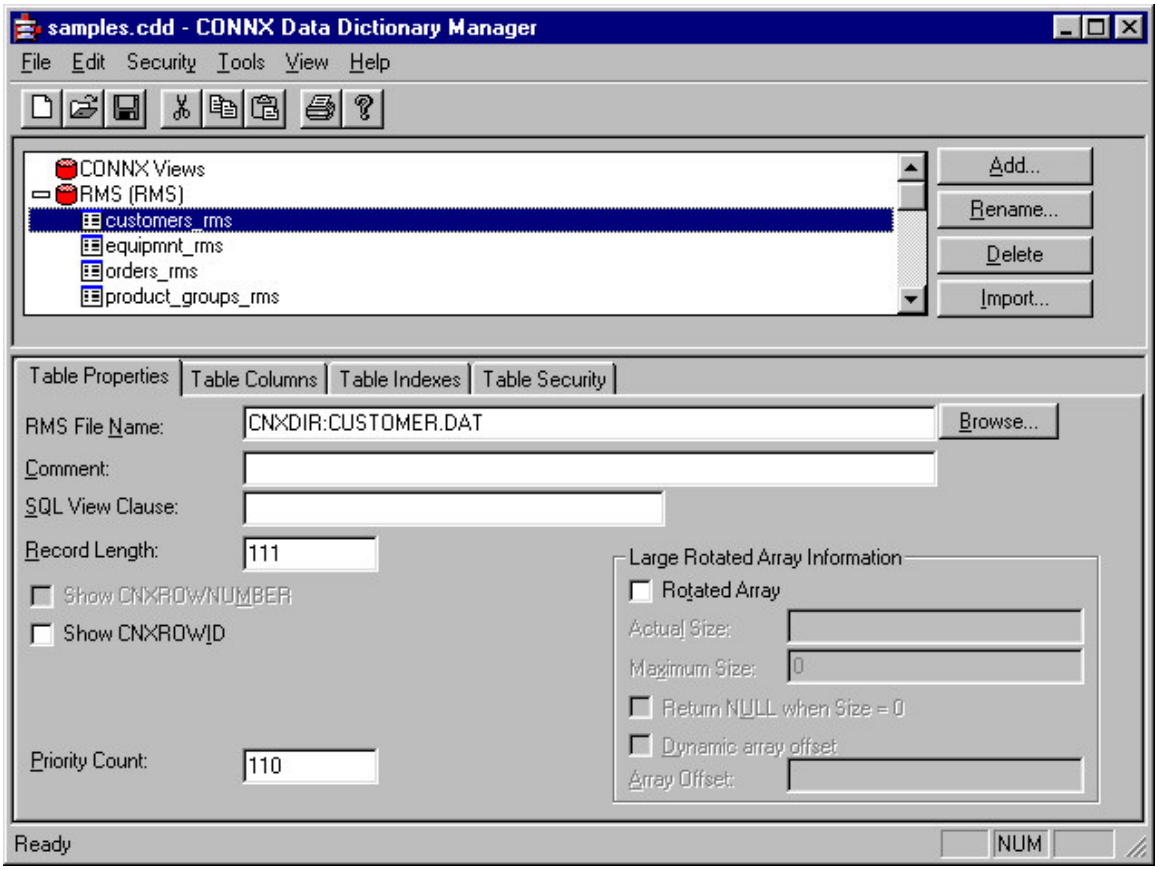
- RMS Text File Import Specification
- CONNX Data Types

Creating CDD Entries Manually

CONNX Data Dictionaries for RMS tables can be created manually. Once a location is established for the type of database files in the CONNX Data Dictionary Manager dialog box, create new entries by using the Add button. Text boxes in the lower CDD pane define table properties and other features of the table, including levels of security. The data file name must be specified on the Table Properties tab in the CONNX Data Dictionary Manager window.

To create an RMS table entry manually (CDD)

1. Click the **Add** button in the CONNX Data Dictionary Manager window.



2. Type the **SQL Object Name.**
1. Must be unique.
2. Valid CONNX table names cannot contain spaces or begin with a number.
3. Maximum length is 50 characters.

3. Select **Table** as the type of object to create in the **Object type** list box. This option determines whether the object is a view or a table definition.

4. Select **RMS** as the type of database in which to create an object in the **Database** list box. The option specifies the type of database in which the table is located. Valid database types for manual entry are RMS and DataFlex.

5. Click the **OK** button.

6. In the **RMS Filename** text box in the CONNX Data Dictionary Manager window, type the physical VMS path for the RMS data file. The maximum length of the filename is 255 characters. The RMS file name may also include VAX logicals. If a logical is used in the file name, it is important that the logical is still defined when logging onto the VMS system as a network process.

7. In the **Comment** text box, type up to 64 characters of descriptive text.

8. In the **SQL View Clause** text box, you may type any valid SQL expression. This text box is used to limit the type of records returned from the table. Maximum length is 128 characters. See SQL View Clause Text Box.

9. Record Length is automatically pulled from the data file.

10. Select the **Show CNXROWID** check box to use the RFA (Record File Address) of its RMS files as the primary key. Use this with RMS files that do not have a unique key.

11. In RMS files, the RFA is represented by the pseudo-field CNXROWID.

12. Under **Large Rotated Array Information**, select **Rotated Array** to convert columnar arrays into rows. (See the section on Rotated Arrays.)

13. The **Priority Count** text box displays a comparison of the relative size of the tables used in join optimization. The numbers may not match the actual number of records in the RMS file. This value is automatically pulled from the data file.

Note: *If a reserved keyword is used as a field or table name in a database object, CONNX automatically appends the name with the characters _col.*

Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

Note: *Any RMS text import file can contain both views and tables to be imported. They do not need to be stored in separate files.*

Related Topics

Reserved Keywords and Symbols

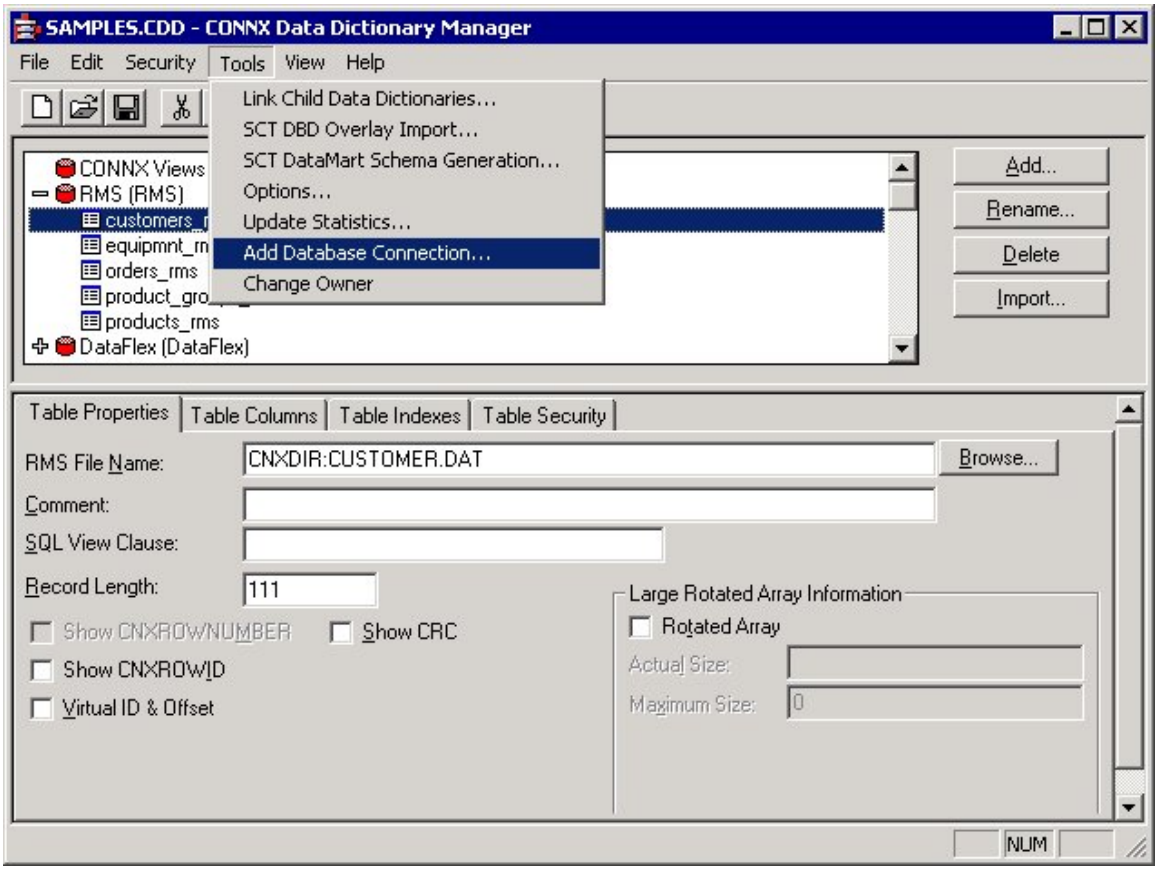
SQL View Clause Text Box

Creating RMS Databases Manually

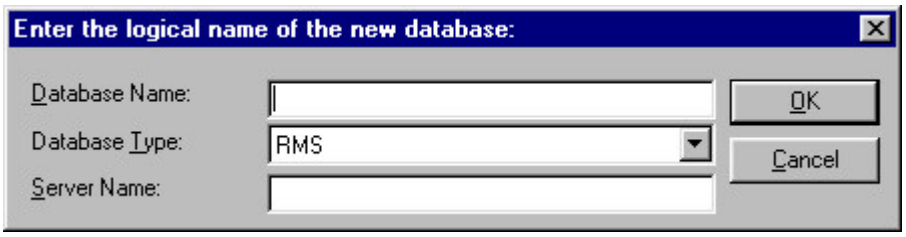
In CONNX, an RMS database represents a collection of RMS files on a given server. With the Add Database Connection feature, you can manually create new database connections to VMS servers on a given port. After the database has been created, you can use the CONNX import to add tables to the database, or you can manually create new table entries.

To build an RMS database manually

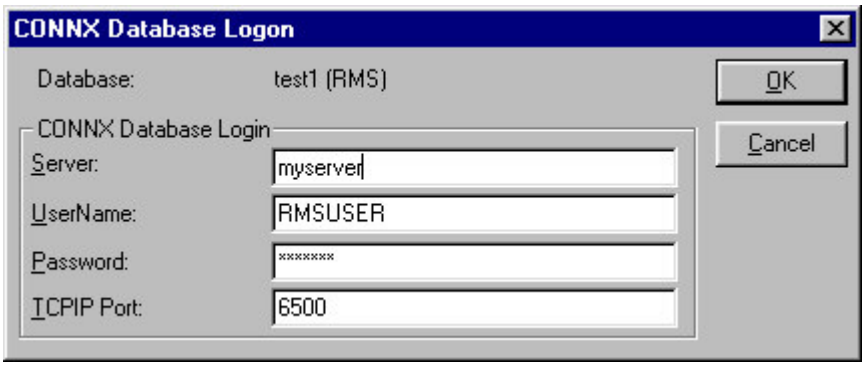
1. Select **Add Database Connection** on the **Tools** menu in the CONNX Data Dictionary Manager window.



2. The **Enter the logical name of the new database:** dialog box appears.



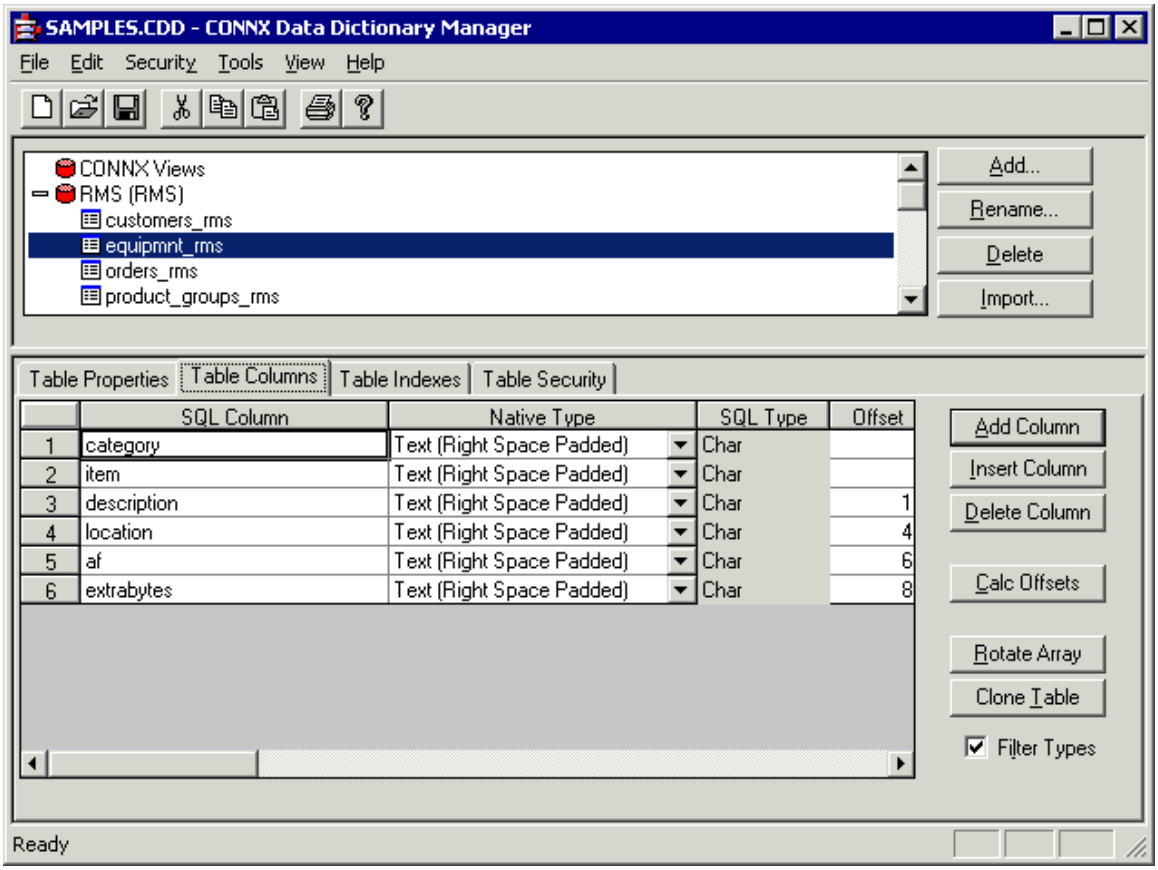
3. Type a name for the database in the **Database Name** text box.
4. Select **RMS** as the type of database to create in the **Database Type** text box.
5. Type the name or IP address of the server in which you intend to store the database in the **Server** text box. Click the **OK** button.
6. The **CONNX Database Logon** dialog box appears with the server name and TCPIP port number entered as defaults.



7. Type an RMS user name in the **User Name** text box.
8. Type an RMS password in the **Password** text box.
9. Click the **OK** button.
10. The new database is added to the list of available databases in the CONNX Data Dictionary Manager window. Each RMS database listed in the CONNX Data Dictionary Manager window can be associated with a different server.

To add columns to an RMS file

1. Select the table to which columns will be added in the CONNX Data Dictionary Manager window.
2. Click the **Table Columns** tab.



3. Click the **Add Column** button. The cursor moves to a new row in the table.
4. Under **SQL Column,** type the name of the new column.

1. A valid column name must be unique within the table.
2. The CONNX column name may be different from the SQL column name. Rename columns or use alias column names.
3. Valid column names cannot contain spaces or begin with numbers.
4. Maximum length is 30 characters.

5. Under **Native Type,** select a data type from the list box. CONNX determines the SQL data type.
6. Click the **Calc Offsets** button to automatically enter the offset value in the **Offsets** column. Do not use the Calc Offsets button if your record contains redefined fields.
7. The remaining five fields display statistics related to the data: Length, Precision, Scale, Array Offset, and Comment. The data in these fields can be modified, depending on the data type. A description of each field is included in the following table:

Field	Definition
Length	Physical length in bytes of the column.
Precision	Number of implied decimal places.

Scale	Number of places to move the decimal point in a numeric field. A scale of +2 will convert the number 345.67 to 3.4567, and a scale of -2 will convert the number 345.67 to 34567.
Array Offset	Used to determine the size of one element of an array when used with the Rotated Array Option. Refer to "Using the Rotated Array Assistant (RMS and VSAM only)" on page 3-18 of the CONNX Security and Administration Guide for more information.
Comment	Used to provide up to 64 characters of descriptive text.

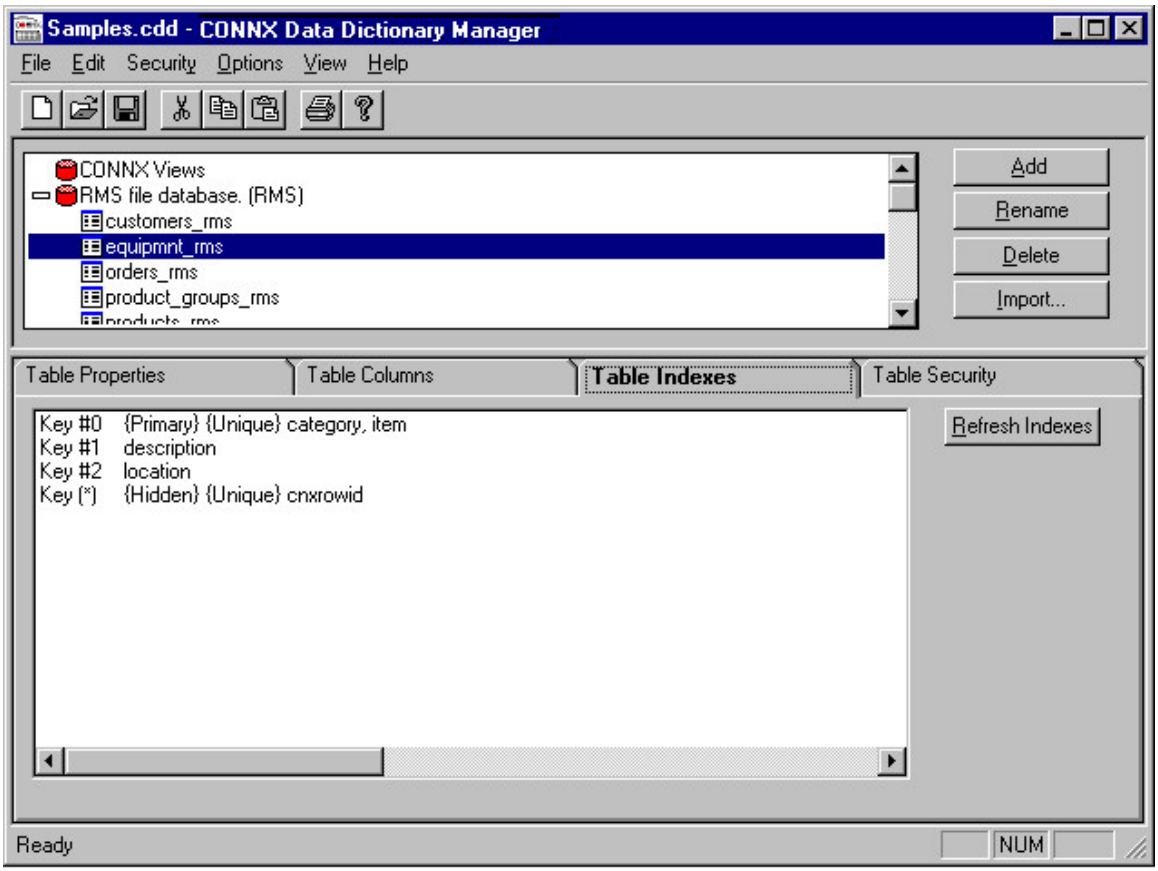
8. Click the **Tables Indexes** tab, and then click the **Refresh Indexes** button to refresh the indexes defined for the .dat RMS file.

Note: *If you do not have an RMS database created for your server, a database is automatically created when you import. The new container has the same name as your server. If you used the server IP address in the Server text box, the name of the new container appears as "RMS_" followed by the IP address.*

To view an index

The Table Indexes tab displays the indexes in the table. CONNX displays this list automatically. To view the indexes:

1. Click the **Table Indexes** tab in the CONNX Data Dictionary Manager window. (CONNX maintains its indexes automatically.)



2. Click the **Refresh Indexes** button to refresh.

DataFlex Tables

Importing DataFlex Table Definitions

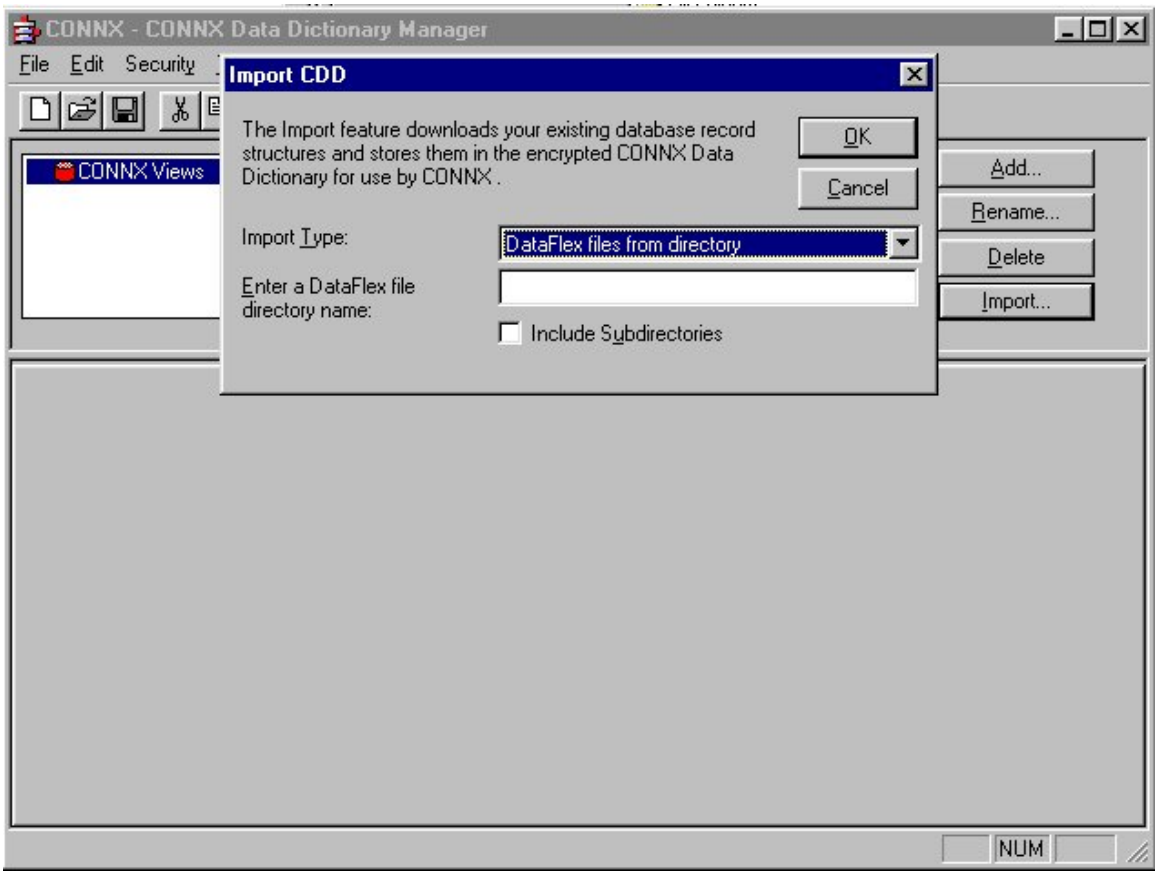
The CONNX server for DataFlex is a full-featured data server that translates SQL requests into native database requests. The ODBC driver makes the server transparent to the end user.

Related Topics

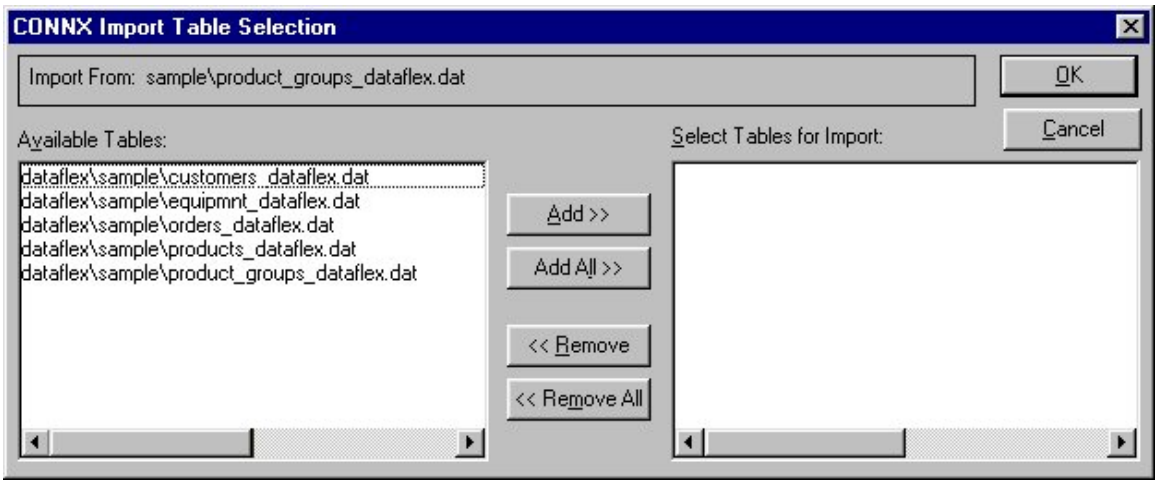

To import existing DataFlex table definitions

To import existing DataFlex table definitions

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **DataFlex files from directory** from the Import Type list box.
3. Type a DataFlex file directory name in the **Enter a DataFlex file directory name** text box. Specify the full path for the directory, local or network.



4. Select the **Include Subdirectories** check box to display the DataFlex subdirectories.
5. All of the record layouts in the specified DataFlex file are imported. No additional logon information is required.
6. Click the **OK** button. The **CONNX Import Table Selection** dialog box appears with a list of available table names. Click the **Add** or **Add All** button to move the tables to the **Select Tables for Import** pane.



7. Click the **OK** button to import the selected tables into CONNX. The DataFlex database tables are added to the list of accessible objects in the CONNX Data Dictionary Manager window.

   Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

Related Topics

- Importing DataFlex Table Definitions
- DataFlex Data Types

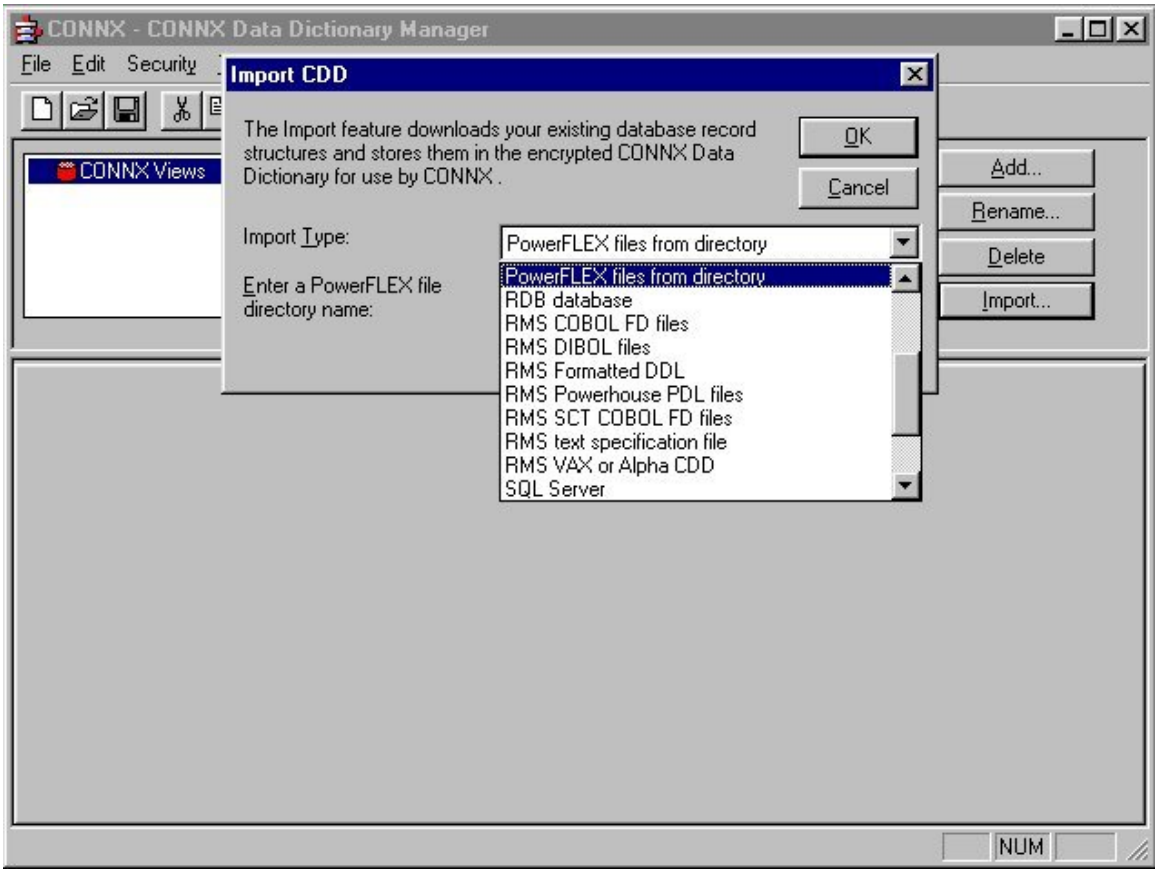
POWERflex Tables

Importing POWERflex Table Definitions

The CONNX server for POWERflex is a full-featured data server that translates SQL requests into native database requests. The ODBC driver makes the server transparent to the end user.

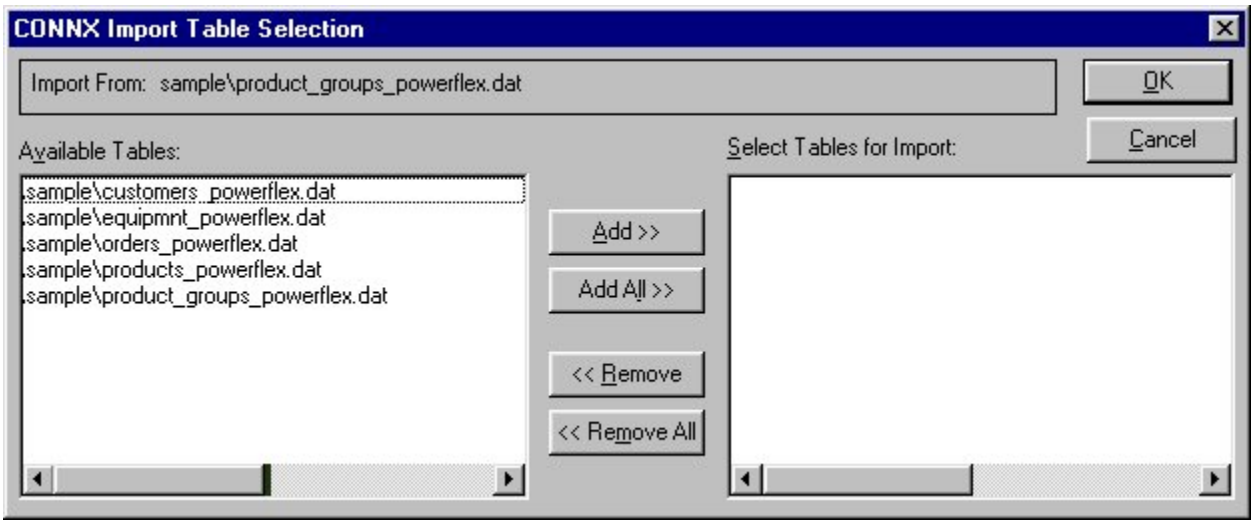
To import existing POWERflex table definitions

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **POWER*flex* files from directory** from the **Import Type** list box.



3. Type a POWERflex file directory name in the **Enter a POWER*flex* file directory name** text box. Specify the full path for the directory, local or network.
4. Select the **Include Subdirectories** check box to display the POWER*flex* subdirectories.
1. All of the record layouts in the specified POWERflex file are imported.
2. No additional logon information is required.

5. Click the **OK** button. The **CONNX Import Table Selection** dialog box appears with a list of available table names. Click the **Add** or **Add All** button to move the tables to the **Select Tables for Import** pane.



6. Click the **OK** button to import the selected tables into CONNX. The POWERflex database tables are added to the list of accessible objects in the CONNX Data Dictionary Manager window.

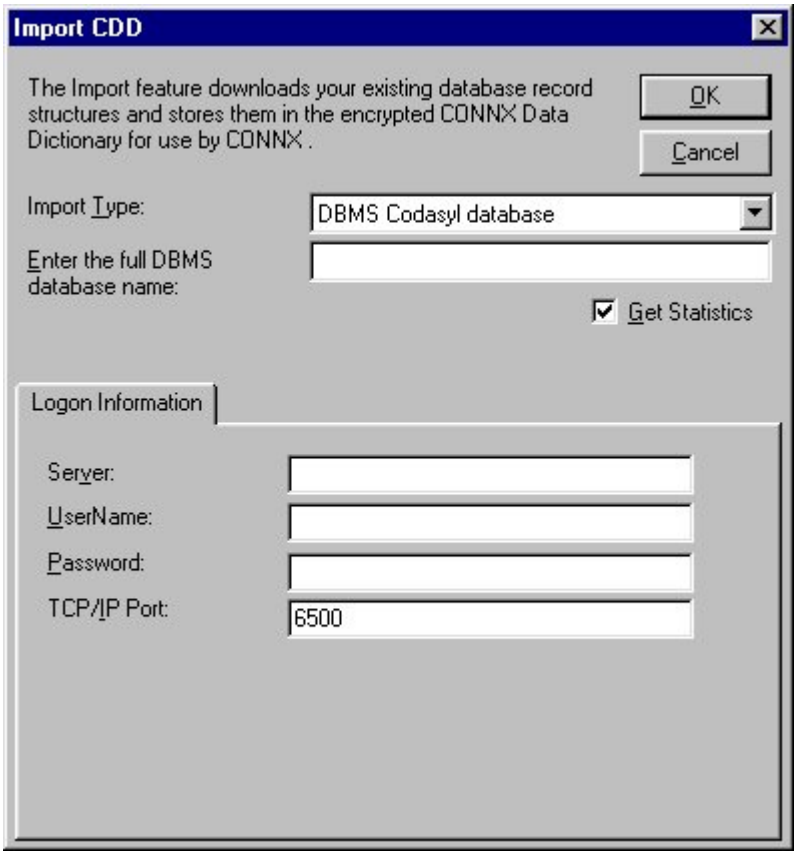
Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

Codasyl DBMS Tables

To import existing Codasyl DBMS tables

When importing DBMS database tables, users can access and transfer DBMS data with all of the data that CONNX can access through a single driver.

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The Import CDD dialog box appears. Select **DBMS Codasyl database** from the **Import Type** list box.

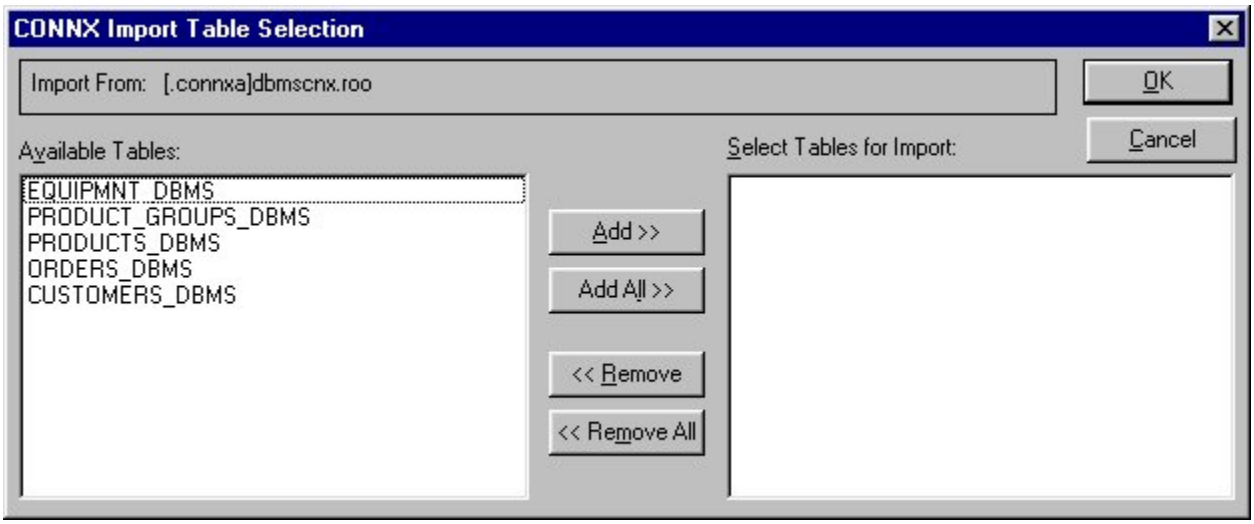


3. Type the DBMS database name in the **Enter the full DBMS database name** text box.
4. Specify the full OpenVMS path for the database. Include the .roo file extension, for example, **DKA600:[MYDIR]PRODUCTS.ROO**

   If you are importing subschemas, the OpenVMS path for the database can include the name of the subschema, for example, **DKA600:[MYDIR]PRODUCTS.ROO SUBSCHEMA** The path may include VMS logicals.
5. Type a server location, user name, and password under **Logon Information.**

1. **Server:** Name of the VMS system running CONNX for DBMS.
2. **UserName:** VMS user name used to log in to the account where the database resides.
3. **Password:** VMS password used to log in to the account where the database resides.
4. **TCP/IP Port:** Set to 6500 by default but can be changed for any current transaction.

6. Click the **OK** button. The **CONNX Import Table Selection** dialog box appears with a list of available table names. Click the **Add** or **Add All** button to move the tables to the **Select Tables for Import** pane.



7. Click the **OK** button to import the selected tables into CONNX. The DBMS database tables are added to the list of accessible objects in the CONNX Data Dictionary Manager window. The Codasyl DBMS database tables are added to the list of accessible objects in the CONNX Data Dictionary Manager window.

***Note**: If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

***Important**: If you intend to import subschemas, see To include subschemas with Codasyl DBMS imports.*

Related Topic

» DBMS Data Types

To include subschemas with Codasyl DBMS table imports

DBMS databases may include a default subschema. A subschema represents a particular subset of a database. Use the following command to determine if subschemas, other than the default subschema, are necessary or exist in the database you intend to import:

```
dbo/dump/subschemas/out=<name of output file> <name of DBMS root file>
   <subschema name>
```

Example:

```
$dbo/dump/subschemas/out=outputfile.out rootfile.roo subschema_name
```

Note that you may have to include the complete path for the root file.

After executing this command, you can edit the outputfile.out file and do a search on the subschema. If other subschemas exist beside the default subschema, they should be listed along with the sets that make up the view.

Contact your database administrator for more information on subschema names and locations.

CONNX DB2 Module

With CONNX, users can access and manipulate data and metadata stored in DB2 system catalogs. This information is used by such front-end applications as Microsoft Access, Visual Basic, PowerBuilder, and Impromptu. Some of the possible implementations for this functionality include data migration, data warehousing, and application development.

CONNX supports dynamic SQL access to DB2 hosts as well as pre-bound static SQL statements accessible through the ODBC API. This approach provides enhanced security options and improved performance for production applications, and is completely transparent to the end user or developer.

Use of the ODBC API also enables invocation of stored procedures created with compiled or interpreted host code, such as C/C++, Java, RPG, Cobol, PL/I, or Assembler.

CONNX can use either TCP/IP or SNA LU 6.2 to access DB2 databases. Importing procedures for both TCP/IP and SNA (Systems Network Architecture) users are described in this section.

Related Topics

- Import and Connect-Time Security Requirements
- How the CONNX DB2 Module Maps ODBC to DRDA Isolation Levels
- Dynamic SQL Package Security
- DB2 for z/OS and DB2 for MVS CDD Import Security Requirements
- DB2 UDB for Windows and Linux CDD Import Security Requirements
- To import existing DB2 tables, views, and stored procedures using TCP/IP
- To import existing DB2 tables, views, and stored procedures using SNA protocol

Import and Connect-Time Security Requirements

Security must be established before connection can be made to any of the DB2 databases used by CONNX.

DB2/400 CDD Import Security Requirements: Access to QSYS2 metadata

During CONNX import procedures, your user ID requires read-only (SELECT) access to the following metadata views:

QSYS2.SYSCOLUMNS
QSYS2.SYSTABLES
QSYS2.SYSPROCS
QSYS2.SYSINDEXES
QSYS2.SYSKEYS
QSYS2.SYSPACKAGE
QSYS2.SYSPARMS

How the CONNX DB2 Module Maps ODBC to DRDA Isolation Levels

CONNX enables the configuration of the initial isolation level for an ODBC connection via the CONNX import utility. The isolation level can be changed programmatically after connecting to the data source via the ODBC 2.x/3.x SQLSetConnectOption/SQLSetConnectAttr APIs. These API functions are called automatically by the higher-level wrapper functions implemented by ADO, RDO, and the MS Access/Visual Basic Jet Dynaset Engine.

Selecting the correct initial isolation level for the CDD depends upon the requirements of the ODBC applications which connect to it. As isolation level increases, concurrent access to shared data decreases, and vice versa. Exclusively read-only ODBC applications, such as report writers, require the lowest isolation level and the highest level of concurrent access.

Online Transaction Processing (OLTP) applications require high isolation levels and restricted concurrent data access. The ODBC 3.x specification defines transaction isolation levels by the presence/absence of key phenomena.

There are three types of key phenomena: dirty reads, nonrepeatable reads, and phantoms:

A **dirty read** occurs when a transaction reads data that has not been committed.

A **nonrepeatable read** occurs when a transaction reads the same row twice, but gets different data for each row.

Phantoms are rows that match search criteria, but are not initially seen, so that different rows are generated for the same criteria if a query is re-executed during the course of a transaction.

The following table defines the four ODBC transaction isolation levels, as defined by SQL-92. An "X" marks each possible phenomenon.

ODBC Transaction Isolation Levels

ODBC Transaction Isolation Level	Dirty reads	Nonrepeatable reads	Phantoms
Read Uncommitted	X	X	X
Read Committed		X	X
Repeatable Read			X
Serializable			

For more information on ODBC isolation levels, refer to the *Microsoft ODBC 3.0 Programmer's Reference, Volume 1, and SDK Guide,* ISBN 1-57231-516-4, published in 1997 by Microsoft Press.

CONNX implements ODBC transaction isolation levels by mapping them to the analogous DB2 and DRDA isolation levels. The No Commit isolation level is implemented only on DB2/400 targets. Note that the No Commit isolation level is the default for DB2/400 CDD/data sources.

The following table defines the three types of transaction isolation levels:

Three Types of Transaction Isolation Levels

Transaction Isolation Level		
ODBC	**DB2**	**DRDA**
Read Uncommitted	Uncommitted Read (DB2 UDB and mainframe targets [OS/390 and MVS])	CHG = Change
Read Committed	Cursor Stability	CS = Cursor Stability
Repeatable Read	Read Stability	ALL = All

Serializable	Repeatable Read	RR = Repeatable Read
Read Uncommitted	No Commit (DB2/400 targets)	NC = No Commit

Note: *A simultaneous ODBC API trace will expedite problem diagnosis. You can start the ODBC API trace via the Windows ODBC Administrator control panel applet.*

Related Topics

CONNX DB2 Dynamic SQL Packages

To establish CONNX and DB2 CDD configuration options

DB2 OS/400

Use of OS/400 Remote Commands

If your OS/400 user ID is authorized to invoke remote commands, CONNX uses its OS/400 remote command functionality to display index, key, and column/field definitions into temporary files in the QTEMP scratch pad. The output from the QTEMP temporary tables is then SELECTed via dynamic SQL, and returned to the CONNX CDD import utility. The following OS/400 CL commands are used:

DSPDBR (Display Data Base Relations)

DSPFD (Display File Description)

DSPFFD (Display File Field Description)

If your User ID is not authorized to use remote commands, index, key, and column/field definitions are imported from the QSYS2. metadata views.

Related Topic

AS/400 Plug-n-Play Mode

Dynamic SQL Package Security

Dynamic SQL Package Security

To build the dynamic SQL packages required by CONNX, your OS/400 user ID requires write access to the target library/collection (default = NULLID), plus the BIND and GRANT privileges. The last step in the package-building logic is a

```
GRANT execute to library.CONX???? to PUBLIC
```

where

library is the target library/collection (default = NULLID)

???? is the package suffix as documented in CONNX DB2 Dynamic SQL Packages.

AS/400 Plug-n-Play Mode

If you select AS/400 Plug-n-Play mode on the Settings tab of the Import dialog box, your user ID does not require the BIND and GRANT privileges. AS/400 Plug-n-Play Mode is the default for all DB2/400 connections.

Related Topics

To import existing DB2 tables, views, and stored procedures using TCP/IP

To import existing DB2 tables, views, and stored procedures using SNA protocol

DB2 and DB2 for MVS

DB2 CDD Import Security Requirements

Access to SYSIBM Metadata

During CONNX import procedures, your user ID requires read-only (SELECT) access to the following system metadata views:

SYSIBM.SYSTABLES
SYSIBM.SYSPROCEDURES
SYSIBM.SYSINDEXES
SYSIBM.SYSKEYS
SYSIBM.SYSPACKAGE
SYSIBM.SYSROUTINES
SYSIBM.SYSPARMS

To retrieve column descriptor information for tables and views, your user ID must be authorized to issue a DYNAMIC PREPARE of the following SQL statement:

```
SELECT * from owner.table
```

where

owner.table is the table/view selected for importing into the CDD.

DB2 UDB for Windows NT 4.0, OS/Warp Server, and Linux

DB2 UDB for Windows and Linux CDD Import Security Requirements

During CONNX import procedures, your user ID requires read-only (SELECT) access to the following system metadata views:

SYSCAT.INDEXES
SYSCAT.PACKAGES
SYSCAT.PROCEDURES
SYSCAT.PROCPARMS
SYSCAT.TABLES

To retrieve column descriptor information for tables and views, your user ID must be authorized to issue a DYNAMIC PREPARE of the following SQL statement:

```
SELECT * from schema.table
```

where

schema.table is the table/view selected for importing into the CDD.

Related Topic

Dynamic SQL Package Security

Dynamic SQL Package Security

To build the dynamic SQL packages required by CONNX and DB2, your UDB user ID requires write access to the target schema (default = NULLID), plus the Create Package and GRANT privileges. The last step in the package-building logic is a

```
GRANT execute to schema.CONX???? to PUBLIC
```

where

schema is the target schema (default = NULLID)

???? is the package suffix as documented in CONNX and DB2 Dynamic SQL Packages. The default administrator ID created during the DB2 UDB 5.x and 6.x installation process (db2admin) has the required authority.

CONNX and DB2 CDD Settings

To import existing DB2 tables, views, and stored procedures using TCP/IP

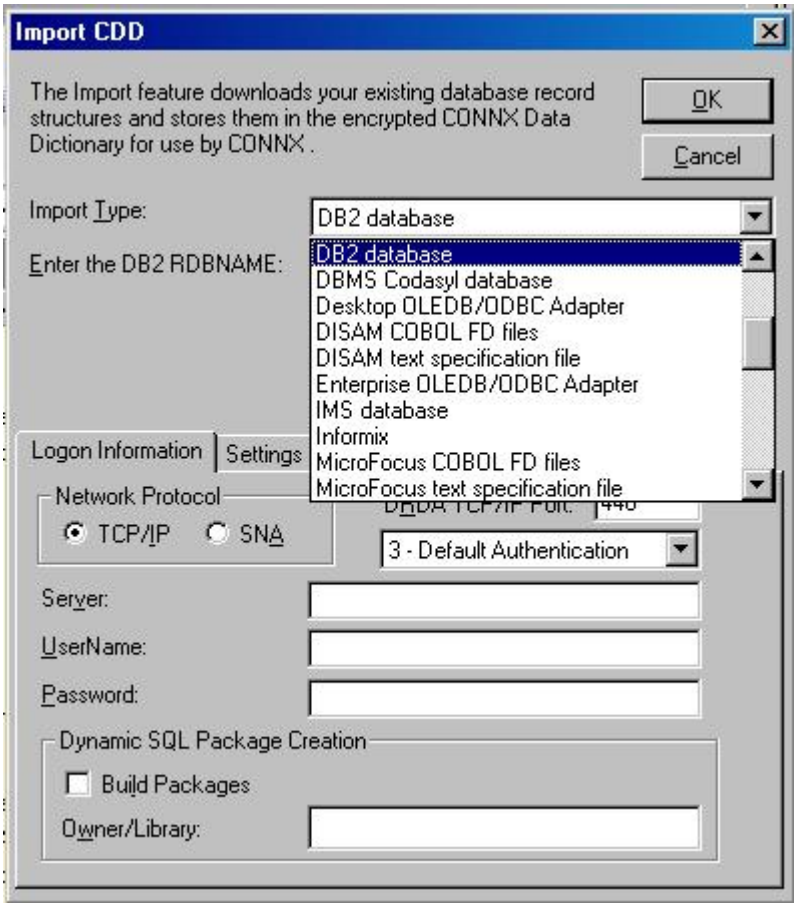
Note: *For DB2 UDB platforms, the RDBNAME = the database name, for example, SAMPLE.*

For OS/400 platforms, the RDBNAME can be displayed via the DSPRDBDIRE command. The network administrator can define the RDBNAME through the ADDRDBDIRE command.

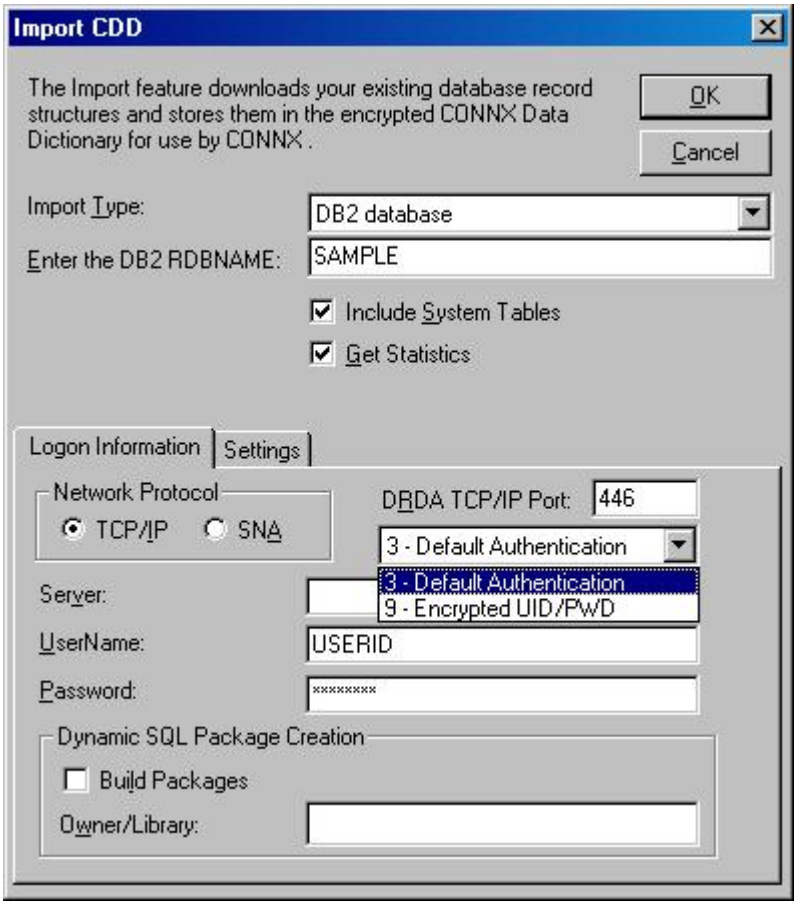
For DB2/MVS or OS/390 platforms, the RDBNAME is the location name. Ask your DB2 administrator for the location name.

Note: *For DRDA target systems supporting TCP/IP, the client must specify a target address. This is an entry field conforming to the four-part TCP/IP address syntax, for example, 102.54.94.97, or a symbolic nickname such as CONNXDB2.*

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **DB2 database** in the **Import Type** list box.



3. Type the RDBNAME (location) in the **Enter the DB2 RDBNAME** text box.



4. If you require access to system metadata, select the **Include System Tables** check box to import system table metadata into the CDD.
5. Select the **Get Statistics** check box to retrieve index and key information and the number of rows per table.
6. Select the **TCP/IP** option on the **Logon Information** tab.
7. From the drop-down box, select either the **Default Authentication** or the **Encrypted UID/PWD** authentication.
8. Type a server location, user name, and password:

1. **Server:** Symbolic or dotted numeric TCP/IP address of DB2 system.
2. **DRDA TCP/IP Port Number:** The DRDA TCP/IP port on which the DB2 target system listens. For mainframe and OS/400 systems, this text box is pre-set to 446. For DB2 UDB systems, enter the port number defined in the host system services file.
3. **UserName:** User name used for logging on to the target host.
4. **Password:** Password used when logging on to the target host.

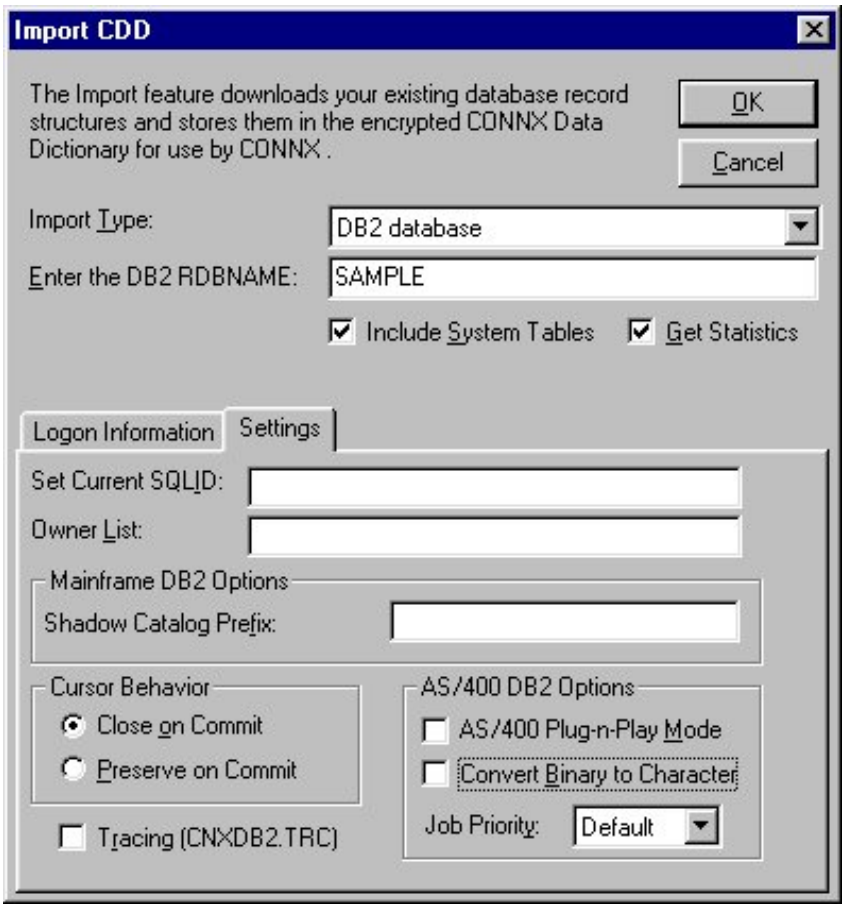
9. Click the **Build Packages** button under **Dynamic SQL Package Creation** to define dynamic SQL packages used by CONNX. This function is a one-time operation, which can be performed by the DB2 administrator on the initial import. All subsequent connect attempts use the pre-built packages, unless the AS/400 Plug-n-Play flag is selected. You must have DB2 administrator authority to build dynamic SQL packages.

If you are connecting to a DB2/400 target, you do not have to click the **Build Packages** check box, since

the AS/400 Plug-n-Play check box on the Settings tab is enabled by default. If you decide to require all users to use the same set of packages, you can do so by clicking the **Build Packages** check box on the initial import.

10. Type the owner, library, or collection name in the **Owner/Library** text box. The option only needs to be selected during the first import session. The default value is NULLID.

11. Select the **Settings** tab in the **Import CDD** dialog box to specify further options.



Refer to the table below for a description of actions to take depending on the selections made on the **Logon Information** and **Settings** tabs.

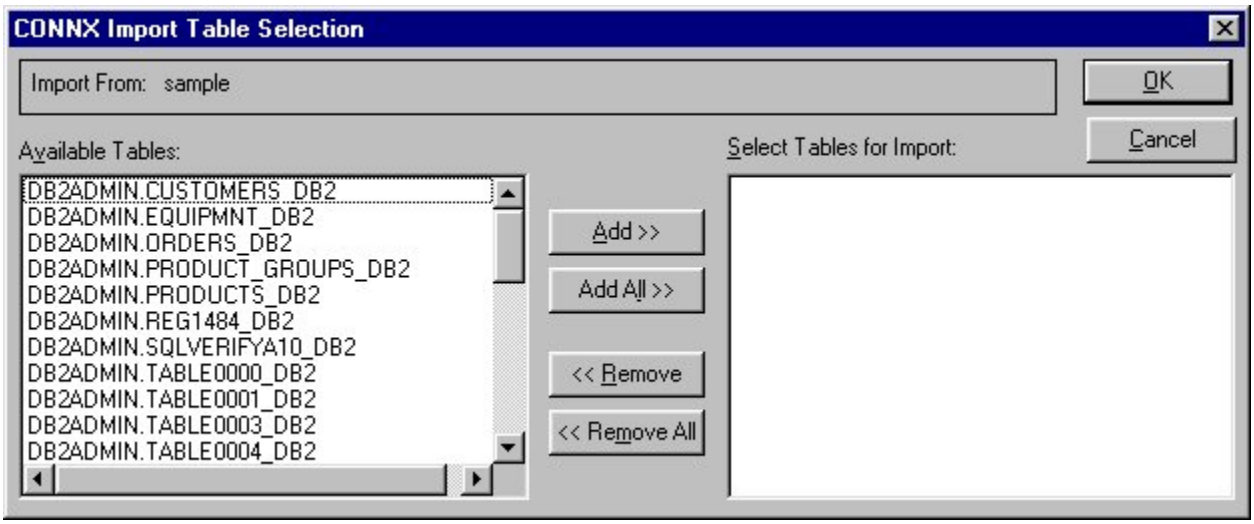
12. To filter the imported tables, views, and stored procedures by owner/schema/collection name, refer to the discussion of the **Owner List** entry field on the **Settings** tab.

13. Select the **Tracing** check box to diagnose potential communications problems. If the box is checked, CONNX writes a binary trace of all sent/received messages via either TCP/IP or APPC/SNA LU 6.2 to the cnxdb2.trc file in the CONNX32 directory. Simultaneously, CONNX writes a text file (cnxdb2.sql) of all executed SQL statements.

The SQL text trace is also written to the same directory/file name, with a file extension of .sql.

Turn the feature off by clearing the check box after successfully tracing problem scenarios and submitting the binary trace file and/or ODBC API trace to CONNX Technical Support.

14. Click the **OK** button. The **CONNX Import Table Selection** dialog box appears with a list of available table names. Click the **Add** or **Add All** button to move the tables to the **Select Tables for Import** pane.



15. Click the **OK** button to import the selected tables into CONNX. The DB2 database tables, views, and stored procedures are added to the list of accessible objects in the CONNX Data Dictionary Manager window.

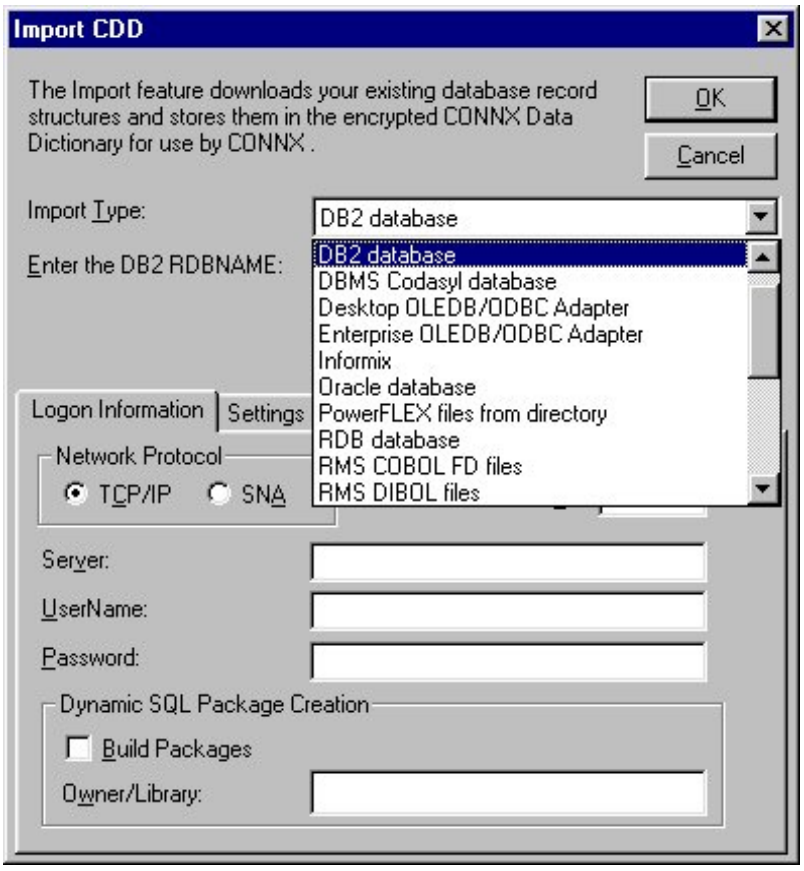
Note:*For DB2 UDB AIX, and Linux, ask your DB2 administrator for the correct port number. To find the correct port number, open a DB2 Command Line Processor window on the target server and type Get DBM CFG. Find the following entry in the output, which is by default:* ***TCP/IP service name (SVCENAME) db2cDB2.*** *Look for the services file in the AIX, Windows, or Linux system directory and find the following entry:* ***db2cDB2 50000/tcp #connection port for the DB2 instance DB2.*** *In this example, the correct port number is 50000.*

AS/400 DB2 Options Settings

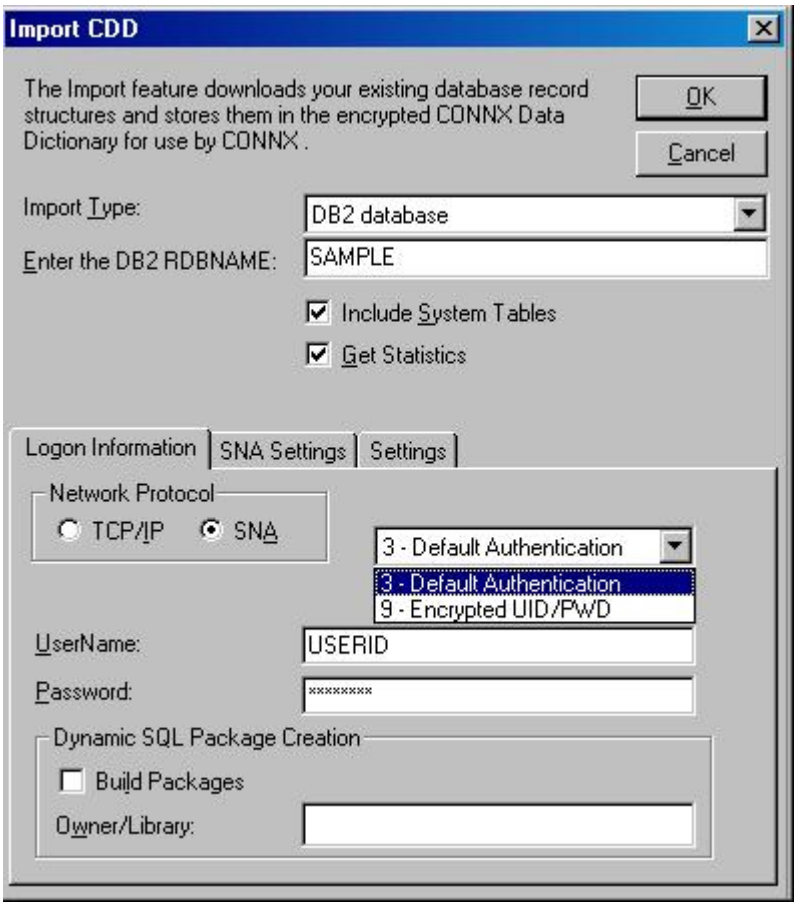
QTEMP = AS/400 temporary scratch pad (per user logon)							
Connected to AS/400	**AS/400 Plug-n-Play ON**	**AS/400 Plug-n-Play OFF**	**Build Packages Option ON**	**Build Packages Option OFF**	**Owner Specified**	**Owner Not Specified**	**Action**
X	X						Always build one package during connect in QTEMP, and build others on demand, as needed, also in QTEMP.
			X			X	Build five packages multiplied by two kinds of cursor behavior multiplied by 1 per set = 10 packages in owner = NULLID. Grant EXECUTE on the packages to PUBLIC.
			X		X		As with Owner Not Specified, but with different owner/library/collection.
X		X		X			Look for prebuilt packages in library/collection NULLID (or other specified library/collection). Error out if packages are not found.
				X			Look for prebuilt packages in library/owner/collection NULLID (or other specified library/owner/schema/collection). Error out if packages are not found.

To import existing DB2 tables, views, and stored procedures using SNA protocol

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **DB2 Database** from the **Import Type** list box.



3. Type the location in the **Enter the DB2 RDBNAME** text box.
4. If you require access to DB2 system metadata, select the optional **Include System Tables** check box to import system table metadata into the CDD.



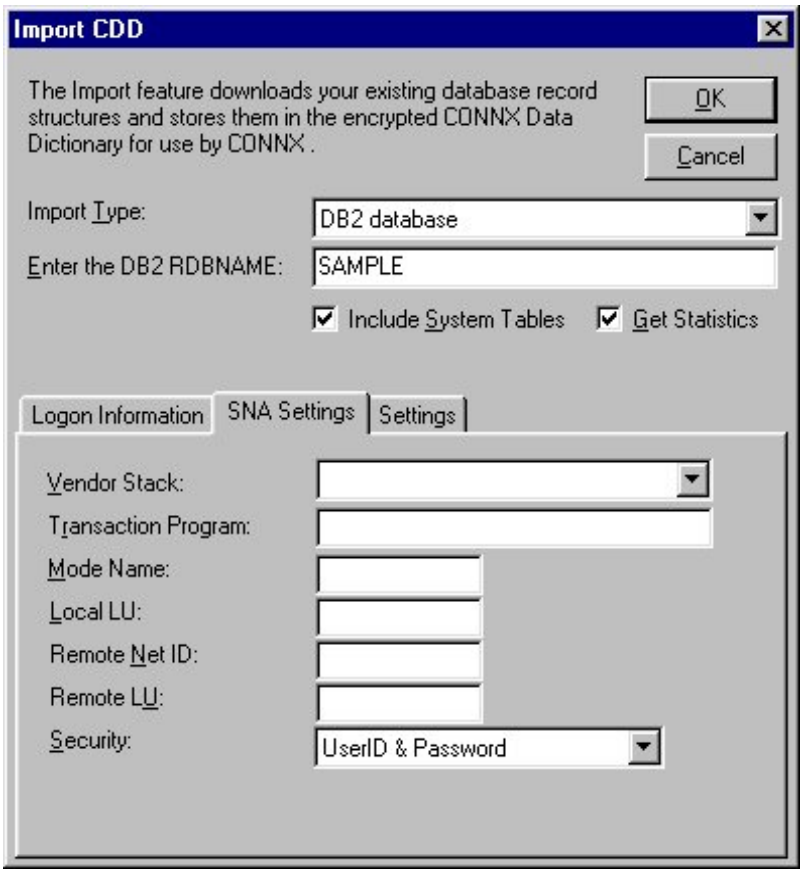
5. Select the **Get Statistics** check box to import keys, indexes, and rows per table.
6. Select the **SNA** option on the **Logon Information** tab.
7. From the drop-down box, select either the **Default Authentication** or the **Encrypted UID/PWD** authentication.
8. Type a server location, user name, and password.
1. UserName: User name used for logging on to the target host.
2. Password: Password used when logging on to the target host.

9. Click the **Build Packages** button under **Dynamic SQL Package Creation** to define dynamic SQL packages used by CONNX. This function is a one-time operation, which can be performed by the DB2 administrator on the initial import. All subsequent connect attempts use the pre-built packages, unless the AS/400 Plug-n-Play flag is selected. You must have DB2 administrator authority to build dynamic SQL packages.

If you are connecting to a DB2/400 target, you do not have to click the **Build Packages** check box, since the AS/400 Plug-n-Play check box on the Settings tab is enabled by default. If you decide to require all users to use the same set of packages, you can do so by clicking the **Build Packages** check box on the initial import.

10. Type the owner, library, or collection name in the **Owner/Library** text box. The option only needs to be selected during the first import session. The default value is NULLID.

11. Select the **SNA Settings** tab. The SNA option enables the configuration of an APPC LU 6.2 connection.



12. Select a **vendor stack** from the list box in the **Vendor Stack** text box and then type the **Mode Name**.

1. **Vendor Stack:** Several vendors implement SNA software, including Attachmate, IBM, Microsoft® , SNA Server, Novell, and NetManage.
2. **Transaction Program:** The host DB2 DRDA "catcher" program. Leave this field blank.
3. **Mode Name:** An Advanced Program-to-Program Conversation (APPC) mode name defines the transmission characteristics of the conversation. IBM has defined a default mode name = IBMRDB, optimized for DRDA traffic. A mode name specifies parameters such as frame size and number of send/receive windows. The mode name must be defined in the selected vendor stack configuration. Maximum length of this field is 8 characters. If a default mode name is defined in the SNA vendor configuration, this field can be left blank.

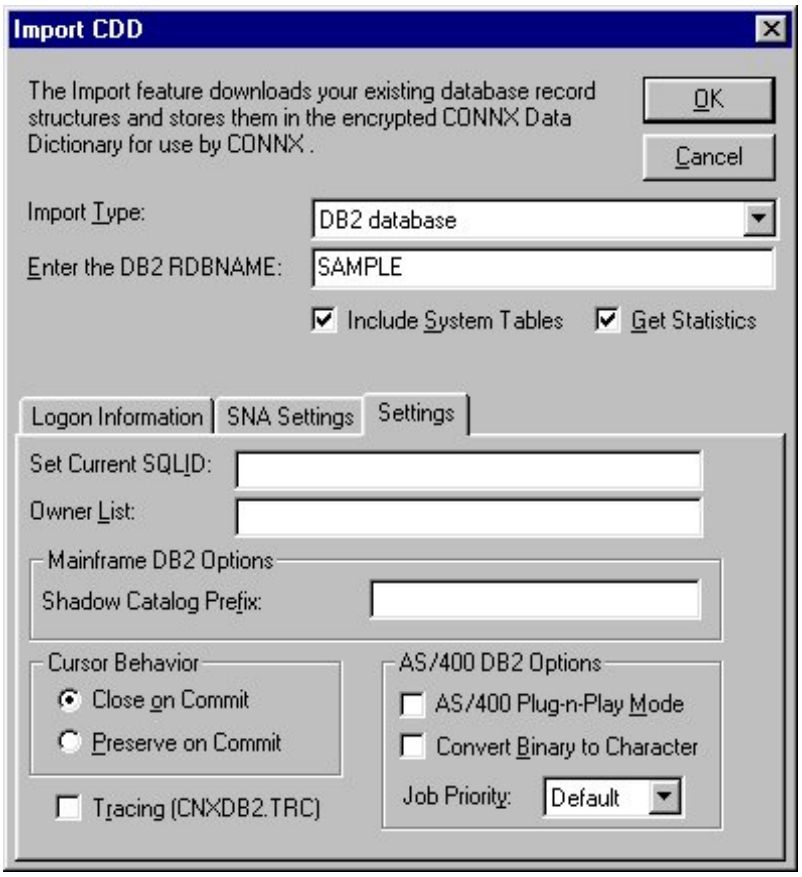
13. Type the **Local LU (logical unit) name,** the **Remote LU name,** and the **Remote Net ID name** (or leave it blank). The Net ID entry fields are for 1- to 8-character network names, such as APPN or IBMIN. These entries must match the names defined in the SNA vendor configuration.

1. **Local LU Alias:** A local LU alias identifies the client PC to the SNA network. This term is used interchangeably with PC system name in AS/400 installations. Local LU has a maximum length of 8 characters. Depending on the SNA vendor-supported feature set, this entry field may be blank if pooled LUs are supported or if a default LU is defined.
2. **Remote Net ID:** If Remote Net ID is left blank, the Remote LU entry field is treated as a Remote LU Alias; a corresponding entry must exist in the SNA vendor configuration. If both a Remote Net ID and a Remote LU Name are entered, a fully qualified netid.luname must be defined in the SNA vendor configuration.
3. **Remote LU:** The Remote LU identifies the target machine for the APPC connection. For AS/400 installations, this term is synonymous with System Name. For DB2 mainframe installations, a

Remote LU uniquely identifies an instance of DDF (Distributed Data Facility), an address space which supports the DRDA AS (Application Server) program. One or more instances of DDF can be active within an SNA network.

4. **Security:** Select the APPC security level from the list box. Choose from UserID & Password, UserID, or None, which correspond to APPC allocate options AP_PGM, AP_SAME, and AP_NONE, respectively. The default is UserID & Password (AP_PGM).

14. Click the **Settings** tab. If you are connecting to a mainframe DB2 target system on which secondary authorization IDs are defined, you can use this entry field to issue a Set Current SQLID command at connect time.



15. Type a schema list in the **Owner List** text box on the **Settings** tab. This list is used to restrict the tables, views, and stored procedures appearing in the initial import list box. The space-delimited list entries can be full library, owner, collection, or schema names, for example, QGPL, QIWS, SYSIBM, QSYS2, SYSCAT, or pattern match entries. The pattern match characters are % and _ for multiple- and single-character pattern matches. Examples are as follows:

1. **QSYS2 A% QGPL:** Lists all tables, views, and stored procedures in QSYS2 and QGPL, and all tables and views for any library/collection/schema/owner beginning with the letter "A."
2. **QSYS2 A_C QGPL:** Lists all tables, views, and stored procedures in QSYS2 and QGPL, and all tables and views for any library/collection/schema/owner that begins with the letter "A" and has the letter "C" as its third and final character.
3. **% or blank:** Lists all tables and views for all libraries/collections/schemas/owners.

16. For DB2 installations which make periodic copies of the SYSIBM.* metadata, enter the copied schema name in the **Shadow Catalog Prefix** text box under **Mainframe DB2 Options.** This directs CONNX to run its import queries against the copied system catalog instead of the SYSIBM.* system metadata.

17. Under **Cursor Behavior,** select the radio button for either **Close on Commit** or **Preserve on Commit.** If you select **Close on Commit,** all dynamic SQL SELECT statements are mapped by CONNX to cursors which close after a commit is issued. If you select **Preserve on Commit,** all dynamic SQL SELECT statements are opened with "sticky" cursors that hold their positions across commits. Preserve on Commit is the preferred setting for Microsoft Access and Visual Basic dynaset processing, but the option may require increased system resources.

18. Under **AS/400 DB2 Options,** the **AS/400 Plug-n-Play Mode** and **Convert Binary to Character** options are checked by default. The **AS/400 Plug-n-Play Mode** option enables each connection to create dynamic SQL packages as needed in the per-connection QTEMP scratch pad. The **Convert Binary to Character** option translates SQL BINARY/VARBINARY, or LONGVARBINARY column data from the host code page to the PC character code page.

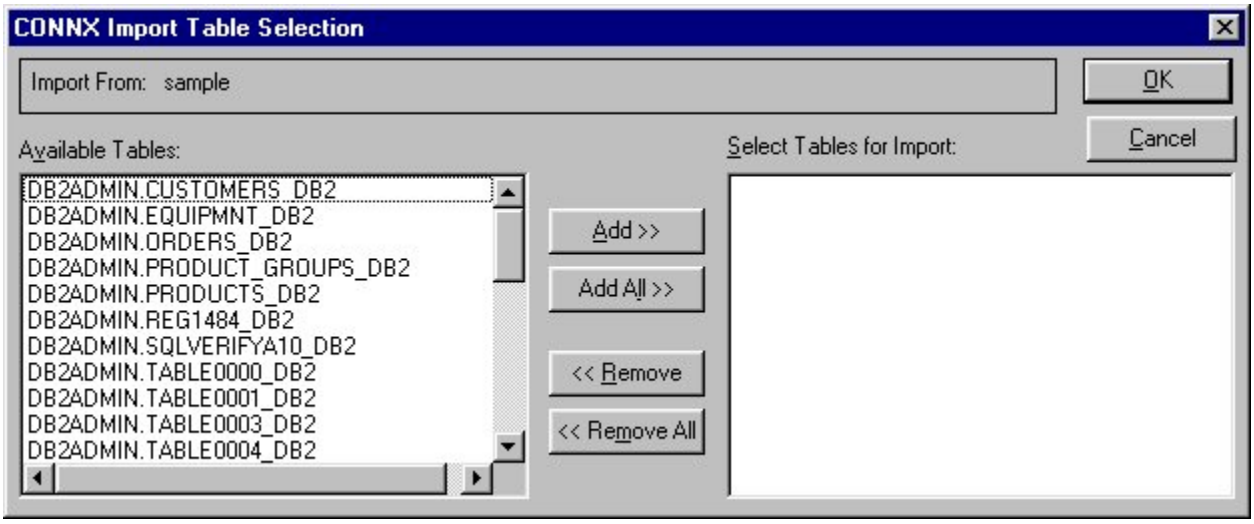
19. Under **Job Priority,** select **Default** to run your SQL requests at the default system job priority. You can also select an alternate job run priority, which will cause CONNX to issue an OS/400 CHGJOB remote command to alter the job run priority, where 1=highest job run priority.

20. Select the **Tracing** check box to diagnose potential communications problems. If the box is checked, CONNX writes a binary trace of all sent/received messages via either TCP/IP or APPC/SNA LU 6.2 to the cnxdb2.trc file in the CONNX32 directory. Simultaneously, CONNX writes a text file (cnxdb2.sql) of all executed SQL statements.

The SQL text trace is also written to the same directory/file name, with a file extension of .sql.

Turn the feature off by clearing the check box after successfully tracing problem scenarios and submitting the binary trace file and/or ODBC API trace to CONNX Technical Support.

21. Click the **OK** button. The **CONNX Import Table Selection** dialog box appears with a list of available table names. Click the **Add** or **Add All** button to move the tables to the **Select Tables for Import** pane.



22. Click the **OK** button to import the selected tables into CONNX. The DB2 database tables, views, and stored procedures are added to the list of accessible objects in the CONNX Data Dictionary Manager window.

Note: *The Preserve on Commit cursor behavior option is not implemented for DB2/OS390 and DB2/MVS.*

CONNX DB2 Dynamic SQL Packages

The Build Packages text box in the CDD import utility instructs the CONNX DB2 ODBC driver to build a user-specified number **N** of package sets for all isolation levels for two types of cursor behavior: Delete/Close (non-sticky) and Open ("hold" or "sticky"). This translates into the following:

N * 5 * 2 = 10 dynamic SQL packages (default for N is 1; maximum is 32)

where

N is the number of identical packages within each set

5 is the number of isolation levels (OS/400 only; 4 for DB2 OS/390, MVS, and UDB)

2 is the number of types of cursor behavior

320 is the maximum number of dynamic SQL packages built by CONNX DB2

Each package consists of 32 sections. The first package for isolation level No Commit with cursor behavior = CLOSE (CONXNC00) defines the following skeleton cursors:

DECLARE NC0001 CURSOR FOR S01

DECLARE NC0002 CURSOR FOR S02

DECLARE NC0003 CURSOR FOR S03

...

DECLARE NC0032 CURSOR FOR S32

Each skeleton cursor is used by the CONNX DB2 ODBC driver to maintain a separate dynamic SQL context, for example:

Select col1 from table maps to cursor 1

Select col1, col2 from table maps to cursor 2

Insert into table (col1,col2) values ('row01', 1) maps to cursor 3

Delete from table where col1 = 'row01' maps to cursor 4

For each isolation level and cursor behavior, 1-32 identical SQL packages can be defined for use by the CONNX DB2 ODBC driver. This provides a theoretical maximum number of unique dynamic SQL contexts = 32 * 32 = 1024 **for each isolation level within a unit of work** for cursor behavior = CLOSE, or **for the life of each ODBC connection** for cursor behavior = OPEN.

For example, an ODBC application could connect to the CONNX DB2 ODBC driver, set AutoCommit mode off, and process any mix of 1024 unique DML (SELECT, INSERT, UPDATE, DELETE) or DDL (CREATE, DROP) SQL statements within a unit of work or during the life of a connection.

For data source (CDD) cursor behavior = CLOSE, when the ODBC application terminates the unit of work via an ODBC 2.x or 3.x SQLTransact/SQLEndTran API function call, the CONNX DB2 ODBC driver issues a COMMIT or ROLLBACK and returns the package section contexts (section numbers and cursor names) to a pool for reuse.

For data source (CDD) cursor behavior = OPEN or PRESERVE ("hold" or "sticky" cursors), when the ODBC application calls SQLTransact/SQLEndTran, the CONNX DB2 ODBC driver issues a COMMIT/ROLLBACK and returns all non-cursor based contexts (INSERT/DELETE/UPDATE, etc.) to the free pool.

For ODBC AutoCommit mode (the default), the CONNX DB2 ODBC driver returns the skeleton cursors to the free pool using a least-frequently used (LFU) algorithm.

In practice, the CONNX DB2 ODBC driver uses only the first package from a package set to process dynamic SQL requests issued by an ODBC application. The choice of current dynamic SQL package set is determined by isolation level and commit mode (AutoCommit or otherwise), as set by the ODBC application (for example, Microsoft Access or Visual Basic) via the ODBC SQLSetConnectOption or SQLSetConnectAttr API functions, and data source cursor behavior (defined in the CONNX DB2 CDD). The extra packages for each package set are defined to support long-running ODBC applications which reuse the same connection, for example, a web server application using an anonymous login such as IUSR_CONNX.

Notes:

1. The dynamic SQL packages and skeleton cursors are managed at the host on a per-connection basis. For instance, any number of CONNX DB2 ODBC client applications can prepare different SQL statements into section 1 of package CONXNC00. The separate SQL contexts for section 1 are managed by the IBM DRDA host program (the Application Server or AS).

2. The user specified number of packages per package set is set to 1 by default. You can adjust the number of packages per package set by adding or changing the following entry in the CONNX.INI file located in your Windows directory.

[DATABASES]

DB2MaxPkgSets=n

Where 1<=n<=32

3. After changing the DB2MaxPkgSets entry, from the same client machine, you can use the CONNX CDD Import application to build **n** CONNX DB2 Dynamic SQL Packages per package set by checking the 'Build Packages' check box.

4. It is not necessary to redistribute the CONNX CDD file used during the import/build process after building additional CONNX DB2 dynamic SQL packages. The CONNX DB2 driver has auto-detect functionality which determines the current number of packages per package set at connect time.

Important: *All CONNX DB2 dynamic SQL package names begin with 'CONX'; hence, the first package built with isolation level = READ_COMMITTED, Access Mode = READ_WRITE, and cursor behavior = del/close is named CONXCS00. The tenth package in this package set is CONXCS09; the thirty-second package in this set is CONXCS0V.*

The dynamic SQL packages created by clicking the 'Build Packages' check box are named as follows:

CONNX DB2 Sample Packages

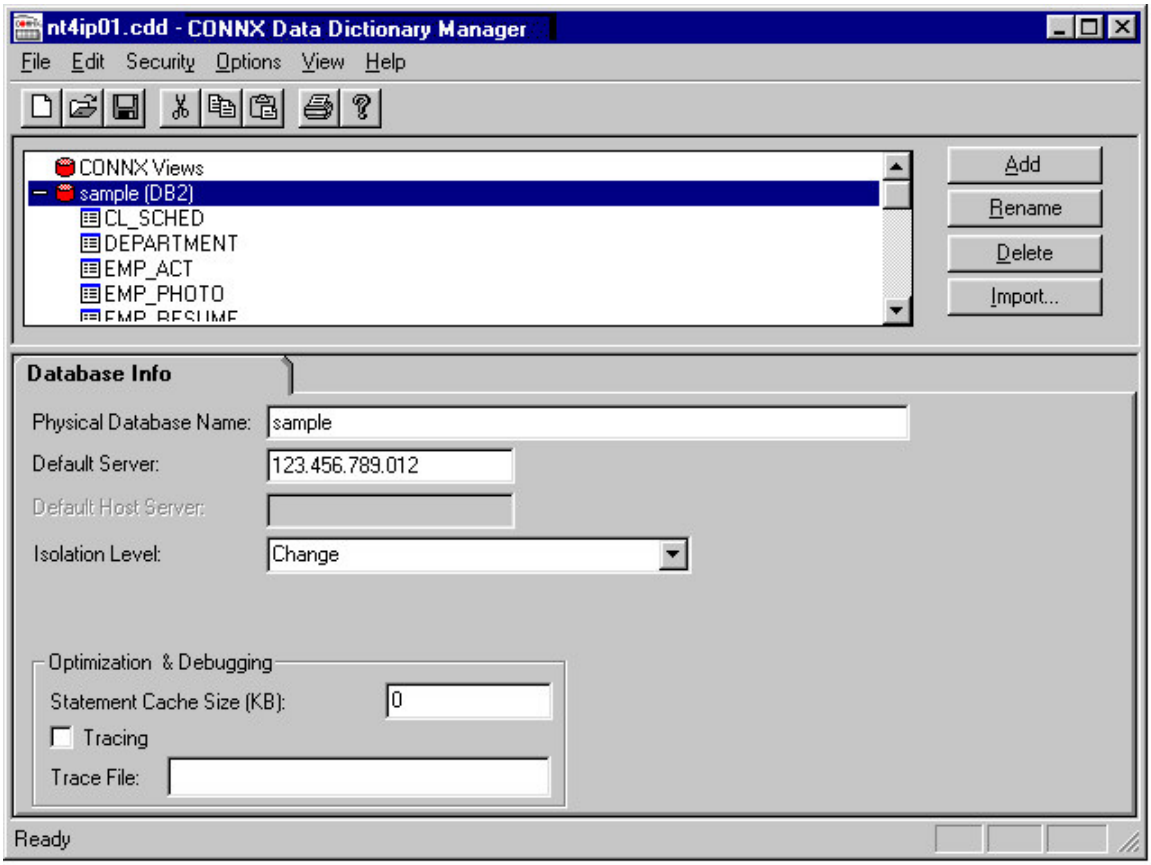
DB2 Platform	Transaction Isolation	Access Mode	Auto Commit	Cursor Behavior	Package Set Suffix
OS/400	READ_UNCOMMITTED	READ_ONLY	ON	DEL/CLOSE	NC0z
				PRESERVE	NC1z
			OFF	DEL/CLOSE	CH0z
				PRESERVE	CH1z
		READ_WRITE	ON	DEL/CLOSE	NC0z
				PRESERVE	NC1z
			OFF	DEL/CLOSE	CH0z
				PRESERVE	CH1z

Non-OS/400	READ_UNCOMMITTED	READ_ONLY	ON	DEL/CLOSE	CH0z
				PRESERVE	CH1z
			OFF	DEL/CLOSE	CH0z
				PRESERVE	CH1z
		READ_WRITE	ON	DEL/CLOSE	CH0z
				PRESERVE	CH1z
			OFF	DEL/CLOSE	CH0z
				PRESERVE	CH1z
All	READ_COMMITTED	READ_ONLY	ON	DEL/CLOSE	CS0z
				PRESERVE	CS1z
			OFF	DEL/CLOSE	CS0z
				PRESERVE	CS1z
		READ_WRITE	ON	DEL/CLOSE	CS0z
				PRESERVE	CS1z
			OFF	DEL/CLOSE	CS0z
				PRESERVE	CS1z
All	REPEATABLE READ	READ_ONLY	ON	DEL/CLOSE	AL0z
				PRESERVE	AL1z
			OFF	DEL/CLOSE	AL0z
				PRESERVE	AL1z
		READ_WRITE	ON	DEL/CLOSE	AL0z
				PRESERVE	AL1z
			OFF	DEL/CLOSE	AL0z
				PRESERVE	AL1z
All	SERIALIZABLE	READ_ONLY	ON	DEL/CLOSE	RR0z
				PRESERVE	RR1z
			OFF	DEL/CLOSE	RR0z
				PRESERVE	RR1z
		READ_WRITE	ON	DEL/CLOSE	RR0z
				PRESERVE	RR1z
			OFF	DEL/CLOSE	RR0z
				PRESERVE	RR1z
Where 0 <= z <= 9 or A <=x<=V					

To establish CONNX and DB2 CDD configuration options

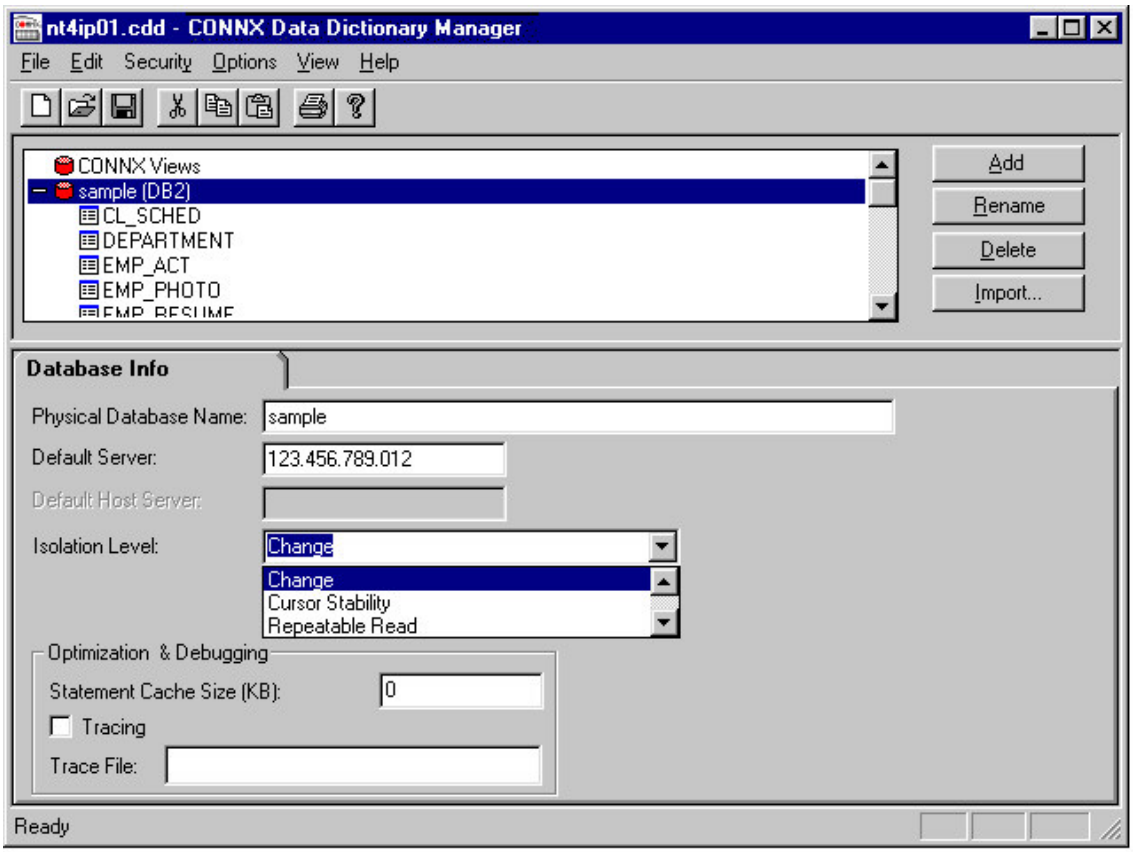
The following procedures may be helpful for troubleshooting connection and configuration options in CONNX.

1. Click on any previously imported DB2 database, e.g. **SAMPLE,** in the CONNX Data Dictionary Manager window.



2. The name of the selected database appears in the **Physical Database Name** text box on the **Database Tab** in the lower pane. The TCP/IP address or SNA location appears in the **Default Server** text box.
3. Select an isolation level in the **Isolation Level** text box.

   Selecting the correct isolation level is very important for optimal performance. Note that as you move down the list, concurrency decreases and exclusivity increases. For instance, if the CDD is to be used by a read-only report writer, the isolation level Change (for mainframe or DB2 UDB targets) or No Commit (for the AS/400) is the appropriate choice, since it maximizes the concurrent usage of DB2 objects. Also note that, in most cases, an online transaction processor requires an All or Repeatable Read isolation level, which locks out updates from other clients until the unit of work is complete.



4. Under **Optimization and Debugging,** enter the **Statement Cache Size,** which specifies the maximum number of kilobytes CONNX can use to store cached information related to the most-frequently executed dynamic SQL statements. CONNX keeps track of the most frequently executed dynamic SQL statements in a virtual memory cache. This cache allows CONNX to optimize the length of messages sent to and received from the target host. Depending on the cursor behavior specified in the CONNX CDD, the cache is refreshed either after each COMMIT/ROLLBACK or whenever the cache exceeds its limit. A value of 0 instructs CONNX to allocate a default cache size of 64 KB. For long-running applications such as Web servers, this value can be revised upward to improve performance. To minimize the cache memory footprint, set the value to 32 (not 0).
5. Select the **Tracing** check box to diagnose potential communications problems. If the box is checked and no **Trace File** name is specified, CONNX writes a binary trace of all sent/received messages via either TCP/IP or APPC/SNA LU 6.2 to the **cnxdb2.trc** file in the CONNX32 directory. Simultaneously, CONNX writes a text file (cnxdb2.sql) of all executed SQL statements.
6. Type a file name in the **Trace File** text box to direct the binary trace to a specified file. The SQL text trace is also written to the same directory/file name, with a file extension of .sql.

   Turn the feature off after successfully tracing problem scenarios and submitting the binary trace file and/or ODBC API trace to CONNX Technical Support.

CONNX and Informix

CONNX Informix Data Module

The CONNX Informix module provides access to Informix data source objects through the Informix ODBC driver.

For multiple connections to Informix via ODBC through a single CONNX Data Dictionary, either the Informix data source name must be a file-based DSN and installed on a shared network server, or the same DSN must be configured on each client computer.

Important: *The selected data source driver must be installed on all client machines that have access to the data source tables since they will be using the same DSN (Data Source Name).*

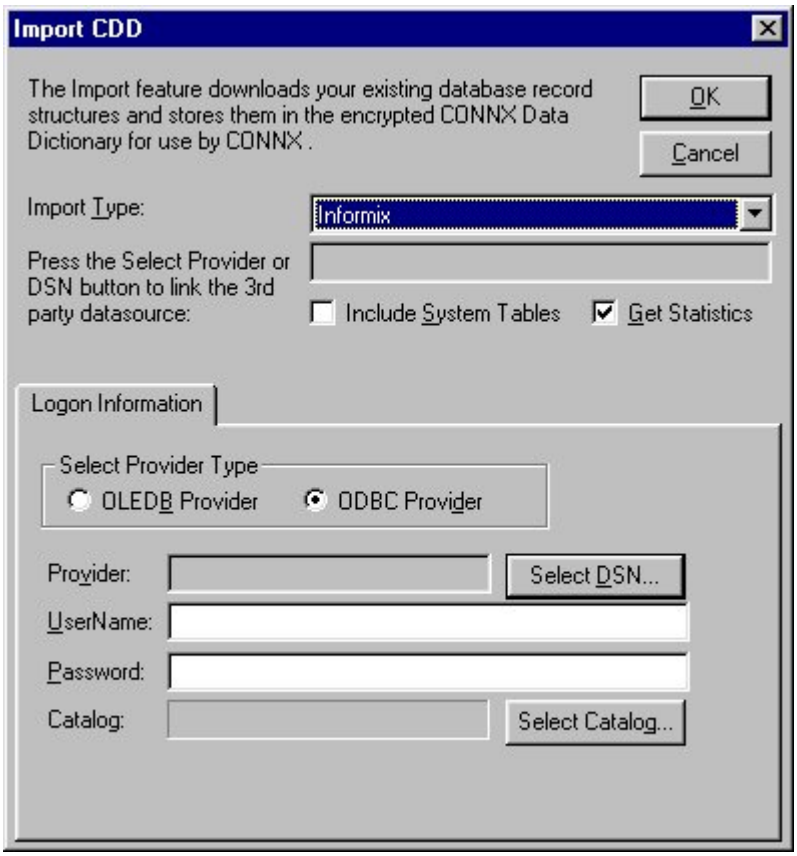
Related Topic

ODBC Data Source Names Used with Multiple Users

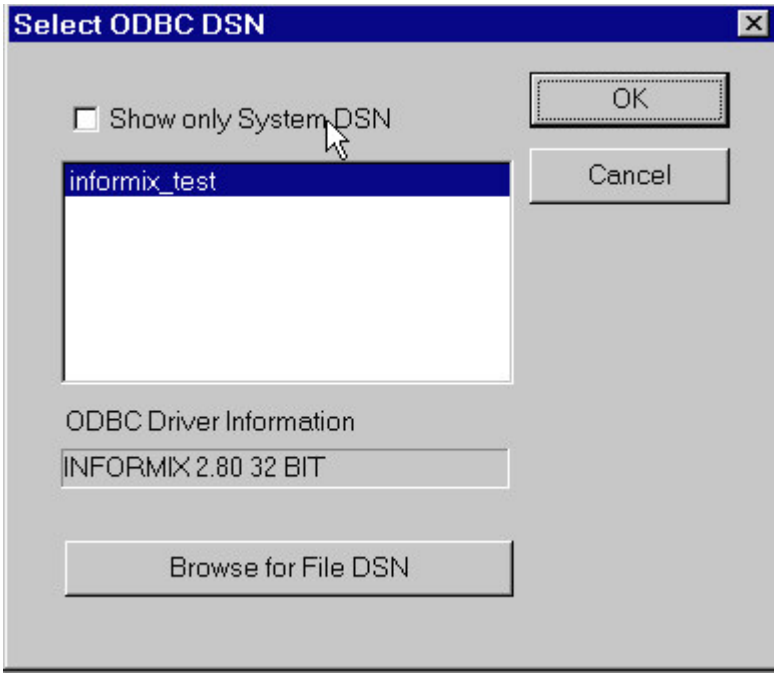
To import tables from an Informix ODBC provider data source

To import tables from an Informix ODBC provider data source

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **Informix** from the **Import Type** list box.



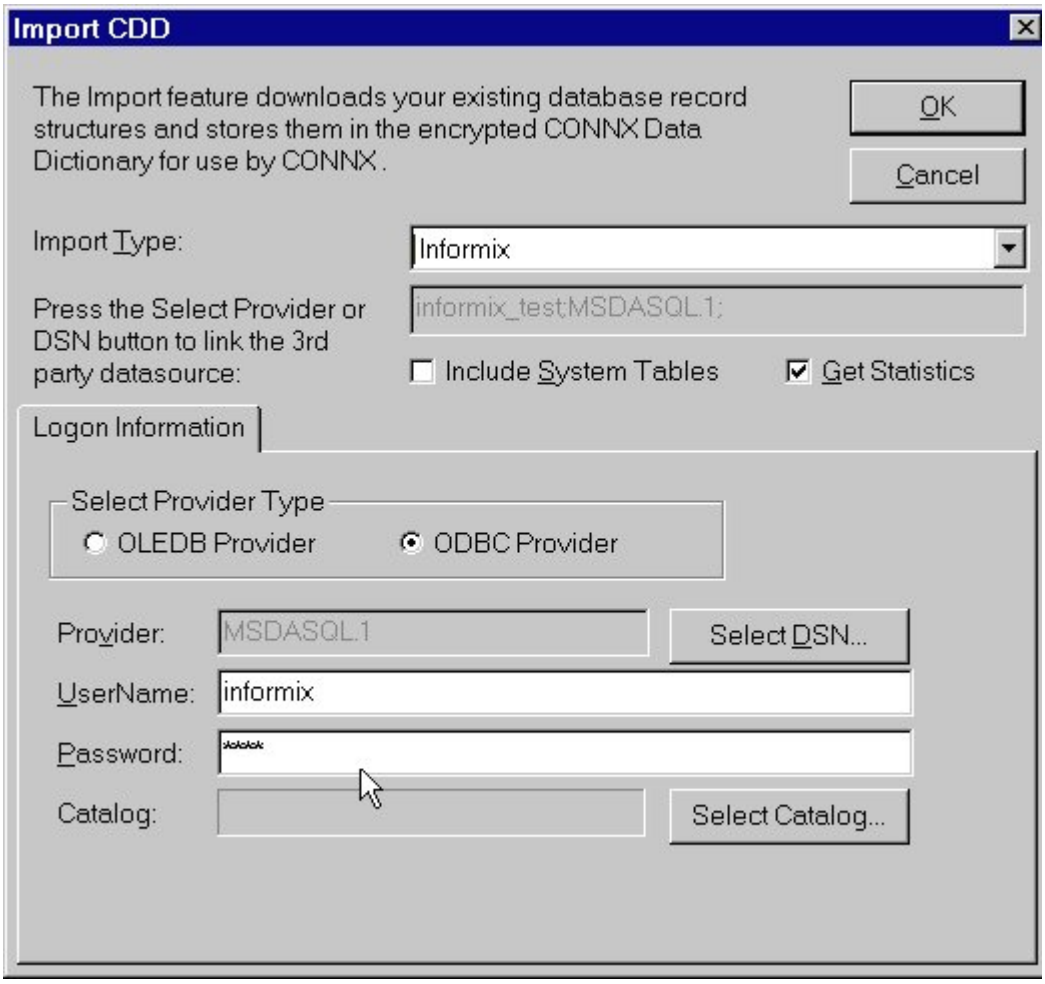
3. Select **ODBC Provider** under **Select Provider Type,** and then click the **Select DSN** button in the **Logon Information** pane.
4. The **Select ODBC DSN** dialog box appears. You can choose either a user-, system-, or file-based DSN, although it is recommended that a file DSN be used, since it can be accessed through a network by multiple users.



5. Your ODBC data source may require additional information. Review the documentation provided by your driver vendor for additional configuration options.

1. **File DSN (Recommended)**
   Select the Browse for File DSN check box to open the Open dialog box. Select a file DSN from lists of available .dsn files.
2. **System DSN or User DSN**
   Select the Show Only System DSN check box to view a list of system DSNs only. Clear the check box to show a list of both user and system DSNs. Note that the ODBC Driver Information text box displays information about the selected ODBC DSN. The same information can be found in the ODBC Data Source Administrator dialog box. Click the **OK** button in the **Select ODBC DSN** dialog box to return to the **Import CDD** dialog box, which displays the selected DSN.

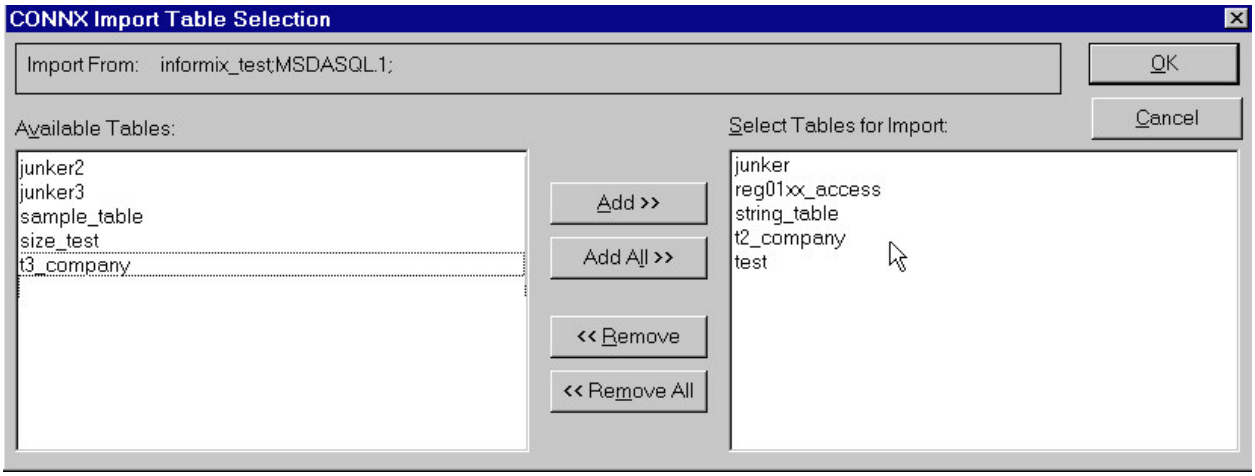
6. The **Import CDD** dialog box appears.



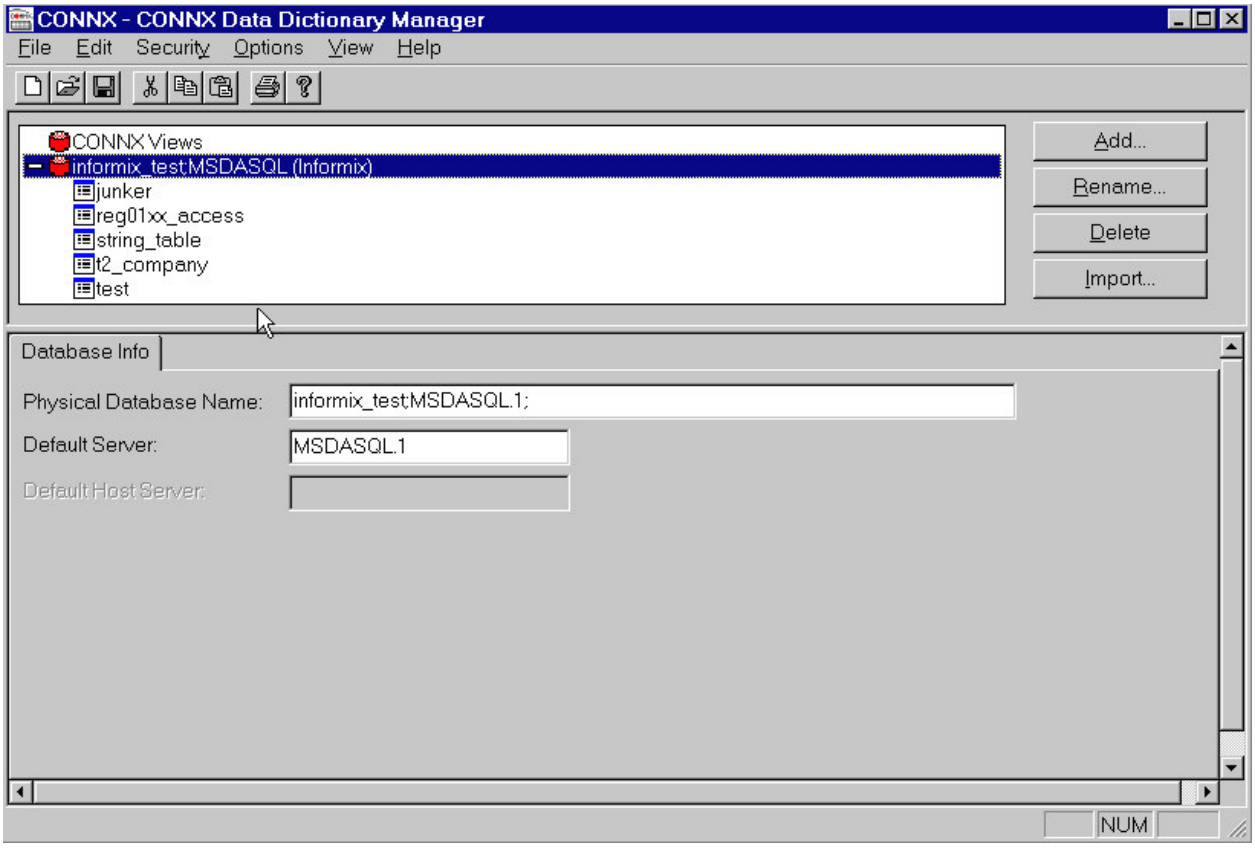
7. Select the **Include System Tables** check box to access system tables.

8. Select the **Get Statistics** check box to identify table sizes. This is used by the CONNX query optimization.

9. Click the **OK** button. The **CONNX Import Table Selection** dialog box appears with a list of available table names. Click the **Add** or **Add All** button to move the tables to the **Select Tables for Import** pane.



10. Click the **OK** button to import the selected tables into CONNX. The imported tables are added to the list of accessible objects in the CONNX Data Dictionary Manager window.



Related Topic

CONNX Informix Data Module

CONNX and Oracle

Importing Oracle Database Tables

By importing Oracle tables, users can access and manipulate Oracle data with all of the data that CONNX can access through a single driver. Some of the possible implementations for this functionality include data migration, ad hoc reporting, data warehousing, application development, and Y2K-compliance modification and confirmation.

Additional core features of Oracle access through CONNX include invocation of stored procedures, support for SQL passthrough, and Rdb list cursor support, which allows storage of up to 2 gigabytes of data in a single field.

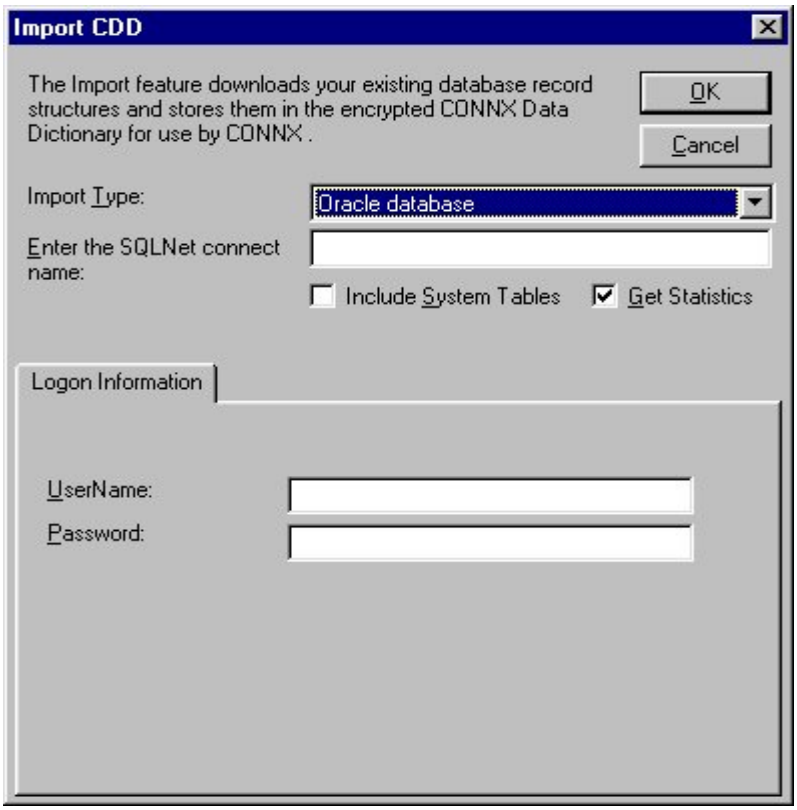
Oracle requires specific logon information during the initial import procedures to further ensure access security. The Import CDD dialog box has a separate tab in which to specify server name, user name, and password, as required.

Related Topics

- To import an existing Oracle database table
- Importing Oracle Rdb Database Tables
- To import an existing Oracle Rdb table

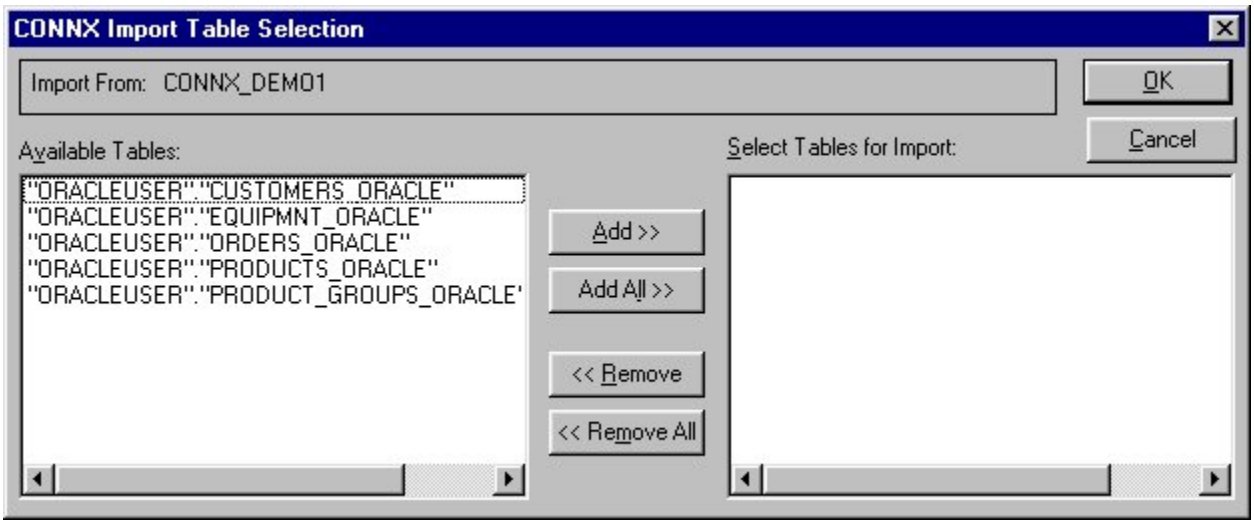
To import an existing Oracle database table

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **Oracle database** from the **Import Type** list box.
3. Type the SQLNet database name in the **Enter the SQLNet connect name** text box.



4. Select the **Include System Tables** check box to import system table definitions into the CDD.
5. Select the **Get Statistics** check box to identify the table sizes.
6. Type a user name, and password under Logon Information.
1. **UserName:** OpenVMS user name used to log in to the account where the database resides.
2. **Password:** OpenVMS password used to log in to the account where the database resides.

7. Click the **OK** button. The **CONNX Import Table Selection** dialog box appears with a list of available table names. Click the **Add** or **Add All** button to move the tables to the **Select Tables for Import** pane.



8. Click the **OK** button to import the selected tables into CONNX. The Oracle database tables are added to the list of accessible objects in the CONNX Data Dictionary Manager window.

Related Topic

Importing Oracle Database Tables

Importing Oracle Rdb Database Tables

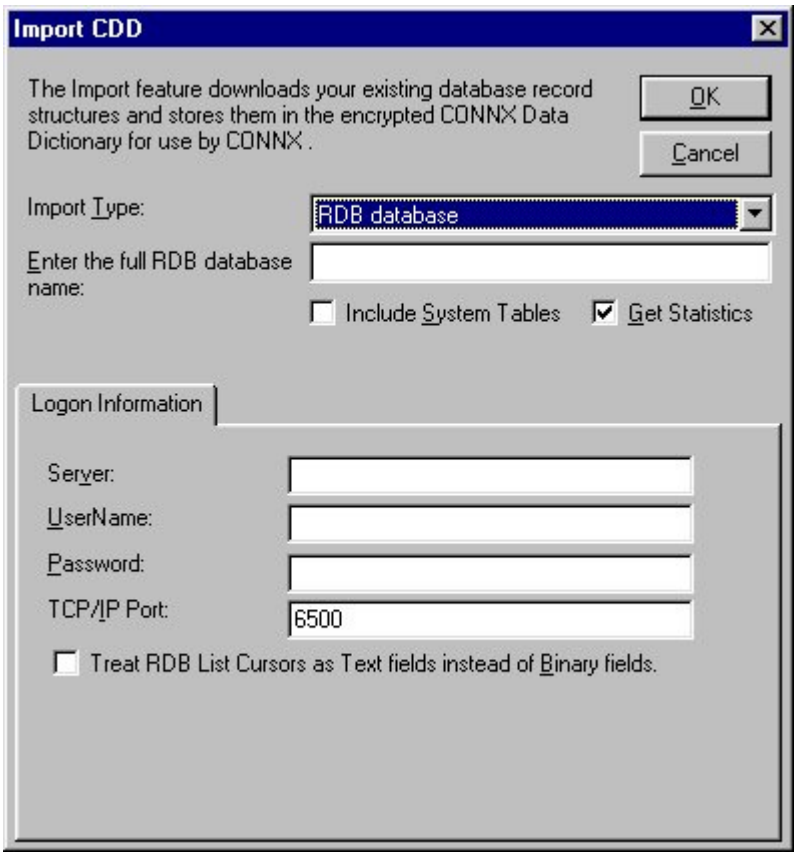
When importing Oracle Rdb database tables, users can access and manipulate Oracle Rdb data with all of the data that CONNX can access through a single driver.

Related Topic

To import an existing Oracle Rdb table

To import an existing Oracle Rdb table

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **Rdb database** from the **Import Type** list box.
3. Type the Rdb database name in the **Enter the Full RDB Database Name** text box. (You can override the name of the RDB database by entering a connection string parameter where, for example, RDBNAME = databasename.)
4. Specify the full VMS path for the database. The path may include logicals.

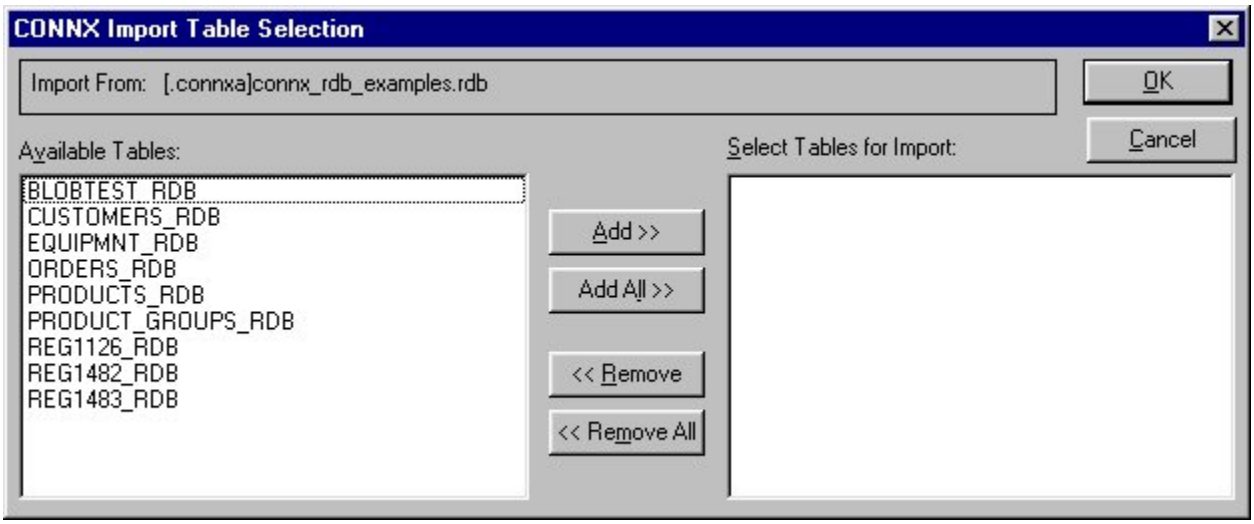


5. Select the **Include System Tables** check box to import table definitions into the CDD.
6. Select the **Get Statistics** check box to identify the table sizes.
7. Type a server location, user name, and password under Logon Information.

1. **Server:** Name of the OpenVMS system running CONNX for Rdb.
2. **UserName:** OpenVMS user name used to log in to the account where the database resides.
3. **Password:** OpenVMS password used to log in to the account where the database resides.
4. **TCP/IP Port:** Set to 6500 by default but can be changed for any current transaction.

8. Click the **Treat RDB List Cursors as Text Fields instead of Binary Fields** check box if your Rdb data contains Rdb list cursors that contain text data. (Rdb list cursors are also known as document fields.) Selecting this option would depend on the front-end application in use.

9. Click the **OK** button. The **CONNX Import Table Selection** dialog box appears with a list of available table names. Click the **Add** or **Add All** button to move the tables to the **Select Tables for Import** pane.



10. Click the **OK** button to import the selected tables into CONNX. The Rdb database tables are added to the list of accessible objects in the CONNX Data Dictionary Manager window.

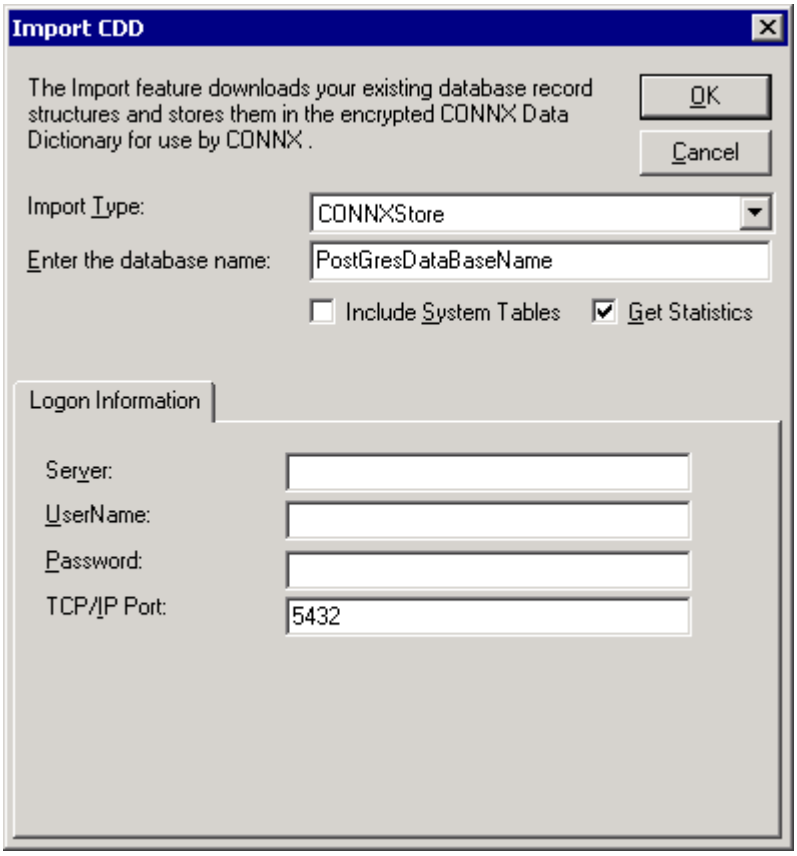
Related Topic

Importing Oracle Rdb Database Tables

CONNX and PostgreSQL

Importing PostgreSQL Database Tables

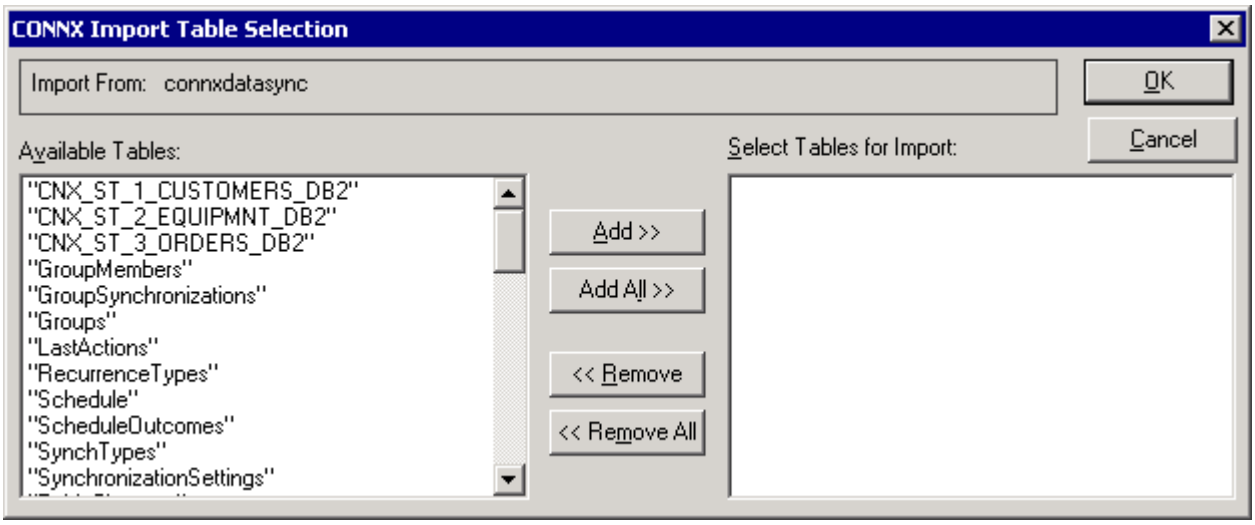
1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The Import CDD dialog box appears. Select **CONNXStore** from the Import Type list box. This is the name CONNX uses for imports of PostgreSQL data.
3. Type the PostgreSQL database name in the **Enter the database name** text box.



4. Specify the full path for the database.
5. Select the **Include System Tables** check box to import table definitions into the CDD.
6. Select the **Get Statistics** check box to identify the table sizes.
7. Type a server location, user name, and password under **Logon Information.**

1. **Server:** Name of the system running CONNX PostgreSQL.
2. **UserName:** User name used to log in to the account where the database resides.
3. **Password**: Password used to log in to the account where the database resides.
4. **TCP/IP Port:** Port 5432 is listed in the TCP/IP Port text box by default. If you have a firewall in place or need use of a specific port, enter the port number.

8. Click the **OK** button. The **CONNX Import Table Selection** dialog box appears with a list of available table names. Click the **Add** or **Add All** button to move the tables to the **Select Tables for Import** pane.



9. Click the **OK** button to import the selected tables into CONNX. The PostgreSQL database tables are added to the list of accessible objects in the CONNX Data Dictionary Manager window.

CONNX and SCT Import Rules

The rules below describe the SCT import logic.

1. A data file is imported in its entirety ONLY if the table will not be split into multiple logical CONNX tables. An exception is the ADFILE. This table will always be imported in addition to the ADFILE logical tables. The ADFILE table is for use in the CNXPREFERENCE address selection logic.
2. If there are CONNX tables of more than 250 fields, the table is divided into multiple tables to accommodate Microsoft Access. The first table contains the first 250 columns and is appended with the suffix "_1". Each subsequent table contains the key information and the next 1-250 fields, and is appended in ascending order "_2", "_3", and so on.

   All arrays are automatically rotated unless one of the following is true:
   - The array is nested - meaning that the current segment is already a rotated array.
   - The segment contains less than 150 fields.

3. Redundant Redefines are removed if all of the below are true.
   - The field number is greater than 5 and does not contain the words "Phone," "Fax," or "Zip."
   - The field is longer than 13 bytes.
   - The redefined field is directly next to the field in question.
   - The redefined field is directly next to the field and the base portions of the field names are identical.
   - The offsets are identical.
   - The array offsets are identical.

   Following is an example of a redundant redefine:

   When the SCT Cobol FD import is executed this SG-LN-DISB-DATES-RED REDEFINES SG-LN-DISB-DATES line decribed below will not be imported in the CONNX CDD. CONNX skips the field based on the redundant field SCT import rules. Alternatively, the SG-LN-D-DT-1, SG-LN-D-DT-2, SG-LN-D-DT-3 and SG-LN-D-DT-4 fields are imported.

   Excerpt from SCT Cobol FD SI$SOURCE:SCSGRC.LIB

```
07 SG-LN-DISB-DATES-RED REDEFINES SG-LN-DISB-DATES.
* SGN34 (042)
09 SG-LN-D-DT-1 PIC 9(08) COMP-3.
* SGN35 (047)
09 SG-LN-D-DT-2 PIC 9(08) COMP-3.
* SGN36 (052)
09 SG-LN-D-DT-3 PIC 9(08) COMP-3.
* SGN37 (057)
09 SG-LN-D-DT-4 PIC 9(08) COMP-3.
```

4. Field groups are removed if they are greater than 50 bytes. (The definition of a field group is a field containing multiple data elements represented as a single element or field in the Cobol FD.)
5. The field contains non-ASCII data (as with the Packed Decimal data type.)
6. The field is not one of the first five fields of the record segment.

In the following example, "03 PR-DATA PIC x(330)" line below for a length of 330 characters is a field group. The "03 PR-920-DATA REDEFINES PR-DATA" line in the COBOL FD breaks down the PR-DATA into 48 individual fields. The fields have different data types, yet the PR-DATA field group is defined as a string. Since the 48 fields are varying data types and the PR-DATA field is a string data type, if PR-DATA was imported and displayed the results would be garbled data.

Excerpt from SCT Cobol FD HR$SOURCE:ECEJRC.LIB:

```
03 PR-DATA PIC X(330).
03 PR-920-DATA REDEFINES PR-DATA.
05 PR-GROUP.
07 PR-FSCL-YR  PIC X(02).
07 PR-POS-N  PIC X(06).
07 FILLER  PIC X(01).
05 PR-OPEN-DATE  PIC 9(08).
05 PR-EXPIR-DATE  PIC 9(08).
05 PR-APPLIC-DEADLINE  PIC 9(08).
05 PR-REQ-TYPE  PIC X(01).
05 PR-STATUS  PIC X(02).
05 PR-STATUS-DATE  PIC 9(08).
05 PR-PRIOR-STATUS  PIC X(02).
05 PR-PRIOR-STATUS-DATE  PIC 9(08).
05 PR-ORG-CD.
07 PR-DIVISION  PIC X(03).
07 PR-DEPARTMENT  PIC X(05).
07 PR-UNIT  PIC X(05).
05 PR-JOB-GROUP  PIC X(01).
05 PR-JOB-CLASS  PIC X(04).
05 PR-EEO-OCCUP-CODE  PIC X(03).
05 PR-EEO-JOB-GROUP  PIC X(02).
05 PR-POS-TYPE  PIC X(01).
05 PR-REPL-POS-NO PIC X(06).
05 PR-POS-EX  PIC X(01).
05 PR-AUTH-HIRE-DATE PIC 9(08).
05 PR-RELOC-IND  PIC X(01).
05 PR-POS-FILL-DATE  PIC 9(08).
05 PR-POS-VAC-DATE  PIC 9(08).
PR-AUTHORIZATION-CODES.
07 PR-AUTH-CODE  PIC X(03)
OCCURS 5 TIMES.
05 PR-RECRUITER-ID  PIC X(15).
05 PR-HIRING-MGR-ID  PIC X(15).
05 PR-SUCCESS-ID  PIC X(15).
05 PR-COMMITTEE-IND PIC X(02).
05 PR-SHIFT-IND  PIC X(01).
05 PR-SUPER-RESPONSE PIC X(01).
05 PR-TOT-ADVERT-COST  PIC S9(7)V99 COMP-3.
05 PR-GEOG-AREA  PIC X(05).
05 PR-FTE  PIC (1)V9(5) COMP-3.
05 PR-WORK-HRS  PIC X(05).
05 PR-RANK  PIC X(30).
05 PR-TRACK  PIC X(01).
05 PR-TENURE  PIC X(01).
05 PR-ORIG-OPER-ID  PIC X(06).
```

```
05 PR-LAST-OPER-ID  PIC X(06).
05 PR-LAST-ACT-DATE  PIC 9(08).
05 PR-COMMENT  PIC X(50).
05 PR-920-USER-FILLER  PIC X(20).
```

7. Redundant arrays are removed if all of the below are true.
   1. The redundant array is directly next to the array in question.
   2. The offsets are identical.
   3. The base portion of the field names is identical.

   Following is an example of a redundant array.

   The AA-SR-INFO-FLAG line shown below is the only line imported into the CONNX CDD for the AA-SR-INFO-FLAG field. The redundant array rule directs CONNX to skip the fields labeled AA-SR-INFO-FLAG-1 through AA-SR-INFO-FLAG-30. These fields are simply redefines of the imported field AA-SR-INFO-FLAG. If imported along with the AA-SR-INFO-FLAG they would create redundancy and clutter the CDD with unnecessary fields.

   Excerpt from SCT COBOL FD SI$SOURCE:ACAARC.LIB:

```
05 AA-SR-INFO-FLAGS.
07 AA-SR-INFO-FLAG  PIC X(01)
OCCURS 30.
05 AA-SR-INFO-FLAGS-RED REDEFINES AA-SR-INFO-FLAGS.
07 AA-SR-INFO-FLAG-1  PIC X(01).
07 AA-SR-INFO-FLAG-2  PIC X(01).
07 AA-SR-INFO-FLAG-3  PIC X(01).
07 AA-SR-INFO-FLAG-4  PIC X(01).
07 AA-SR-INFO-FLAG-5  PIC X(01).
07 AA-SR-INFO-FLAG-6  PIC X(01).
07 AA-SR-INFO-FLAG-7  PIC X(01).
07 AA-SR-INFO-FLAG-8  PIC X(01).
07 AA-SR-INFO-FLAG-9  PIC X(01).
07 AA-SR-INFO-FLAG-10  PIC X(01).
07 AA-SR-INFO-FLAG-11  PIC X(01).
07 AA-SR-INFO-FLAG-12  PIC X(01).
07 AA-SR-INFO-FLAG-13  PIC X(01).
07 AA-SR-INFO-FLAG-14  PIC X(01).
07 AA-SR-INFO-FLAG-15  PIC X(01).
07 AA-SR-INFO-FLAG-16  PIC X(01).
07 AA-SR-INFO-FLAG-17  PIC X(01).
07 AA-SR-INFO-FLAG-18  PIC X(01).
07 AA-SR-INFO-FLAG-19  PIC X(01).
07 AA-SR-INFO-FLAG-20  PIC X(01).
07 AA-SR-INFO-FLAG-21  PIC X(01).
07 AA-SR-INFO-FLAG-22  PIC X(01).
07 AA-SR-INFO-FLAG-23  PIC X(01).
07 AA-SR-INFO-FLAG-24  PIC X(01).
07 AA-SR-INFO-FLAG-25  PIC X(01).
07 AA-SR-INFO-FLAG-26  PIC X(01).
07 AA-SR-INFO-FLAG-27  PIC X(01).
07 AA-SR-INFO-FLAG-28  PIC X(01).
```

```
07 AA-SR-INFO-FLAG-29  PIC X(01).
07 AA-SR-INFO-FLAG-30  PIC X(01).
```

8. The array to be removed is the flattened CONNX array (occur clause) if the segment is already a rotated array (nested array).

9. The array to be removed is the SCT explicit redefine if the segment is not a rotated array. CONNX then rotates the array, negating the need for the redundant SCT explicit array redefine.

Related Topics

To import from RMS SCT COBOL FD files

To perform an RMS SCT DBD Overlay Import

SCT DBD Overlay Conventions

Overlay Conventions

The following record types are used in the overlay process:

Record Type	Description
DF	For file determination.
DS	To parse record segments.
DC	To populate the comment field in the CONNX CDD.
DE	To populate the DBD field name in the CONNX CDD.
DY	To populate the DBD field name if no DBD record is found.

- Each field can be renamed once, based on the offset and record segment. Subsequent field names for the same offset and length are ignored.
- The DE record is used to parse the segment if the SCT table has a SQL view clause in the CONNX CDD for the relevant table.
- If a segment indicator is found in the DS record, the dynamic array offset is used to calculate the field offset being renamed.
- All comments that begin with the words FOCUS or RECORD COPY are ignored.
- All comments are truncated to the first 250 characters.
- Any field containing the word FILLER is not renamed although the comments are applied to all FILLER fields that have comments in the DBD.

Related Topics

» To import from RMS SCT COBOL FD files
» To perform an RMS SCT DBD Overlay Import

CONNX SQL Server Module

The CONNX SQL Server module enables access to SQL Server database objects through the Microsoft SQL Server driver. CONNX has been tested with SQL Server 7.0 and SQL Server 2000 for ODBC and OLE DB.

For multiple connections to SQL Server via ODBC through a single CONNX Data Dictionary, either the SQL Server data source name must be a file-based DSN and installed on a shared network server, or the same DSN must be configured on each client computer.

Important: *The selected data source driver must be installed on all client machines that will have access to the data source tables since they will be using the same DSN (Data Source Name) as described in ODBC Data Source Names Used with Multiple Users in SQL Server 7.0 and 2000.*

ODBC Data Source Names Used with Multiple Users in SQL Server 7.0 and 2000

There are three types of ODBC DSNs that can be used to create connections to SQL Server databases using CONNX: user, system, or file-based. The configuration of the DSN must be carefully considered whenever a CDD is intended for use by multiple users on multiple machines.

If the ODBC connection is made using the user or system-based DSN, every machine that uses this CONNX Data Dictionary must have the same DSN registered in its ODBC driver manager. For example, if the creator of the COMP_INVEN.CDD has a user DSN "My_inventory" that connects to the SQL Server machine warehouse, then every user of that CDD must have the same DSN "My_inventory" registered on their machine. To enable all users to access a newly added ODBC data source, it is recommended that you use a file-based DSN.

A file-based DSN is not specific to any machine or user and can be made accessible to all. The file-based DSN can be placed on a network share. However, with file-based DSNs the path entered by the user who imported the ODBC tables is stored in the CDD. For example, if the creator of the CDD file COMP_FDINV.CDD enters a file DSN of X:\ACCOUNTING\INVEN.DSN, all users with access to that CDD must be able to access X:\ACCOUNTING\INVEN.DSN.

One convention used to solve this scenario is that all user machines are given access to the same network drive. Another convention is to use a UNC (universal naming convention), such as \\MySERVER\Myshare\INDEV.DSN, which grants users access to the network drive without allocating a specific drive letter.

As with any database connection, if the tables on the database are modified, the CDD may need to be re-imported.

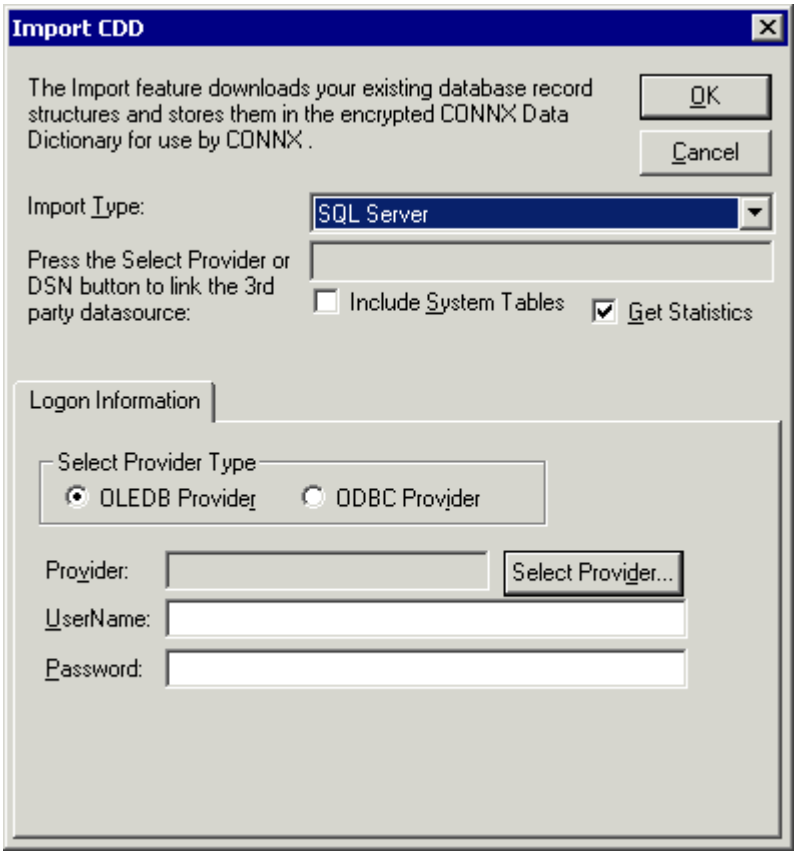
Regardless of DSN configuration, however, the ODBC driver or OLE DB provider must be installed on all client machines accessing the data source except in three-tier scenarios where it must be installed on the middle tier.

Related Topics

- CONNX SQL Server Module
- To import an existing Microsoft OLE DB provider for SQL Server 7.0 data objects
- To import objects from an existing SQL Server 6.x or 7.0 provider data source

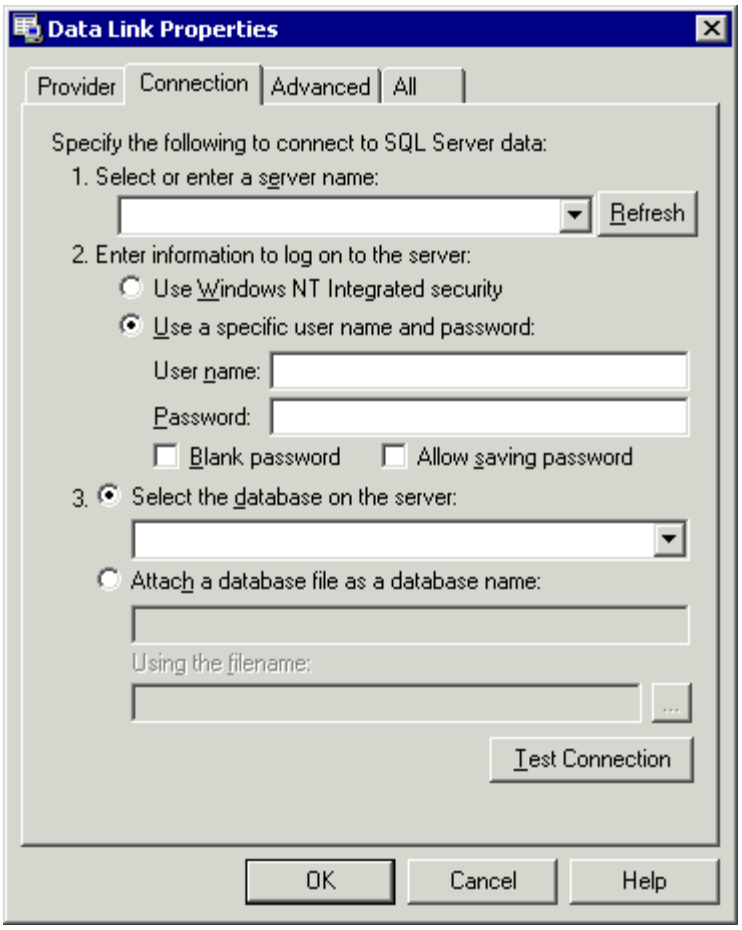
To import a SQL Server database using an OLE DB Provider

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **SQL Server** from the **Import Type** list box in the **Import CDD** dialog box.

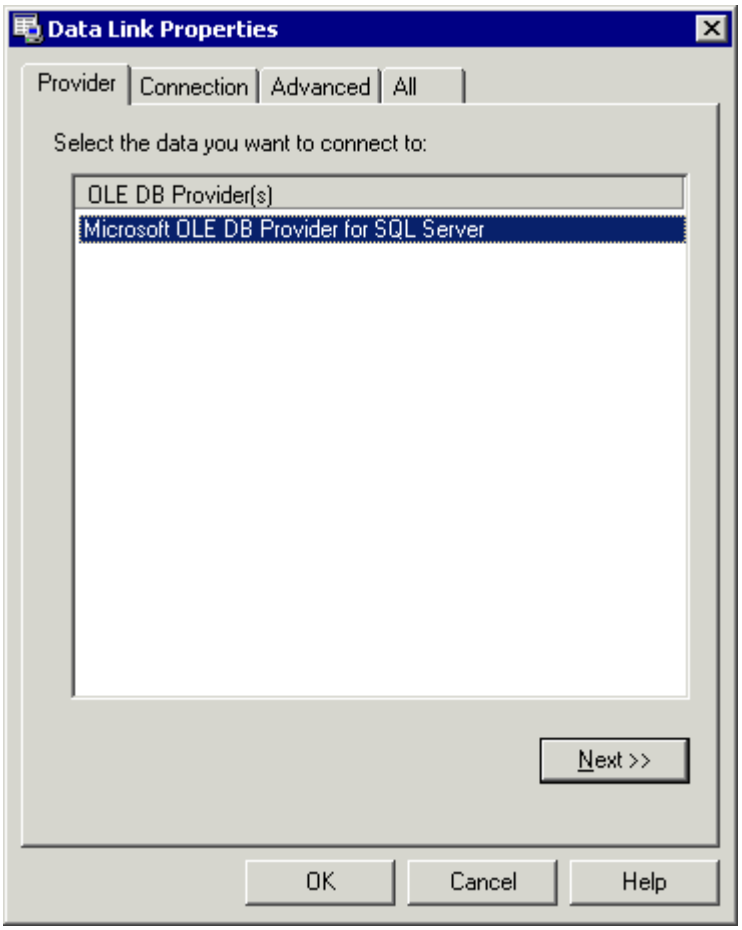


Important: *This direct OLE DB connection to SQL Server only works with SQL Server 7.0 or later.*

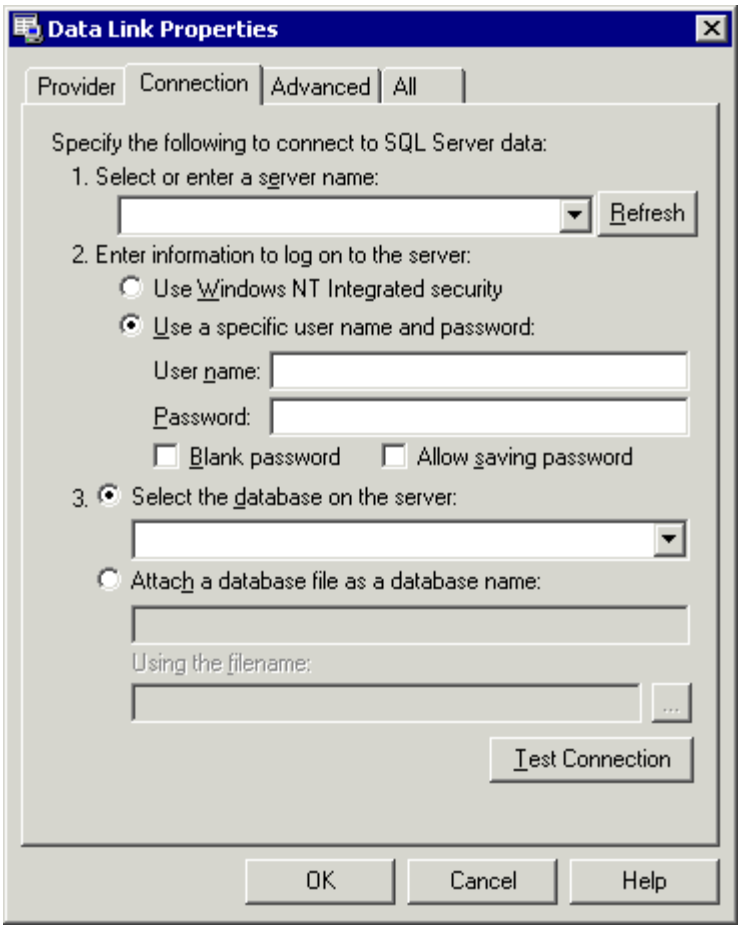
3. Select **OLE DB Provider** under **Select Provider Type,** and then click the **Select Provider** button. The **Data Link Properties** dialog box appears.



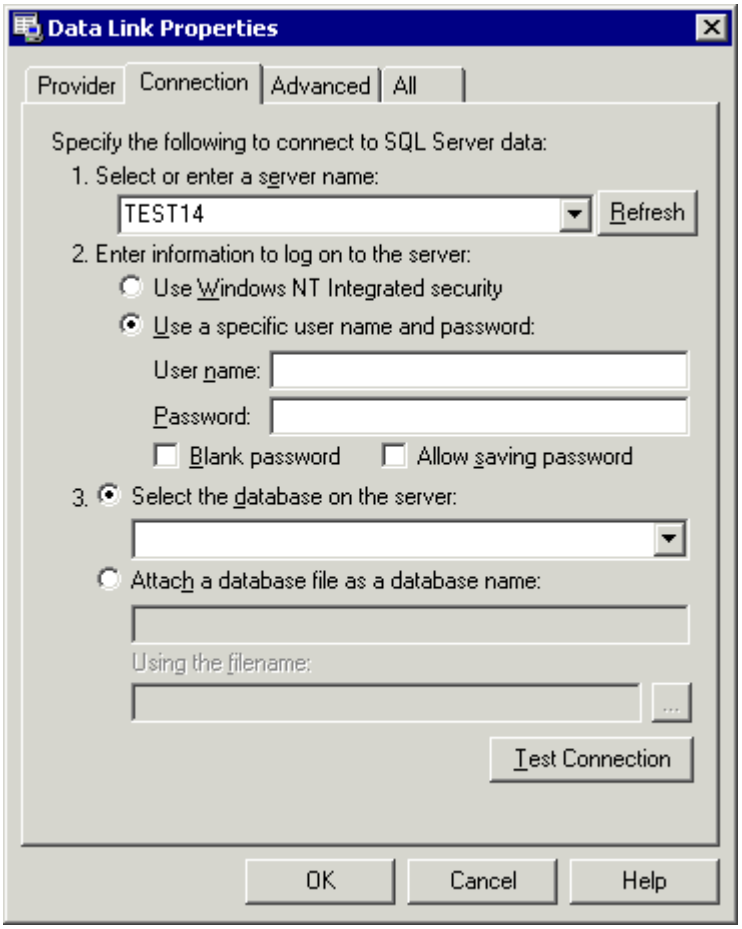
4. Select the **Provider** tab, and then select the desired OLE DB Provider for SQL Server. In this instance, it is the Microsoft OLE DB Provider for SQL Server.



5. Click the **Next** button to return to the **Connection** tab.


Data Link Properties
Provider
Connection
Advanced
All
Specify the following to connect to SQL Server data:
1. Select or enter a server name:
Refresh
2. Enter information to log on to the server:
Use Windows NT Integrated security
Use a specific user name and password:
User name:
Password:
Blank password
Allow saving password
3. Select the database on the server:
Attach a database file as a database name:
Using the filename:
Test Connection
OK
Cancel
Help

6. Enter a server name in **Item 1** on the **Connection** tab.



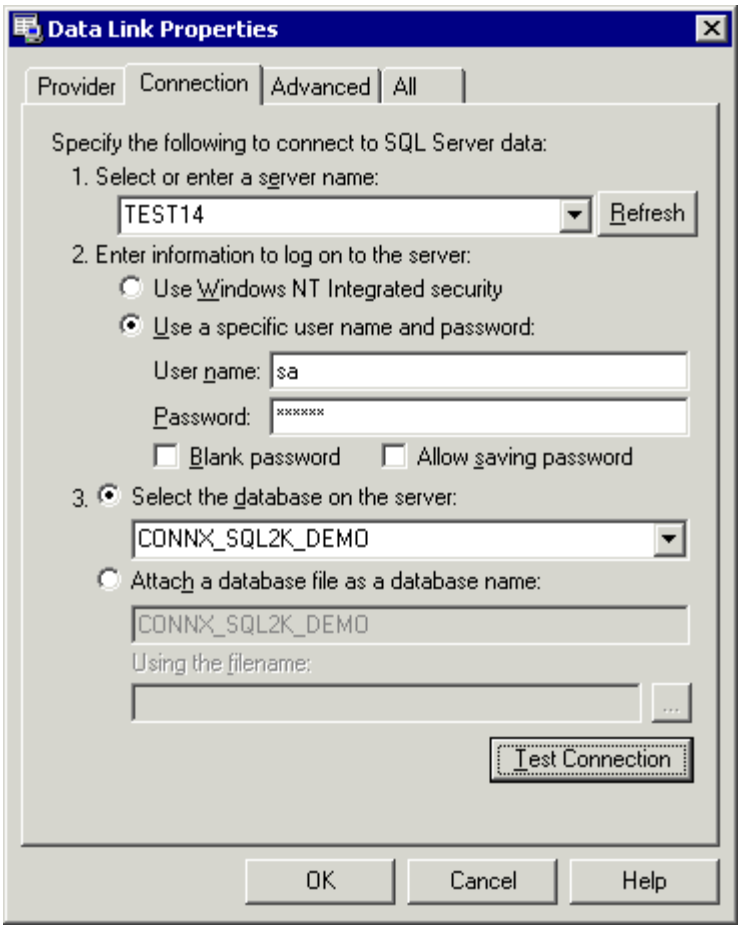
7. Enter a user name and password**.**

   Important: *If you open the list box in Item 1, it uses the SQLOLEDB Enumerator, which attempts to make network calls.*

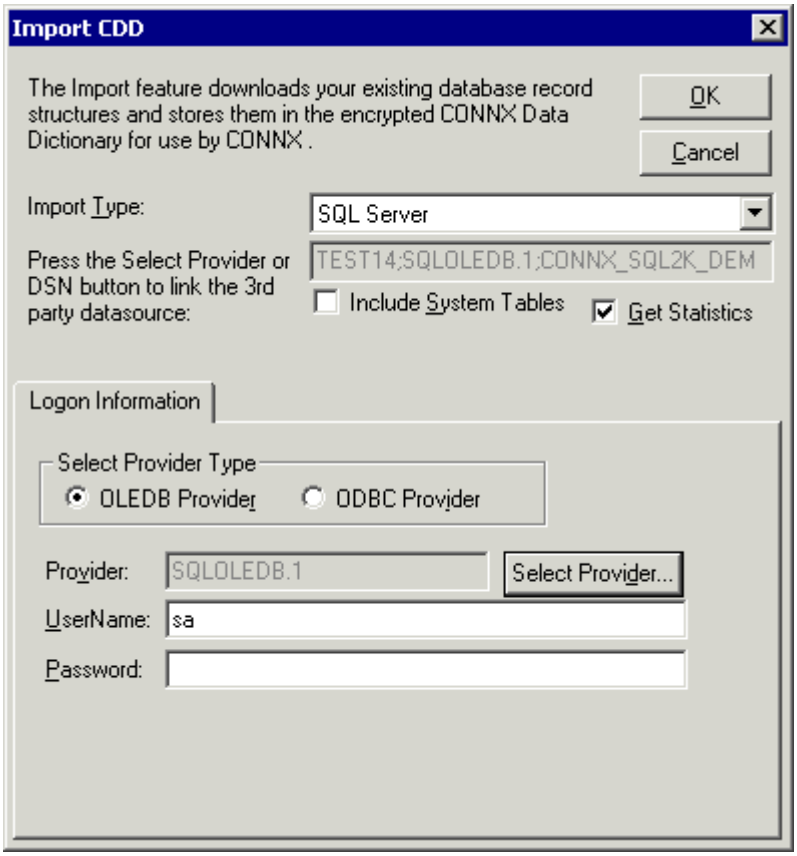
   Note: SQL Server documentation may use the word "database" for the word "catalog."

8. Select a catalog from the list box in **Item 3.** SQL Server loads the selected catalog automatically. Click the **Test Connection** button to verify that the SQL catalog is available. If Item 3 is left blank, the user's default catalog is used.

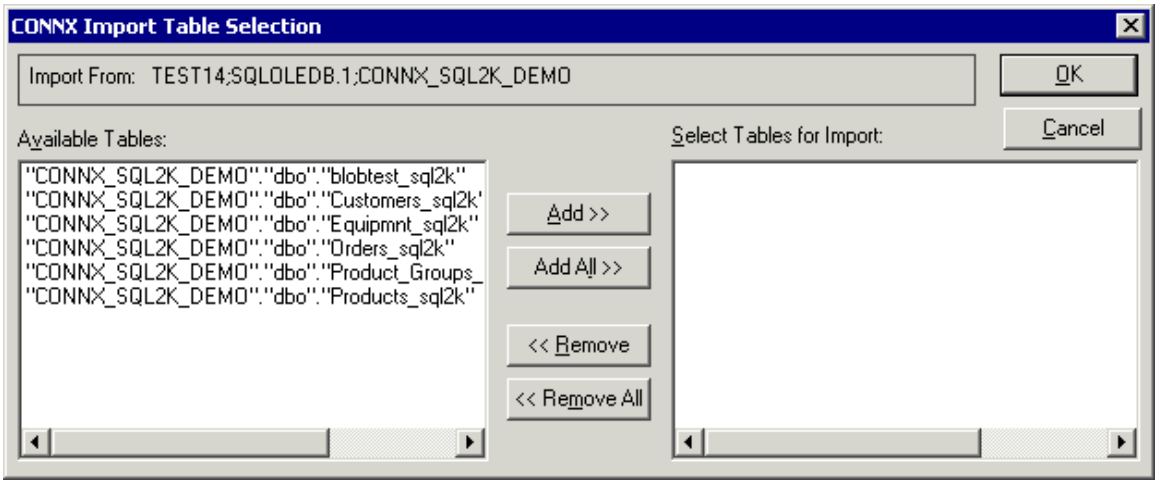
   If you have unattached SQL database files (consult SQL Server documentation), you may reattach the database file as a catalog by selecting **Attach a database file as a database name.** The system mounts that database file and uses it as the current catalog.



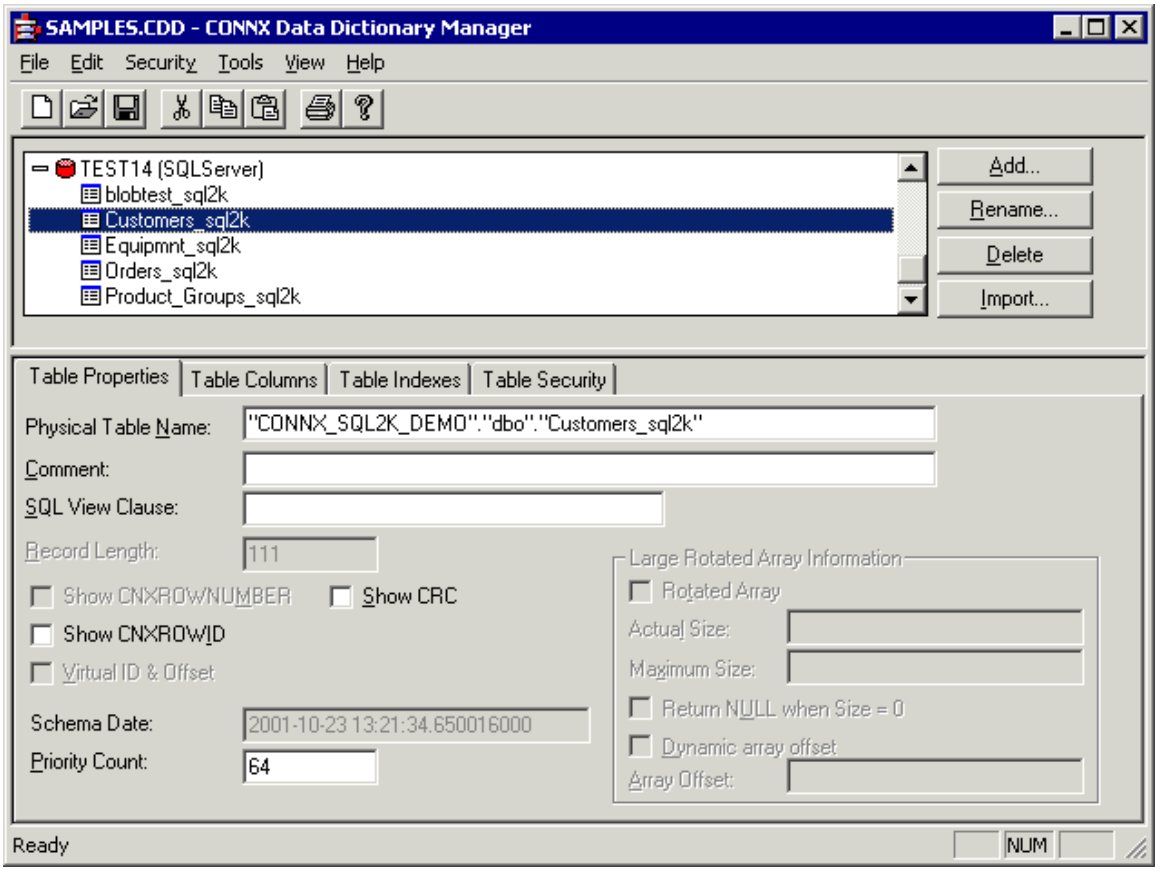
9. Click the **OK** button. The **Import CDD** dialog box appears. Reenter the SQL password, if required.



10. Normally, only user-defined tables can be selected for import. Select the **Include System Tables** check box to enable the import of non-user-defined tables. Select the **Get Statistics** check box to identify the table sizes. This is used by CONNX query optimization. Click the **OK** button.
11. The **CONNX Import Table Selection** dialog box appears. To import all of the tables in the database, click the **Add All** button. To import some of the tables in the database, select the tables to import, and then click the **Add** button.



12. Click the **OK** button to import the selected catalogs into CONNX. The imported catalogs appear in the list of accessible objects in the CONNX Data Dictionary Manager window.

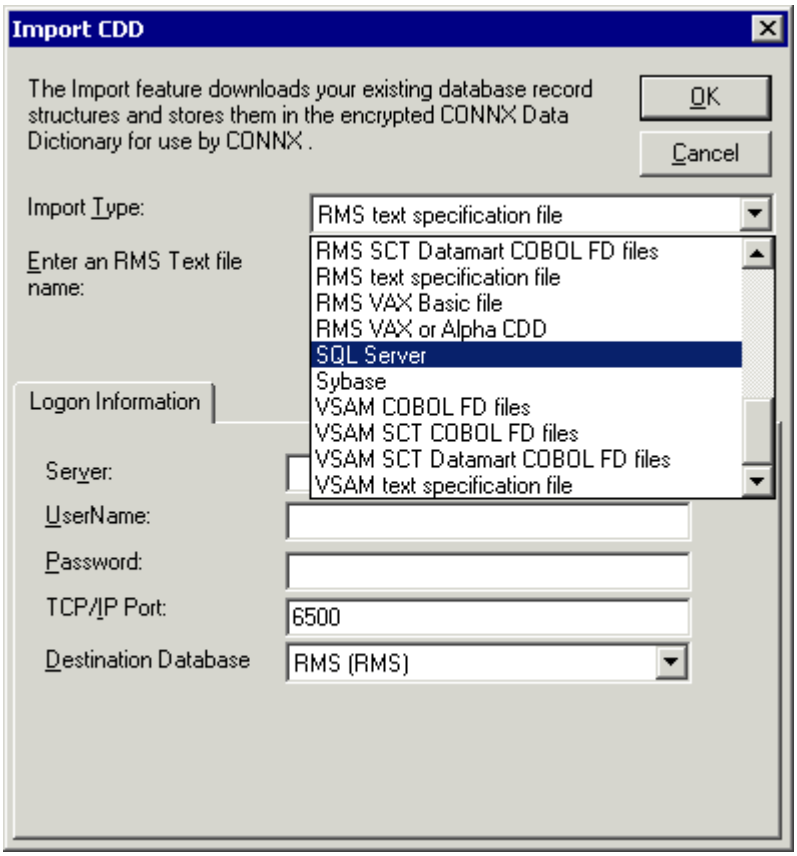


Troubleshooting

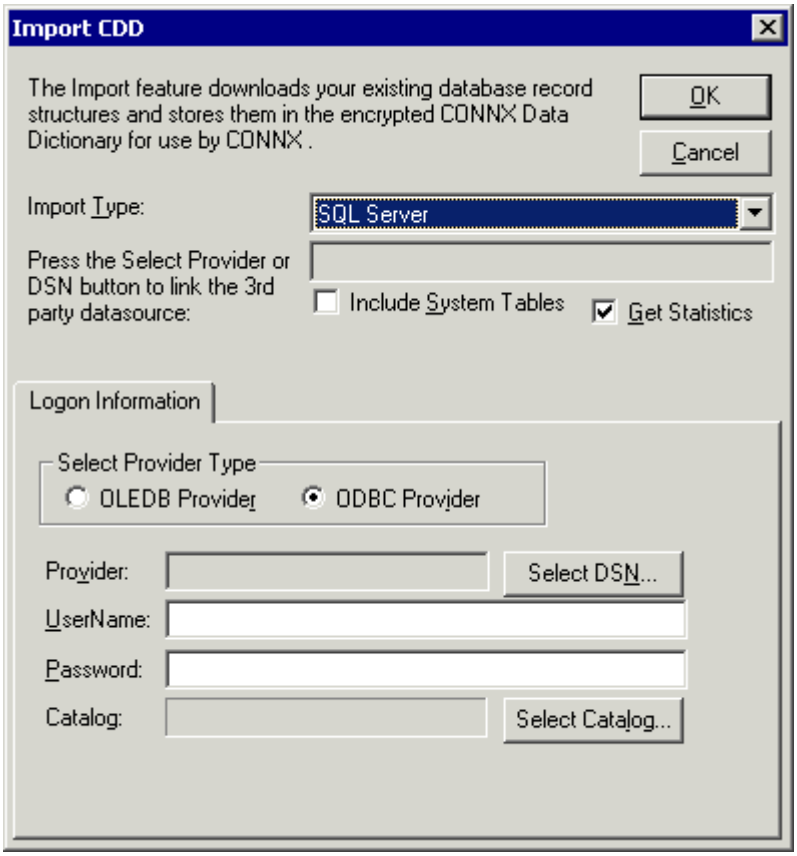
If, when connecting to SQL Server, a blank table appears in the **CONNX Import Table Selection** dialog box in step 11, it means that the user may not have sufficient privileges to access the catalog. The SQL Server administrator should use the Enterprise Manager tool to change the security level for that user to enable access to the table. In the SQL Server Enterprise Manager, the **Properties** dialog box of each user has a database access tab that can be used to control access levels.

To import objects from a SQL Server ODBC provider data source

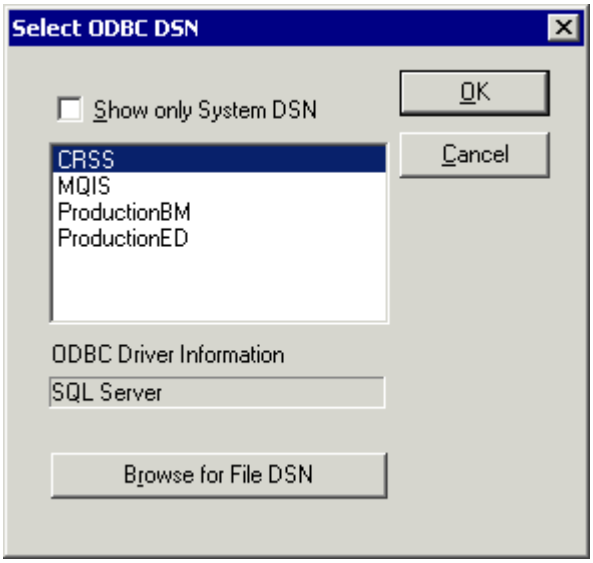
1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **SQL Server** from the **Import Type** list box.



3. Select the **Include System Tables** check box to access system tables. Select the **Get Statistics** check box to identify table sizes. This is used by CONNX query optimization.
4. Select **ODBC Provider** under **Select Provider Type,** and then click the **Select DSN** button in the **Logon Information** pane.



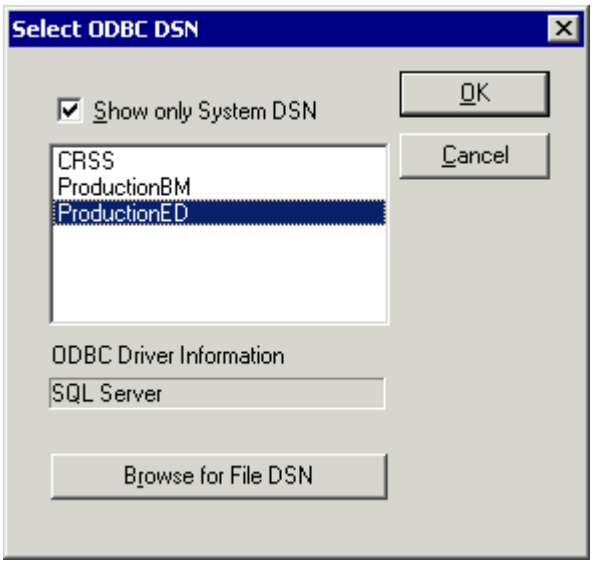
5. The **Select ODBC DSN** dialog box appears. You can choose either a user, system, or file-based DSN, although it is recommended that a file-based DSN be used, since it can be accessed through a network by multiple users.



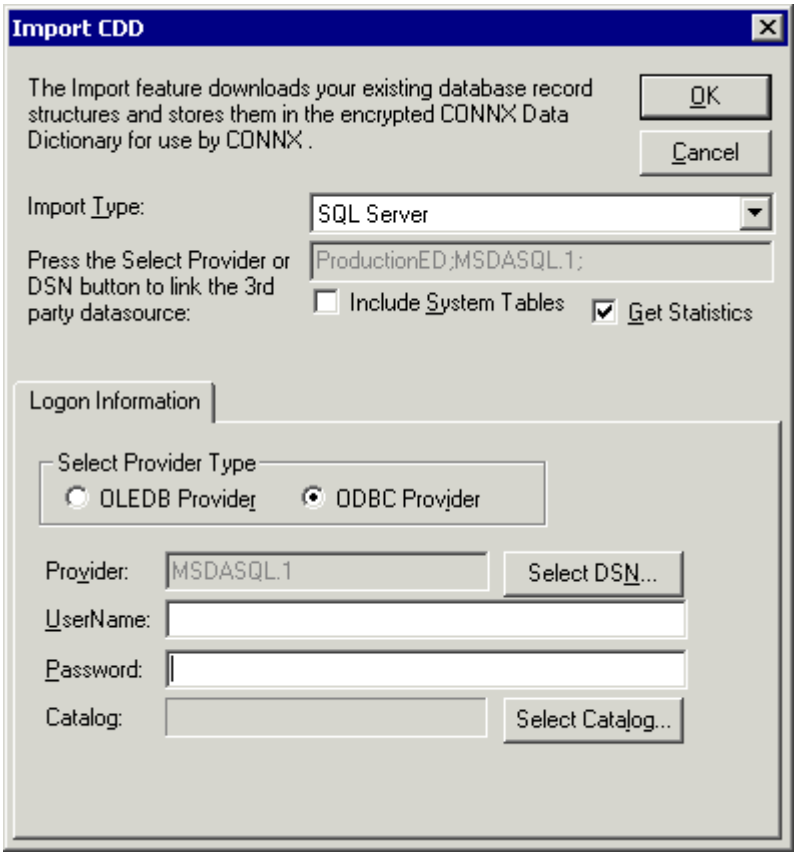
File DSN (Recommended)
Click the **Browse for File DSN** button to open the **Open** dialog box. Select a file-based DSN from lists of available .dsn files.

System DSN or User DSN
Select the **Show only System DSN** check box to view a list of system DSNs only. Clear the check box to show a list of both user and system DSNs. Note that the ODBC Driver Information text box displays information about the selected ODBC DSN. The same information can be found in the ODBC Data Source Administrator dialog box. Click the **OK** button in the **Select ODBC DSN** dialog box to return to the **Import CDD** dialog box, which displays the selected DSN.

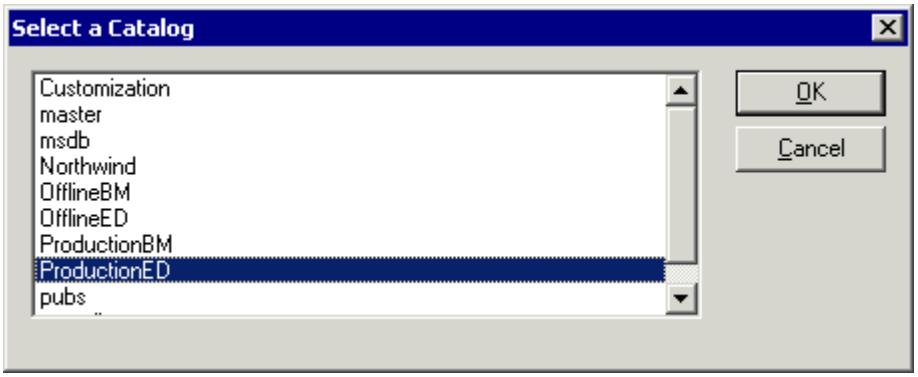
In this example, a system DSN is used. Click the **OK** button.



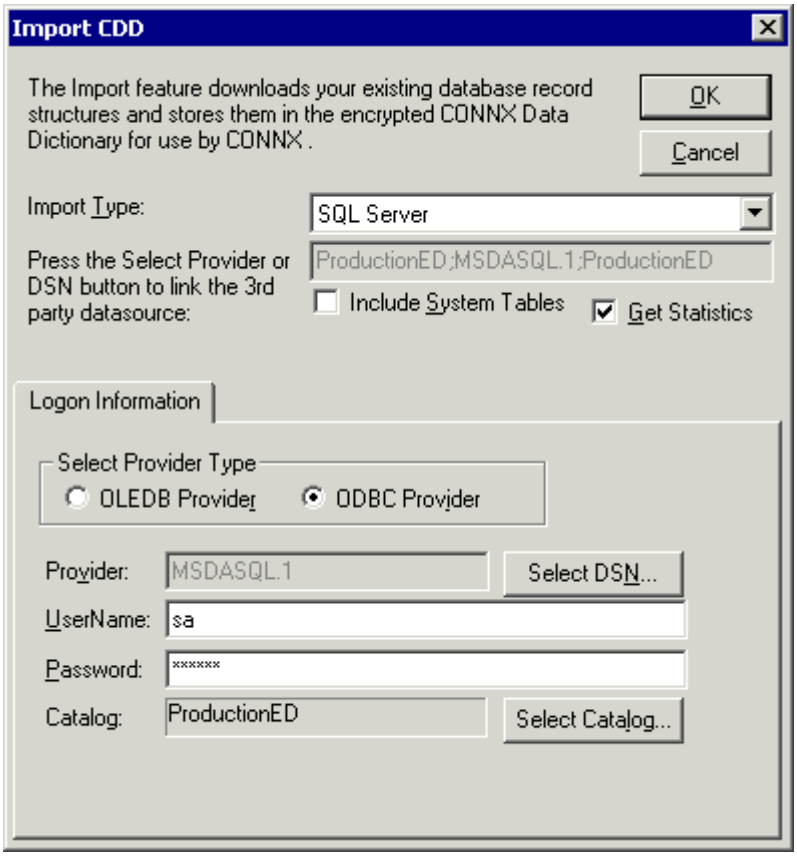
6. The **Import CDD** dialog box appears.



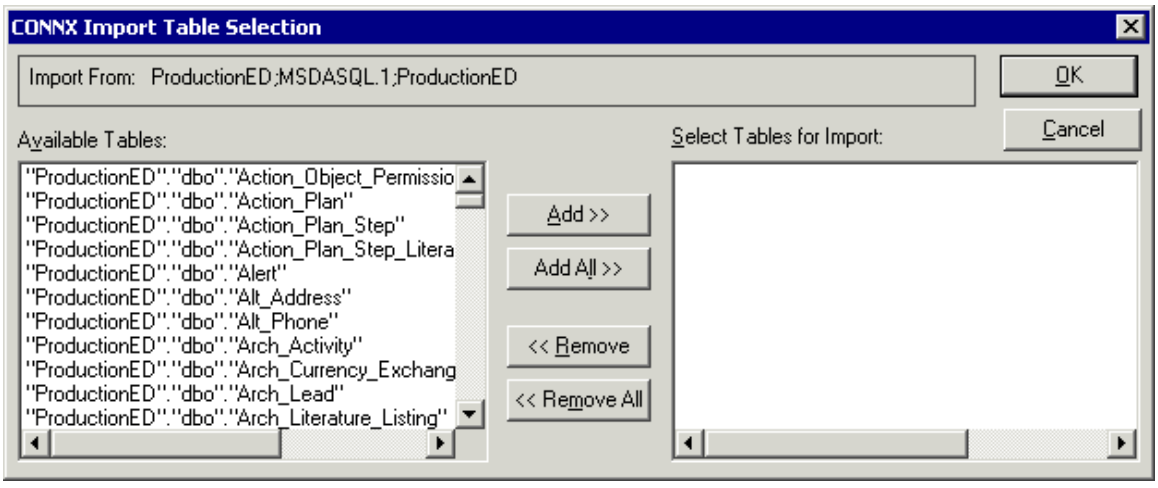
7. Enter a user name and password appropriate to the data source.
8. To specify a catalog, enter a name and password for the database, and then click the **Select Catalog** button to log into the database and view a list of available catalogs. The **Select a Catalog** dialog box appears.



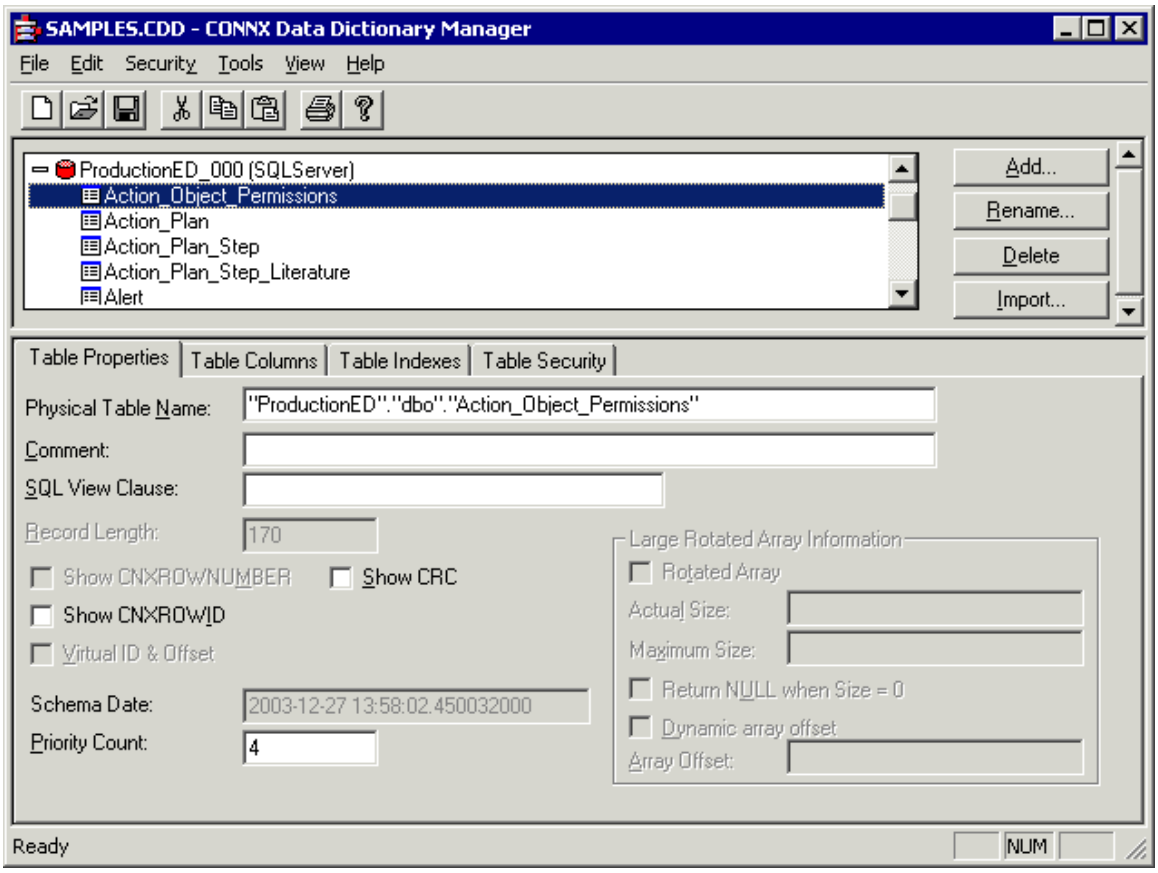
9. Select a catalog, and then click the **OK** button to return to the **Import CDD** dialog box. Normally, only user-defined tables can be selected for import.



10. Click the **OK** button. The **CONNX Import Table Selection** dialog box appears with a list of available table names. To import all of the tables in the database, click the **Add All** button. To import some of the tables in the database, select the tables to import, and then click the **Add** button.



11. Click the **OK** button to import the selected tables into CONNX. The imported tables are added to the list of accessible objects in the CONNX Data Dictionary Manager window.


SAMPLES.CDD - CONNX Data Dictionary Manager
File Edit Security Tools View Help
ProductionED_000 (SQLServer)
Action_Object_Permissions
Action_Plan
Action_Plan_Step
Action_Plan_Step_Literature
Alert
Add...
Rename...
Delete
Import...
Table Properties
Table Columns
Table Indexes
Table Security
Physical Table Name:
"ProductionED"."dbo"."Action_Object_Permissions"
Comment:
SQL View Clause:
Record Length:
170
Show CNXROWNUMBER
Show CRC
Show CNXROWID
Virtual ID & Offset
Schema Date:
2003-12-27 13:58:02.450032000
Priority Count:
4
Large Rotated Array Information
Rotated Array
Actual Size:
Maximum Size:
Return NULL when Size = 0
Dynamic array offset
Array Offset:
Ready
NUM

CONNX and Sybase

CONNX Sybase Data Module

The CONNX Sybase module provides access to Sybase data source objects through the Sybase ODBC driver.

For multiple connections to Sybase via ODBC through a single CONNX Data Dictionary, either the Sybase data source name must be a file-based DSN and installed on a shared network server, or the same DSN must be configured on each client computer.

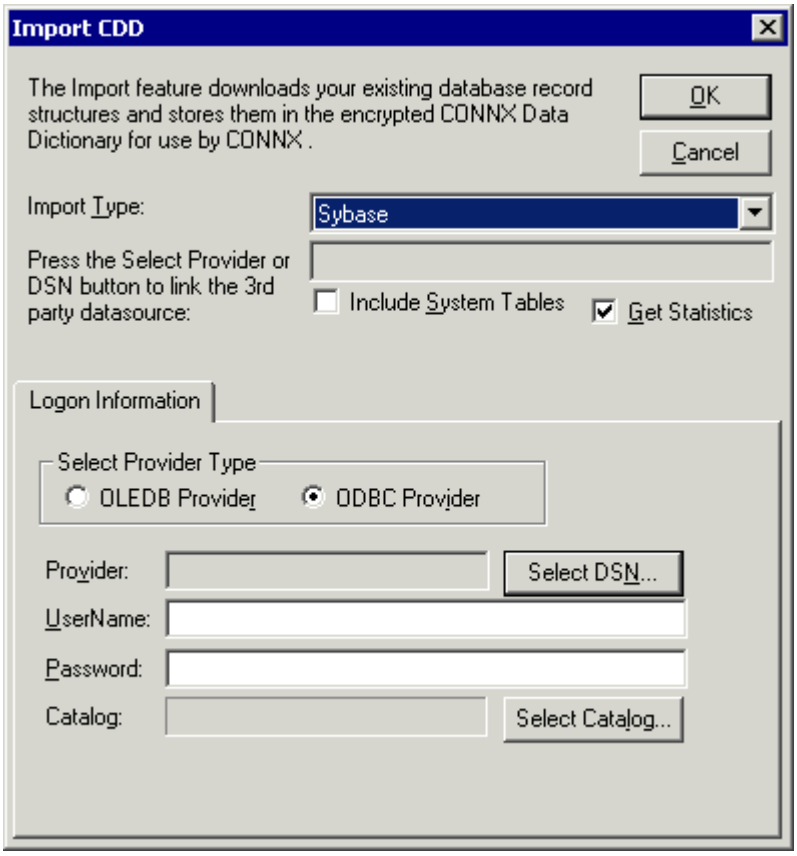
Important: *The selected data source driver must be installed on all client machines that have access to the data source tables since they will be using the same DSN (Data Source Name) as described in ODBC Data Source Names Used with Multiple Users.*

Related Topics


To import tables from a Sybase ODBC-compliant data source

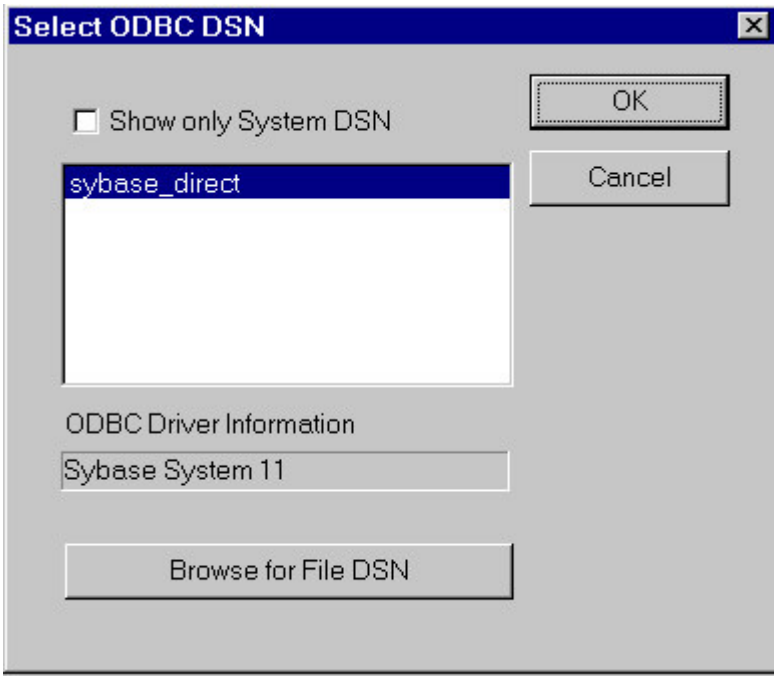
To import tables from a Sybase ODBC-compliant data source

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **Sybase** from the **Import Type** list box.



Important: The driver used to import table information must support ADO and be fully ODBC Level 2 compliant.

3. Select **ODBC Provider** under **Select Provider Type,** and then click the **Select DSN** button in the **Logon Information** pane.
4. The **Select ODBC DSN** dialog box appears. You can choose either a user, system, or file-based DSN, although it is recommended that a file DSN be used, since it can be accessed through a network by multiple users.



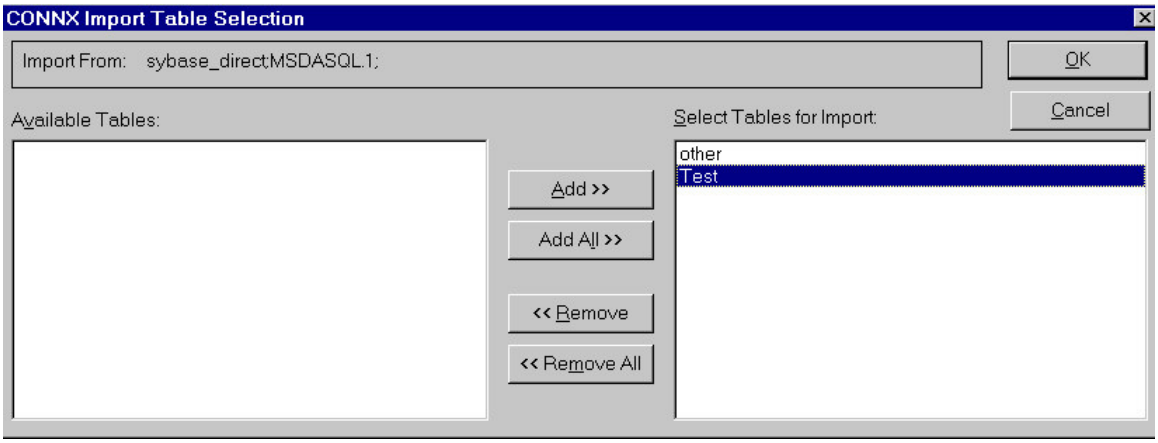
Your ODBC data source may require additional information. Review the documentation provided by your driver vendor for additional configuration options.

File DSN (Recommended)
Select the **Browse for File DSN** check box to open the **Open** dialog box. Select a file DSN from lists of available .dsn files.

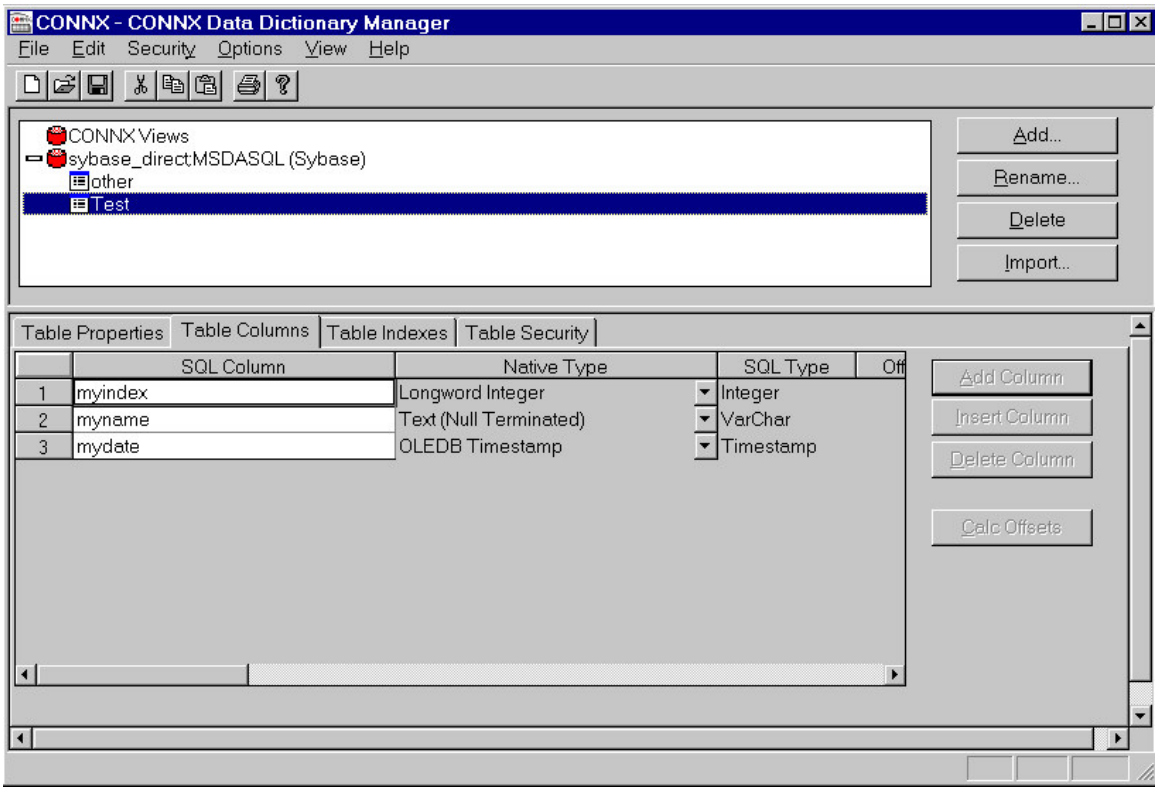
System DSN or User DSN
Select the **Show Only System DSN** check box to view a list of System DSNs only. Clear the check box to show a list of both User and System DSNs. Note that the **ODBC Driver Information** text box displays information about the selected ODBC DSN. The same information can be found in the ODBC Data Source Administrator dialog box. Click the **OK** button in the **Select ODBC DSN** dialog box to return to the **Import CDD** dialog box, which displays the selected DSN.

Important: *Some applications, such as web servers, can only access system DSNs or file-based DSNs.*

5. The **Import CDD** dialog box appears.
6. Select the **Include System Tables** check box to access system tables.
7. Select the **Get Statistics** check box to identify table sizes. This is used by the CONNX query optimization.
8. Click the **OK** button. The **CONNX Import Table Selection** dialog box appears with a list of available table names. Click the **Add** or **Add All** button to move the tables to the **Select Tables for Import** pane.



9. Click the **OK** button to import the selected tables into CONNX. The imported tables are added to the list of accessible objects in the CONNX Data Dictionary Manager window.



Related Topic

CONNX Sybase Data Module

CONNX and VSAM

CONNX and VSAM - MVS-OS/390

To import from VSAM COBOL FD files (CICS) - MVS z/OS

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears.
3. Select **VSAM COBOL FD** files in the **Import Type** list box.
4. Type a CICS user name in the **User Name** text box.
5. The CICS user ID must be authorized for FTP access to the target host, and must contain a RACF OMVS segment. For example, the default CICS user ID CICSUSER can be modified via RACF commands to permit FTP access by adding an OMVS segment:

```
USER=CICSUSER  NAME=CICSUSER
OMVS INFORMATION
_________

UID= 0000000000
CPUTIMEMAX= NONE
ASSIZEMAX= NONE
FILEPROCMAX= NONE
PROCUSERMAX= NONE
THREADSMAX= NONE
MMAPAREAMAX= NONE
```

6. The CICS userid used for CONNX CDD imports need not be authorized for TSO access, as long as it has read authority to the target COBOL copybook partitioned data sets. By default, the CONNX COBOL copybooks reside in CONNX.VVRR.COPYBOOK

7. Type a **CICS** password in the **Password** text box.

8. The TCPIP port number is set to 6500 by default, but can be configured via the CONNX NX01 CICS transaction.

9. Select the destination database from the Destination Database list box. See Adding a Database Connection for more information.

10. Type a **COBOL FD partitioned data set and member name** in the Enter a COBOL FD data set member text box.

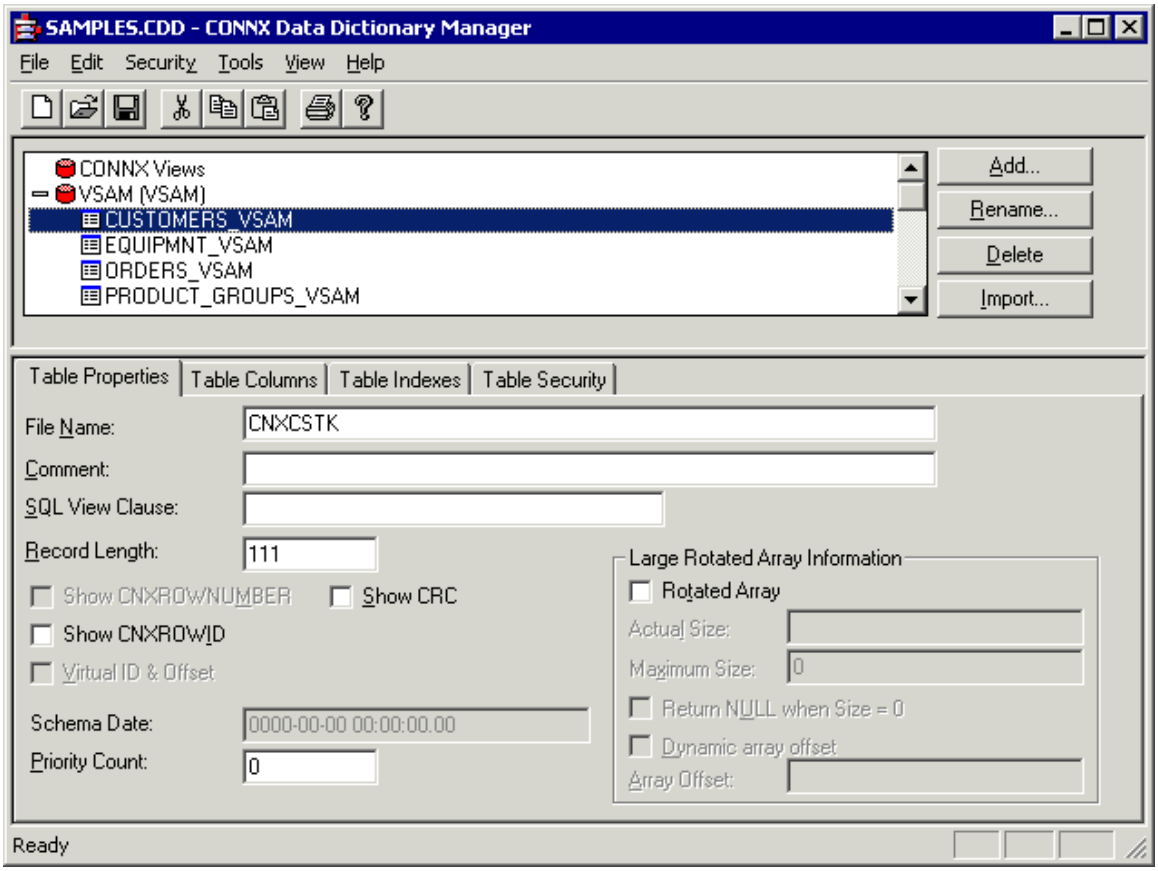
11. Specify the fully qualified PDS (partitioned data set) and member name for the COBOL FD, for example, CONNX.VVRR.COPYBOOK(CUSTOMER).

12. A COBOL FD specification does not contain the corresponding CICS File Control Table (FCT) name or Started Task DD (Data Definition) name; therefore, it must be specified on the Table Properties tab in the CONNX Data Dictionary Manager window when the data is returned.

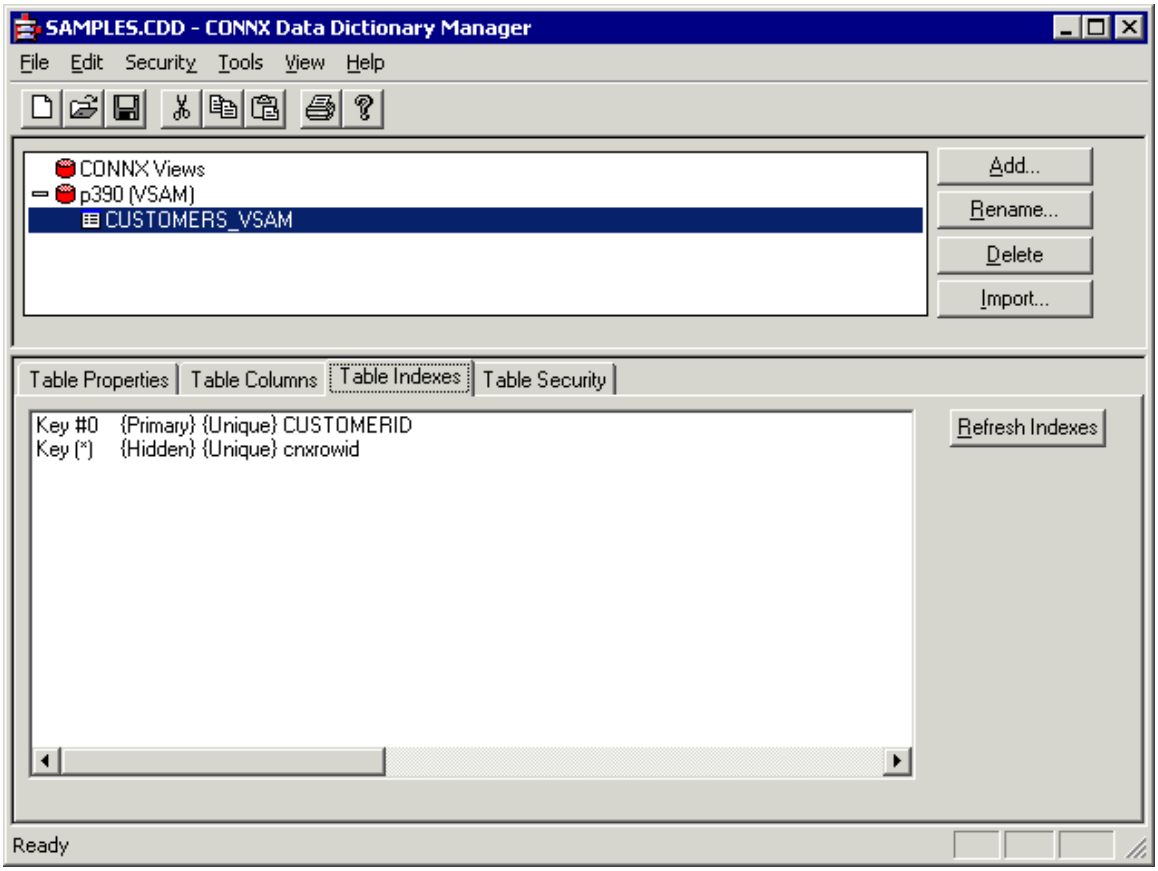
    1. All of the record layouts in the specified file are imported.
    2. No additional logon information is required.

13. Select each file to import and follow these steps:

    1. Click the **Table Properties** tab in the CONNX Data Dictionary Manager window.
    b. Type the **CICS FCT** name for the file in the CICS File Name text box. For example, the CICS short file name for the CONNX VSAM KSDS sample customer file is CNXCSTK.



3. Tab out of the **CICS File Name** text box to display the **CONNX Database Logon** dialog box. Click the **OK** button.
4. Click the **Table Indexes** tab to display the key information for the imported CICS VSAM file.



5. If the **Table Index** information list box is empty, click the **Refresh Indexes** button. If the list box remains empty, no indexes are defined on the imported VSAM file.
6. Repeat steps a) through e) for each file for which there is imported metadata.
7. Save the CDD by selecting the **File** menu and then clicking **Save.**

14. For the CONNX sample files using the CONNX.VVRR.COPYBOOK prefix, select from this list of member names and CICS file names:

CONNX.VVRR.COPYBOOK

CUSTOMER CNXCSTK

CUSTOMRE CNXCSTE

CUSTOMRR CNXCSTR

EQUIPMNE CNXEQE

EQUIPMNR CNXEQR

EQUIPMNT CNXEQK

ORDER CNXORK

ORDERE CNXORE

ORDERR CNXORR

PRODGRP CNXPGK

PRODGRPE CNXPGE

PRODGRPR CNXPGR

PRODUCT CNXPRDK

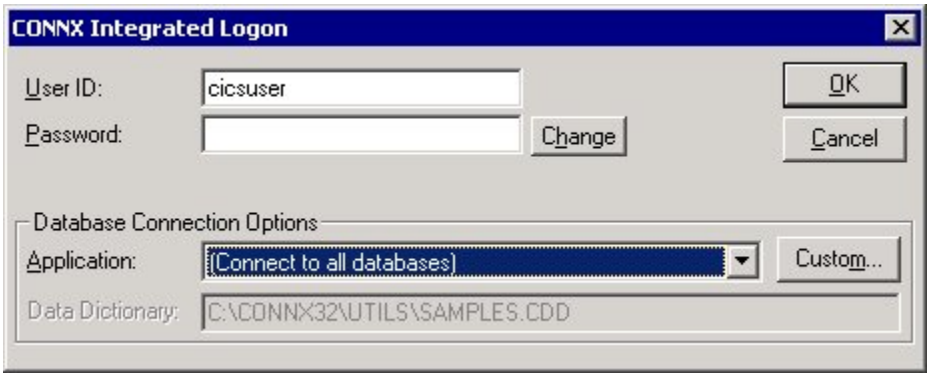
PRODUCTE CNXPRDE

PRODUCTR CNXPRDR

Note: *For your site, specify the COBOL copybook PDS/member and the corresponding CICS file name to import test or production metadata.*

To use the CONNX Integrated Logon with an ODBC-compliant application

If you are using an ODBC-compliant application, such as InfoNaut™, to start a connection to a CONNX ODBC data source, use your CICS user ID and password.



For more information on security for CONNX and IMS, see CONNX Security.

To import from SCT COBOL FD (File Definition) files - MVS-OS/390

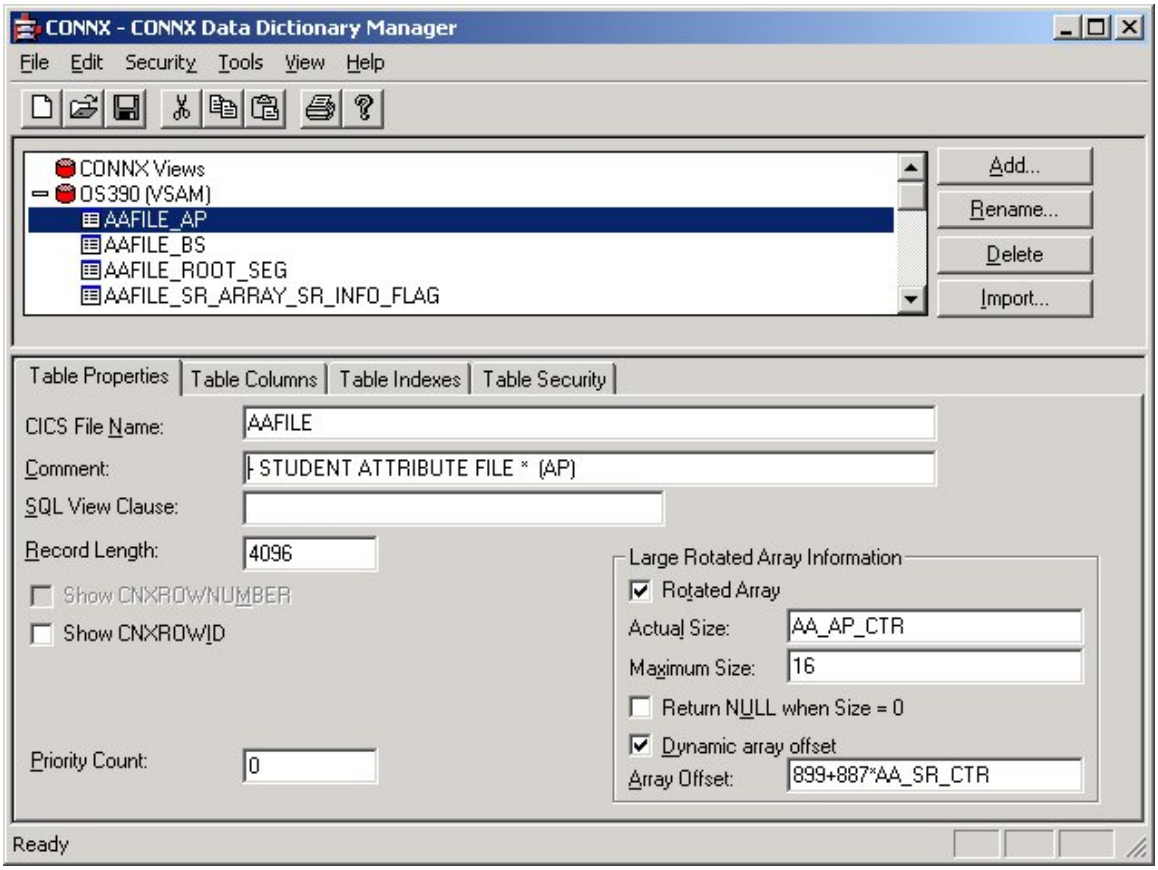
This procedure is site-specific for SCT customers using CONNX for VSAM with either CICS or Started Task. See To import from VSAM COBOL FD (File Definition) files (CICS) - MVS-OS/390 for information on how to verify FTP access.

1. Click the **Import** button in the CONNX CDD Windows Application window.
2. The **Import CDD** dialog box appears. Select **VSAM SCT COBOL FD files** in the **Import Type** list box.

   screen shot
3. Type an SCT COBOL FD partitioned data set and member name in the **Enter an SCT COBOL FD data set member** text box.

   Specify the fully qualified partitioned data set and member name for an SCT file, for example, to import the SCT AAFILE, enter:

   `SCT.SISBASE.SOURCE(ACAARC)` Click the **OK** button.
4. From the list of available files, select each to display the properties, columns, and indexes.



5. Save the CDD by selecting the **File** menu and then clicking **Save**.

   Note: *Partial (for example, AC*) or complete (*) wildcard characters may be used in place of a file name during import.*

   Note: *If CONNX detects more than one record layout when importing, multiple data dictionary*

entries are created.

***Important**: The SCT import rules are defined in SCT Import Rules.*

To perform a VSAM SCT DBD (Database Definition) Overlay Import - MVS-OS/390

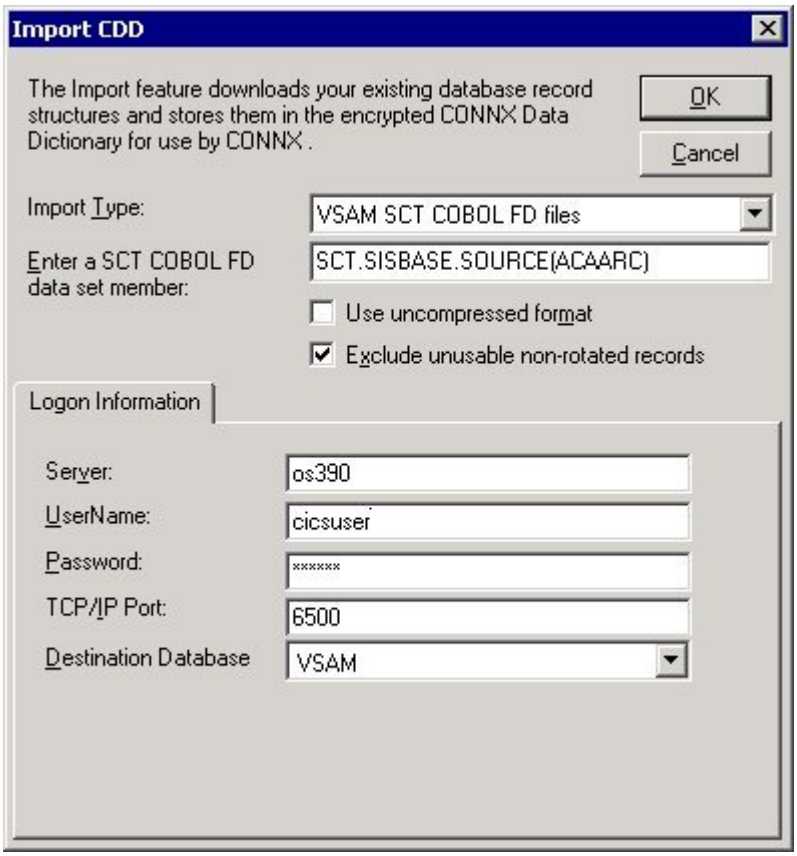
You can overlay the imported COBOL FD field names in your CONNX CDD with the corresponding FOCUS™ DBD names. You can also use the CONNX SCT DBD Overlay Import feature to add comments that correlate to the help screens in your application. Additionally, the comments contain the SCT mnemonic for each field, making it easy to locate a desired field by using the CONNX Find feature.

See To import from VSAM COBOL FD (File Definition) files (CICS) - MVS-OS/390 for information on how to authorize and verify FTP access.

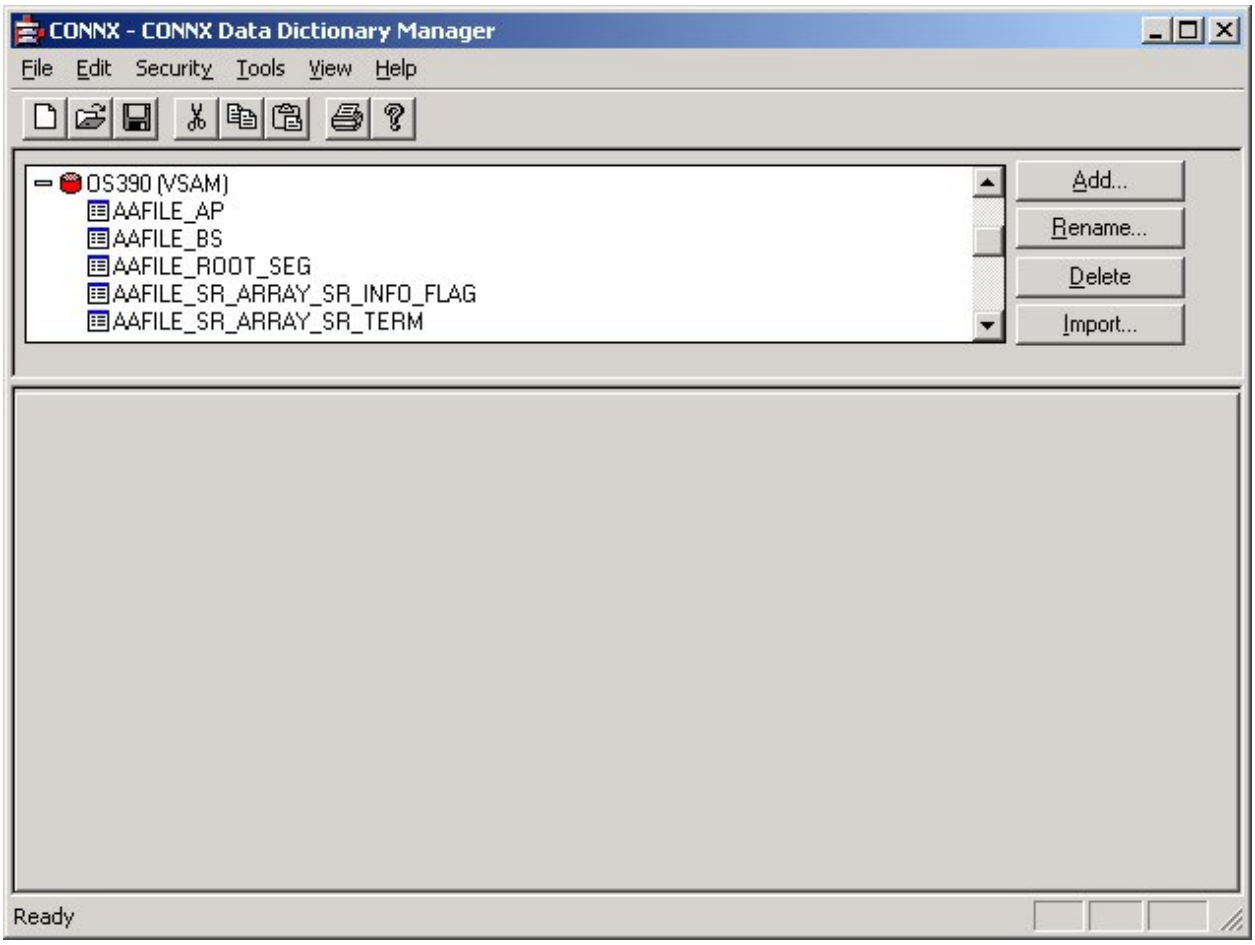
1. Click the **Import** button in the CONNX CDD Windows Application window.
2. The **Import CDD** dialog box appears. Select **VSAM SCT COBOL FD** files in the **Import Type** list box.
3. Type an SCT COBOL FD partitioned data set and member name in the **Enter an SCT COBOL FD data set member** text box. To import the SCT AAFILE, for example, enter:

   SCT.SISBASE.SOURCE(ACAARC)

4. The SIS files from the specified COBOL FD files are imported.

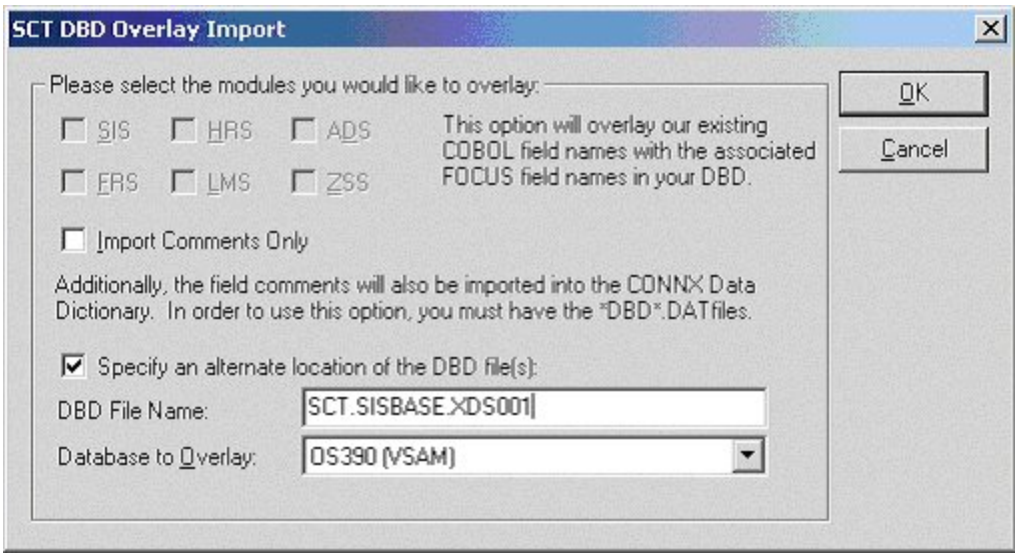


5. Click the **OK** button to import the SCT layouts.
6. The SCT file layouts are imported. The process may take a few minutes depending on the number of files imported.



7. On the **Tools** menu in the CONNX Data Dictionary Manager window, select **SCT DBD Overlay Import.**

8. The **SCT DBD Overlay Import** dialog box appears. Select the **Import Comments Only** check box to add only the DBD comments currently in your CDD. Importing comments only adds the mnemonic file and file identifier and any comments existing for each field in the DBD file.



9. Select the **Specify an alternate location of the DBD file** check box and then type the location in the text box below.

10. It is not necessary to have the "DBD".DAT files in order to import from an OS/390 system. Given the default SCT installation prefixes, the SCT Focus DBD sequential overlay files are:

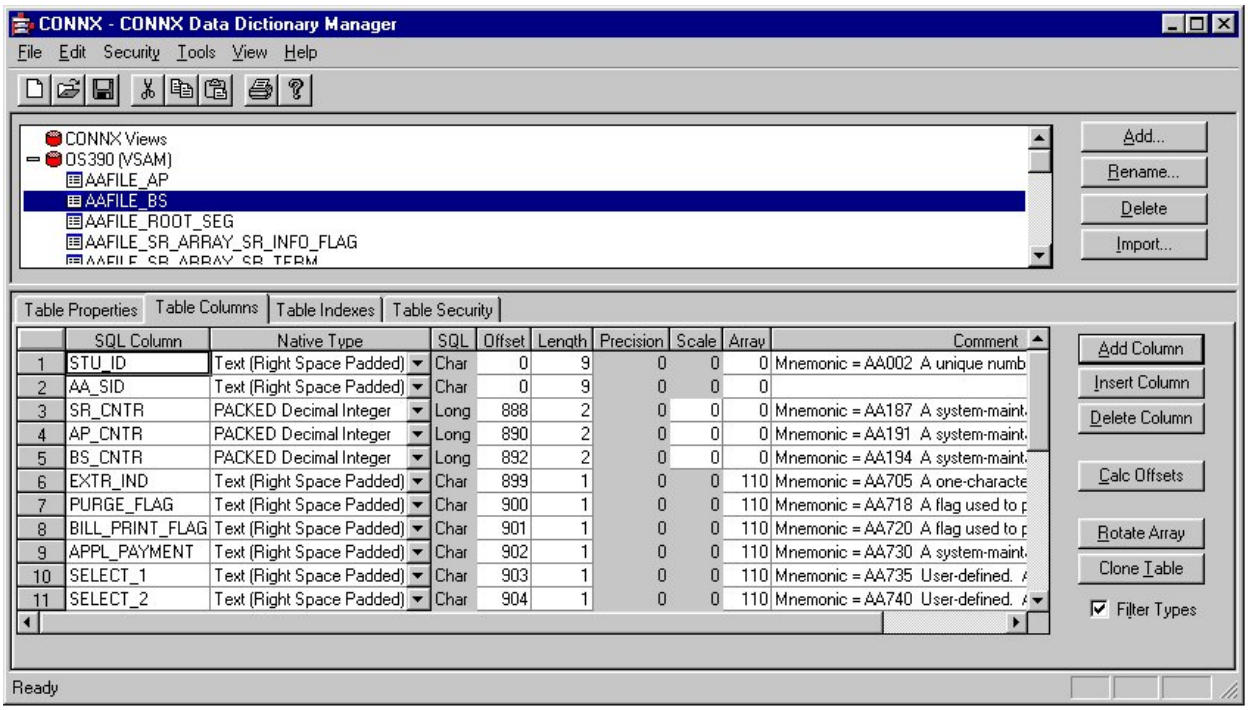
SCT.ADSBAS.XEDS004	Alumni Development System
SCT.FRSBASE.XDS002	Financial Records System
SCT.HRSBASE.XDS003	Human Resource System
SCT.LMSBASE.XDS005	Loan Management System
SCT.SISBASE.XDS001	Student Information System
SCT.ZSSBASE.XDS000	Z Support Software System

11. Select the VSAM database to overlay in the **Database to Overlay** list box.

12. Click the **OK** button. Note that SIS modules are used for this example.

13. The conversion to FOCUS field names may take a few minutes, depending on the number of files imported.

14. Once the files are converted, select a table from the list in the upper pane. The COBOL field names in the selected modules are converted to FOCUS field names. Note that comments containing a mnemonic code for each file and field are inserted, and that the field names have changed.



***Note**: Partial (for example AC*) or complete (*) wildcard characters may be used in place of a file name during import.*

Importing Remote VSAM Files

Background and Prerequisites

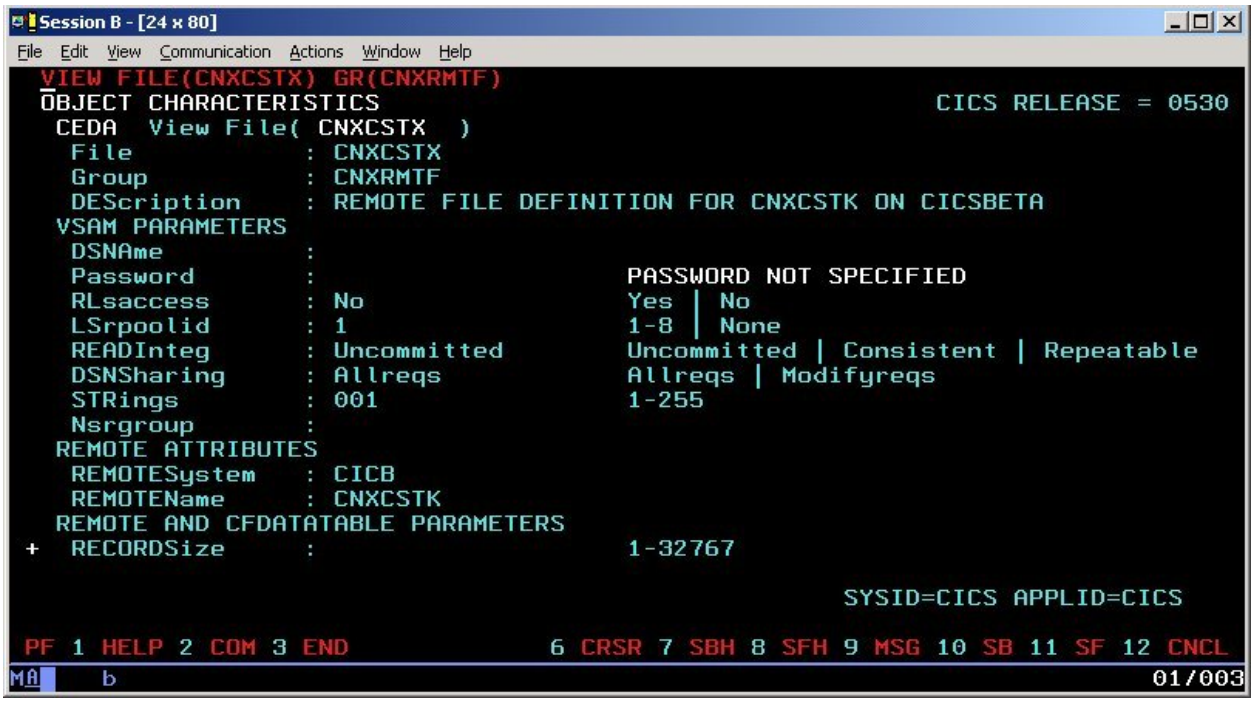
CONNX for CICS/VSAM supports CICS remote file access for VSAM Key-Sequenced Data Sets (KSDS). When two or more CICS regions are configured to communicate using either intersystem communication (ISC) or multiregion operation (MRO), the CONNX CICS / VSAM programs installed on one CICS region can access CICS remote files defined on separate CICS regions.

Example:

CONNX for CICS/VSAM is installed on region CICS. An MRO link is defined to another region, CICSBETA, which is started on the same OS/390 system. Two instances of the CONNX sample VSAM KSDS Customer file are defined on region CICS; the first (CNXCSTK) is local; the second (CNXCSTX) is remote and refers to a copy of the CNXCSTK file defined on region CICSBETA.

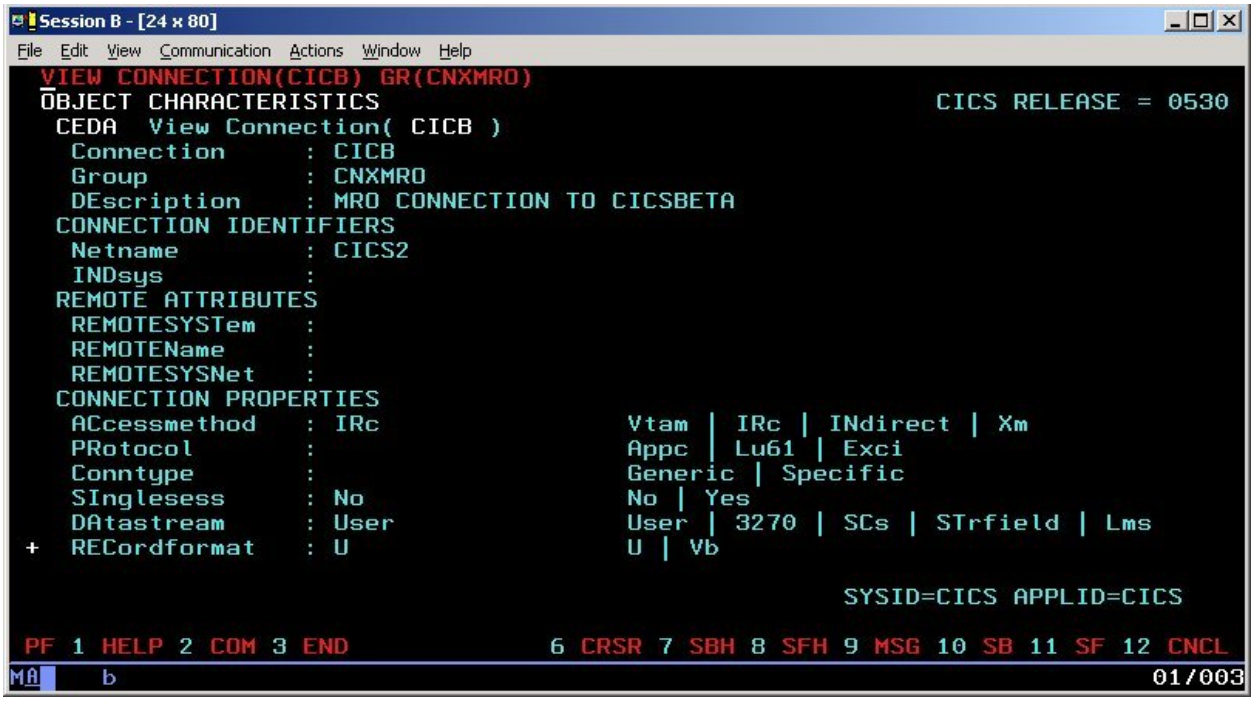
Remote File Definition

The screen shot below demonstrates how the remote file CNXCSTX is defined to the local CICS RDO file:



The remote attributes map file CNXCSTX to file CNXCSTK on remote system CICB, which is defined as an MRO link to region CICSBETA:

MRO Link Definition



For complete information on how to set up communications between multiple CICS regions, refer to the following IBM manual *CICS Intercommunication Guide*

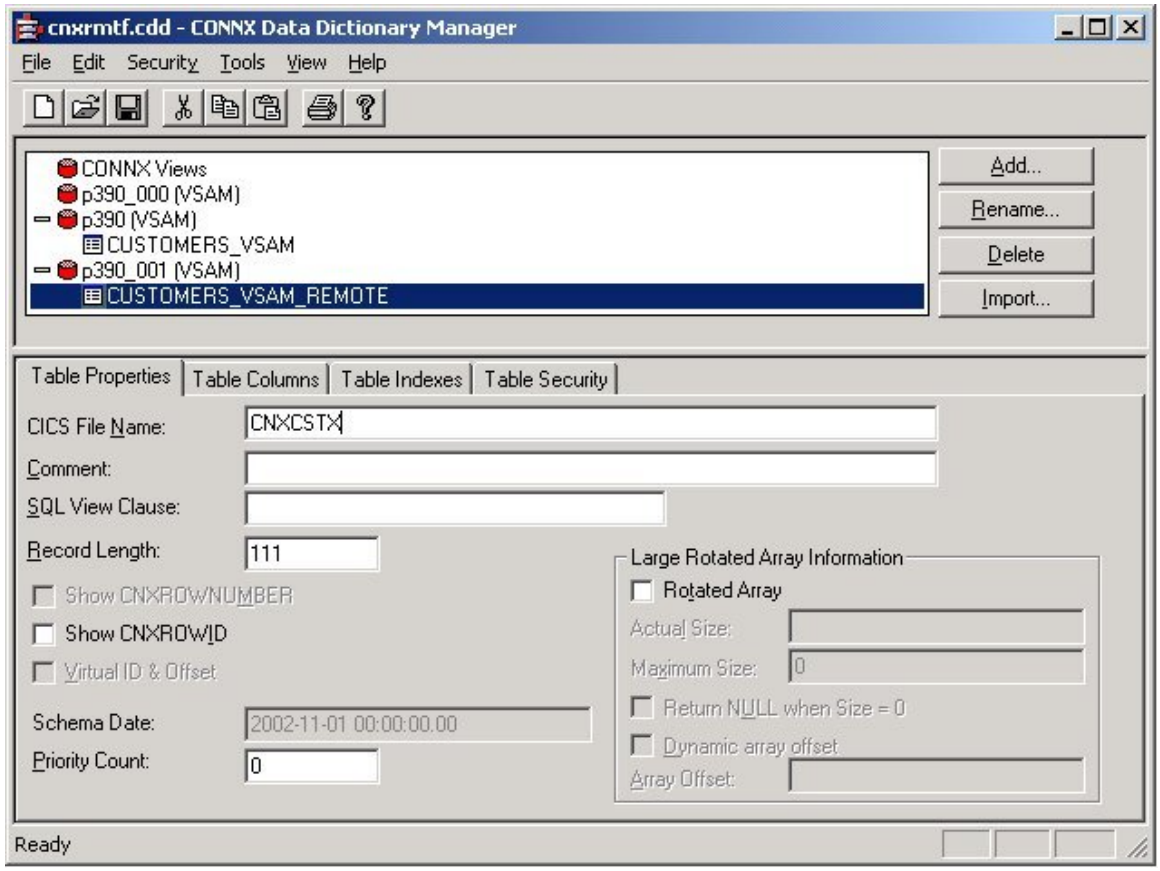
Document Number: SC33-1695-34

To import remote VSAM files without matching local files from COBOL FD files (CICS) - MVS-OS/390

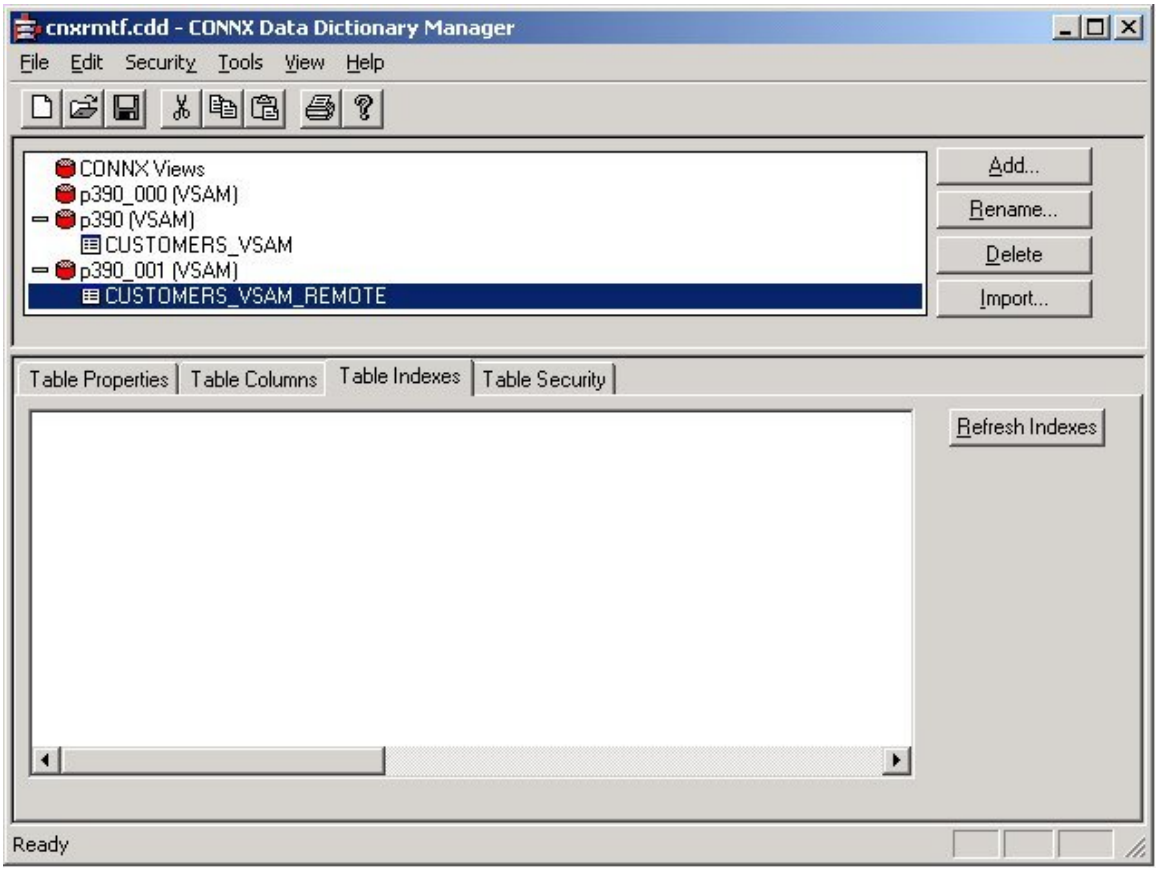
1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears.
3. Select **VSAM COBOL FD files** in the **Import Type** list box.
4. Type a CICS user name and password in the **User Name** and **Password** text boxes.
5. The TCPIP port number is set to 6500 by default, but can be configured via the CONNX NX01 CICS transaction. See "To convert the CONNX port number to the default" in the CONNX Installation Guide for configuration information.
6. Select a Destination Database from the list box. See Add a Database Connection for more information.
7. Type a COBOL FD partitioned data set and member name in the **Enter a COBOL FD data set member** text box.
8. Specify the fully qualified PDS (partitioned data set) and member name for the COBOL FD, for example, CONNX.VVVRR.COPYBOOK(CUSTOMRX).
9. A COBOL FD specification does not contain the corresponding CICS File Control Table (FCT) name; therefore, it must be specified on the Table Properties tab in the CONNX Data Dictionary Manager window when the data is returned.
1. All of the record layouts in the specified file are imported.
2. No additional logon information is required.

10. Select each file to import and follow these steps:

1. Select the VSAM file definition in the upper pane (CUSTOMERS_VSAM_REMOTE in this example) and click the **Table Properties** tab in the CONNX Data Dictionary Manager window.
2. Type the **remote CICS RDO or FCT name** for the file in the **CICS File Name** text box. For example, the CICS short file name for the remote CONNX VSAM KSDS sample customer file is CNXCSTX.



3. Tab out of the CICS File Name text box to display the **CONNX Database Logon** dialog box. Click the **OK** button.
4. Click the **Table Indexes** tab to display the key information for the imported CICS VSAM file.



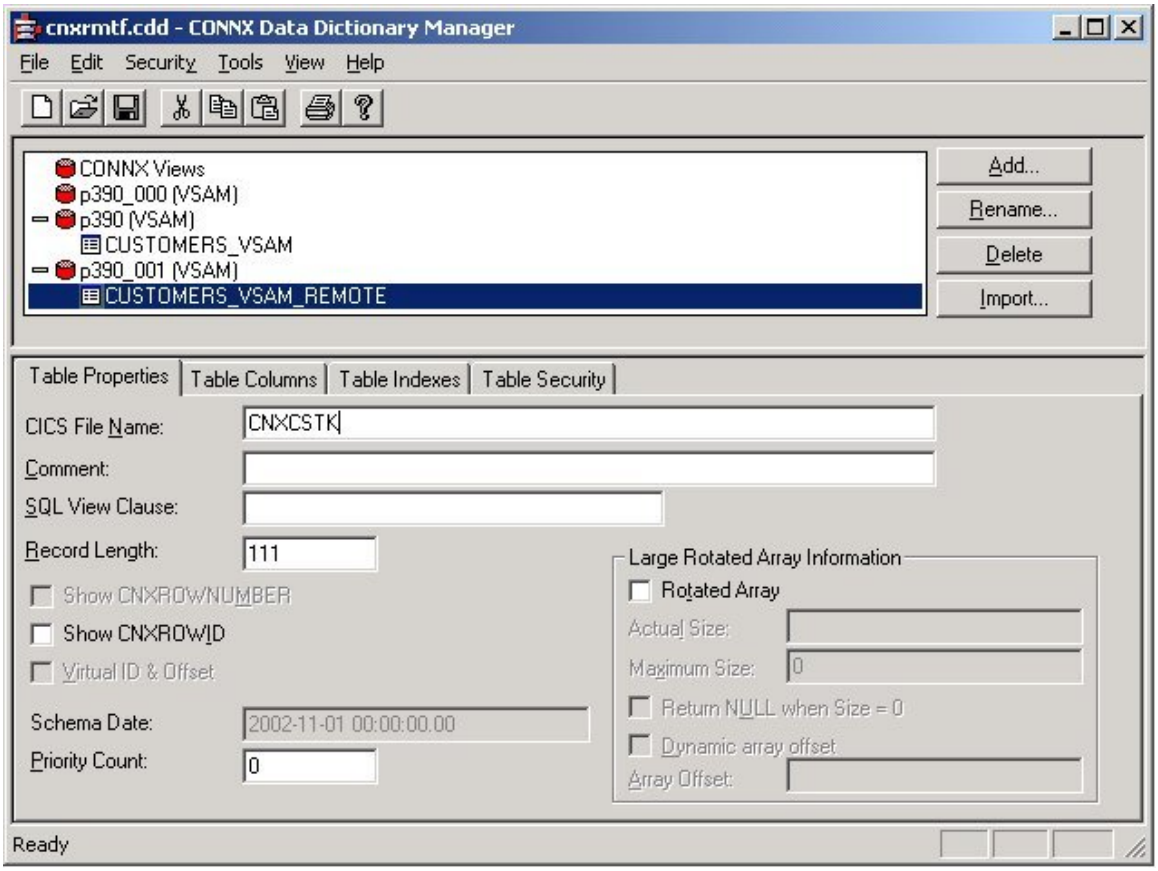
11. The definition of the remote Customer VSAM KSDS files to the CONNX Data Dictionary file is complete. SQL queries against CUSTOMERS_VSAM_REMOTE are mapped to remote file CNXCSTX.

To import remote VSAM files with matching local VSAM files from COBOL FD (File Definition) files (CICS) - MVS-z/OS

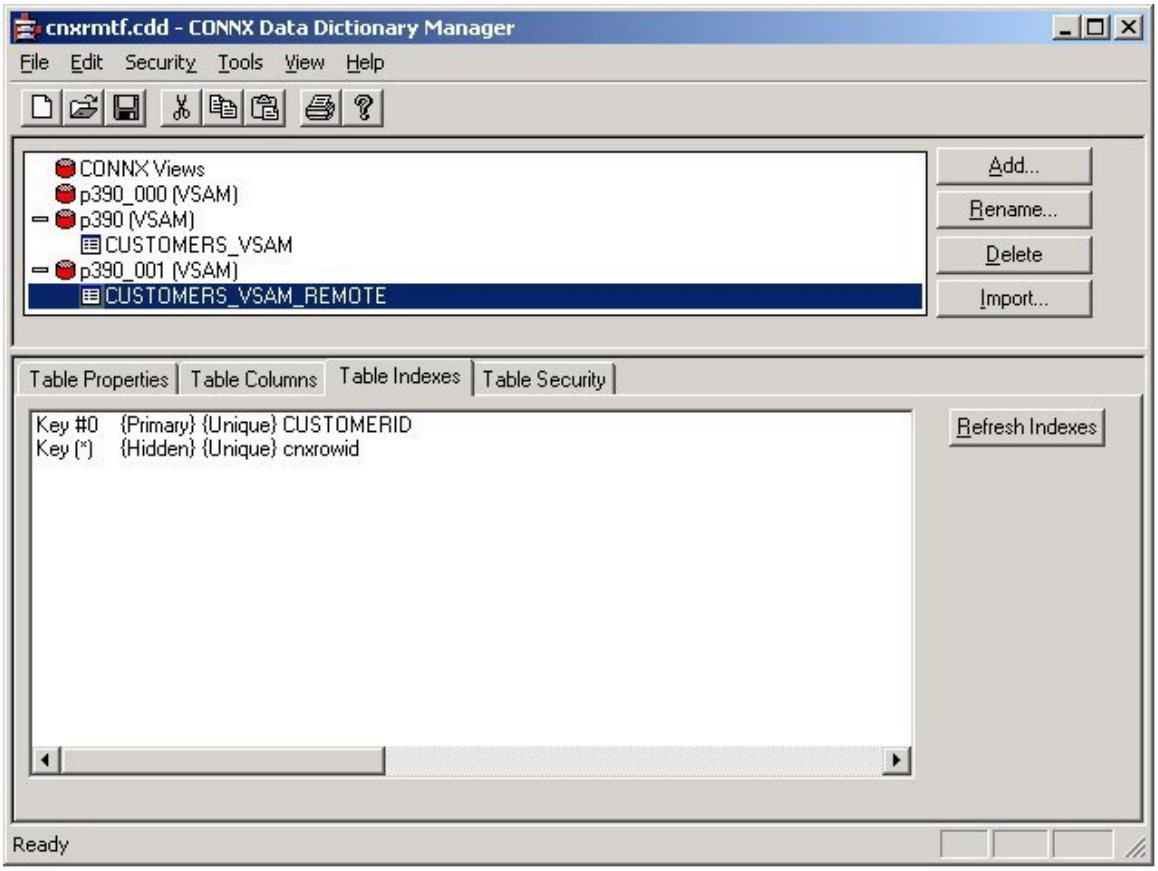
In the current example, the local and remote CONNX sample VSAM Customer KSDS CICS file definitions (CNXCSTK and CNXCSTX) point to different physical VSAM datasets with identical file attributes, record counts, and formats. Note that, in the above procedure, no indexes were imported for the CUSTOMERS_VSAM_REMOTE KSDS file (CNXCSTX). CONNX supports SQL requests against remote files without matching local files, but whenever a matching local file exists, importing the indexes from the local file (CNXCSTK in the above example) improves performance.

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears.
3. Select **VSAM COBOL FD files** in the **Import Type** list box.
4. Type a CICS user name and password in the **User Name** and **Password** text boxes.
5. The TCPIP port number is set to 6500 by default, but can be configured via the CONNX NX01 CICS transaction. See "To convert the CONNX port number to the default" in the CONNX Installation Guide for configuration information.
6. Select a Destination Database from the list box. See Add a Database Connection for more information.
7. Type a COBOL FD partitioned data set and member name in the **Enter a COBOL FD data set member** text box.
8. Specify the fully qualified PDS (partitioned data set) and member name for the COBOL FD, for example, CONNX.VVRR.COPYBOOK(CUSTOMRX). A COBOL FD specification does not contain the corresponding CICS File Control Table (FCT) name; therefore, it must be specified on the **Table Properties** tab in the CONNX Data Dictionary Manager window when the data is returned.
1. All of the record layouts in the specified file are imported.
2. No additional logon information is required.

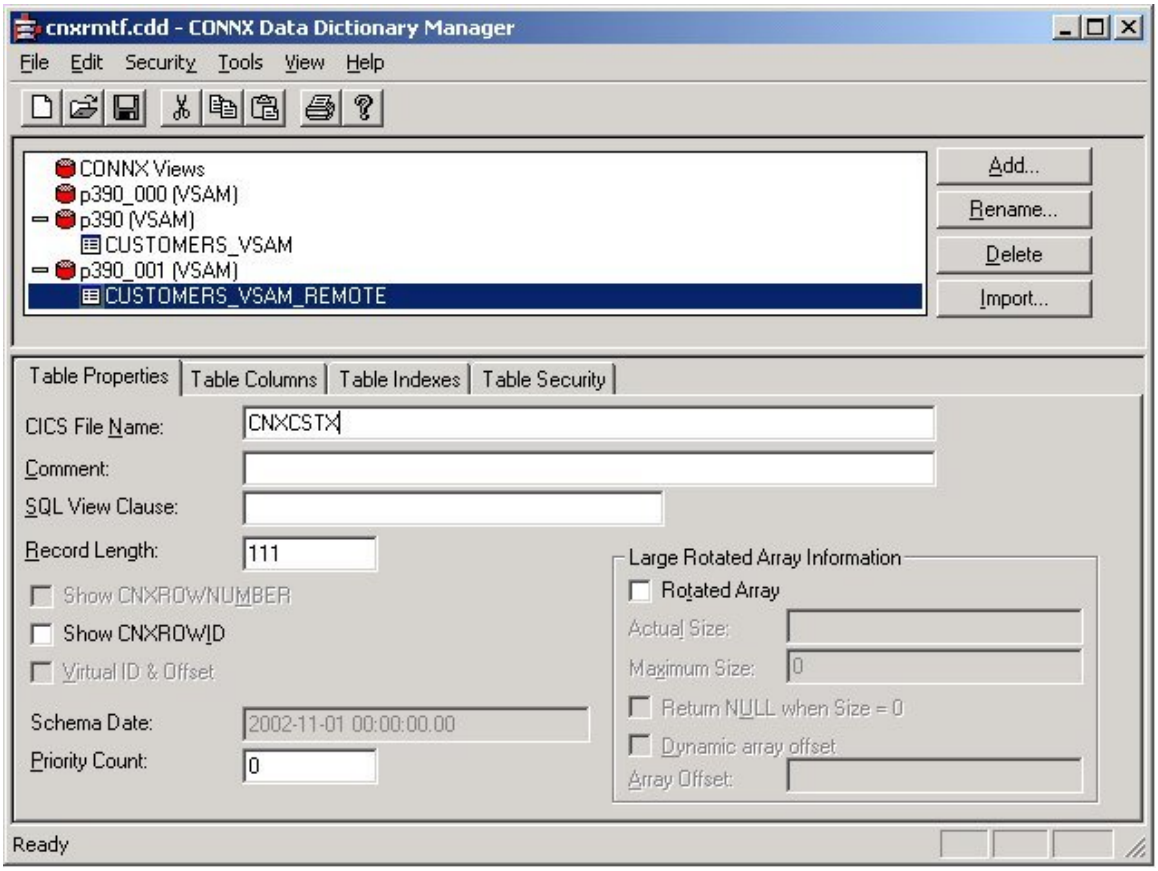
9. Select each file to import and follow these steps:
1. Select the VSAM file definition in the upper pane (CUSTOMERS_VSAM_REMOTE in this example) and click the **Table Properties** tab in the CONNX Data Dictionary Manager window.
2. Type the matching local CICS RDO or FCT name for the remote file in the **CICS File Name** text box. For example, the local CICS short file name for the CONNX VSAM KSDS sample customer file is CNXCSTK. In this example, both the local and the remote Customers VSAM files are defined as KSDS files with identical attributes, but only the local CICS short file name contains complete information such as record length, key offset, and key length. To import this information into the CONNX CDD, key in the matching local CICS short file name (CNXCSTK):



3. Tab out of the CICS File Name text box to display the CONNX Database Logon dialog box. Click the **OK** button.
4. Click the **Table Indexes** tab to display the key information for the imported CICS VSAM file.



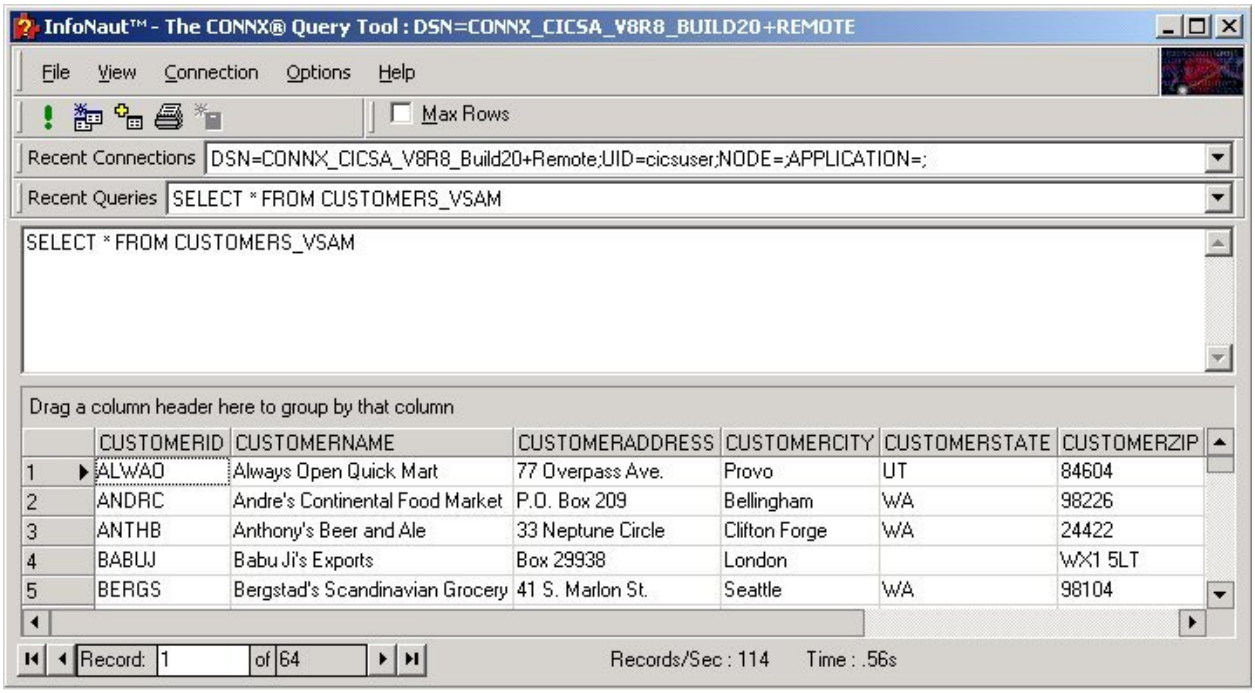
5. Finally, select the **Table Properties Tab,** change the CICS short file name to the remote file name (CNXCSTX), and save the CDD file without tabbing out of the **CICS File Name** text box:

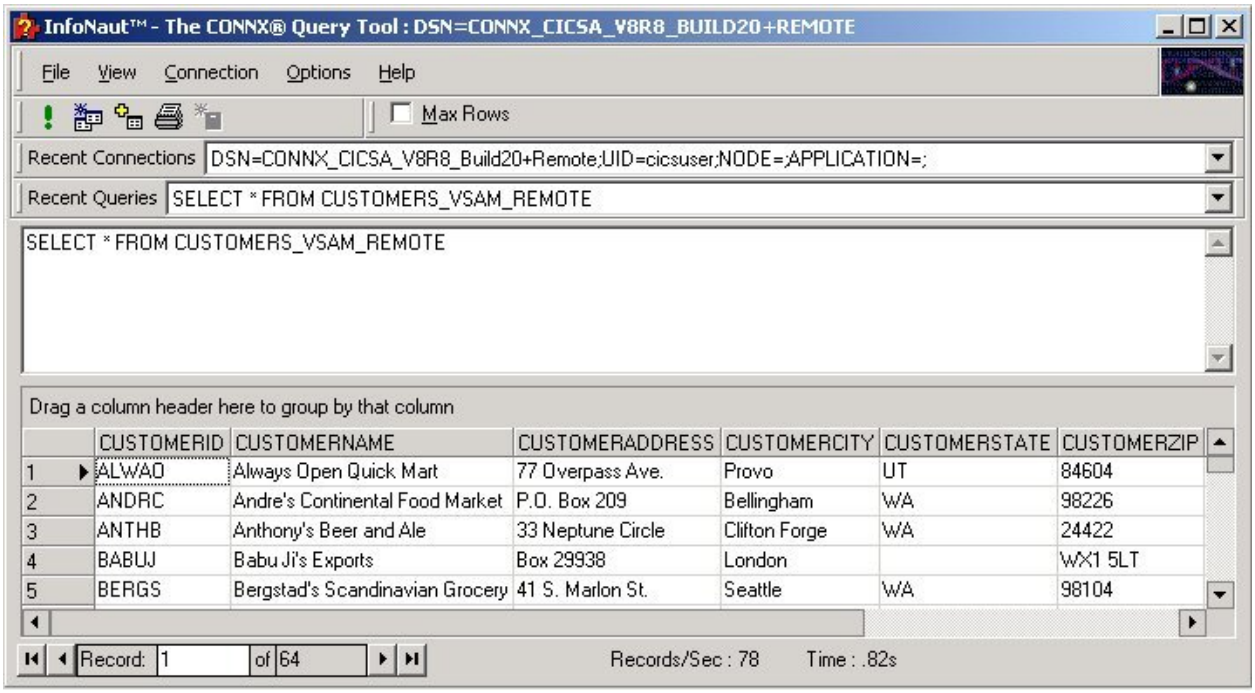


10. The definition of the remote Customer VSAM KSDS file using the matching local file attributes is complete. SQL queries against CUSTOMERS_VSAM_REMOTE are mapped to remote file CNXCSTX.

Local and Remote File Query Performance

The following screen shots of the CONNX InfoNaut Query application demonstrate differences in response time for SQL queries against identical local and remote CICS VSAM KSDS files:





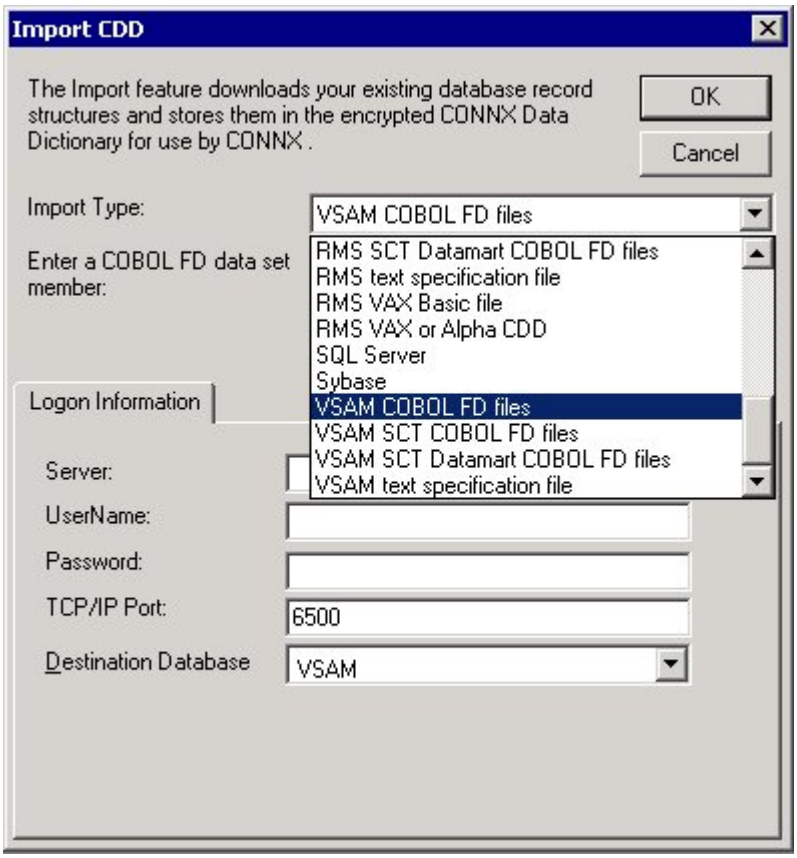
As the records per second counts (114 vs. 78) demonstrate, the same SQL query executed against identical local and remote VSAM KSDS files runs faster against the local file. CONNX SQL query performance against remote VSAM KSDS files can be optimized by importing index information from corresponding local files, if available. Once the key information is provided for remote files, CONNX executes the same file traversal logic for both local and remote files. The remaining differences in

performance measures such as records per second derive from the MRO or ISC link overhead between the CICS regions.

CONNX and VSAM - QSAM/PDS

To import from VSAM COBOL FD files - Started Task

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears.



3. Select VSAM COBOL FD files in the **Import Type** list box.
4. In the **Enter a COBOL FD data set member** text box, enter **CONNX.VVRR.STASK.COPYBOOK.(CUSTOMER).**
5. Specify the fully qualified PDS (partitioned data set) and member name for the COBOL FD, for example, CONNX.VVRR.STASK.COPYBOOK(CUSTOMER). A COBOL FD specification does not contain the corresponding File Control Table (FCT) name or Started Task DD (Data Definition) name; therefore, it must be specified on the **Table Properties** tab in the CONNX Data Dictionary Manager window when the data is returned.
1. All of the record layouts in the specified file are imported.
2. No additional logon information is required.

6. In the **Server** text box, enter the symbolic or dotted numeric TCP/IP address of the target host, and then enter your TSO user name and password. Enter a TCP/IP port number, and then click the **OK** button.

7. Type a user name in the **User Name** text box.

8. The user ID must be authorized for FTP access to the target host, and must contain a RACF OMVS segment. For example, the default user ID USER can be modified via RACF commands to permit FTP access by adding an OMVS segment:

```
USER=USER  NAME=USER
OMVS INFORMATION
________

UID= 0000000000
CPUTIMEMAX= NONE
ASSIZEMAX= NONE
FILEPROCMAX= NONE
PROCUSERMAX= NONE
THREADSMAX= NONE
MMAPAREAMAX= NONE
```

9. The userid used for CONNX CDD imports need not be authorized for TSO access, as long as it has read authority to the target COBOL copybook partitioned data sets. By default, the CONNX COBOL copybooks reside in CONNX.VVRR.STASK.COPYBOOK.(CUSTOMER).

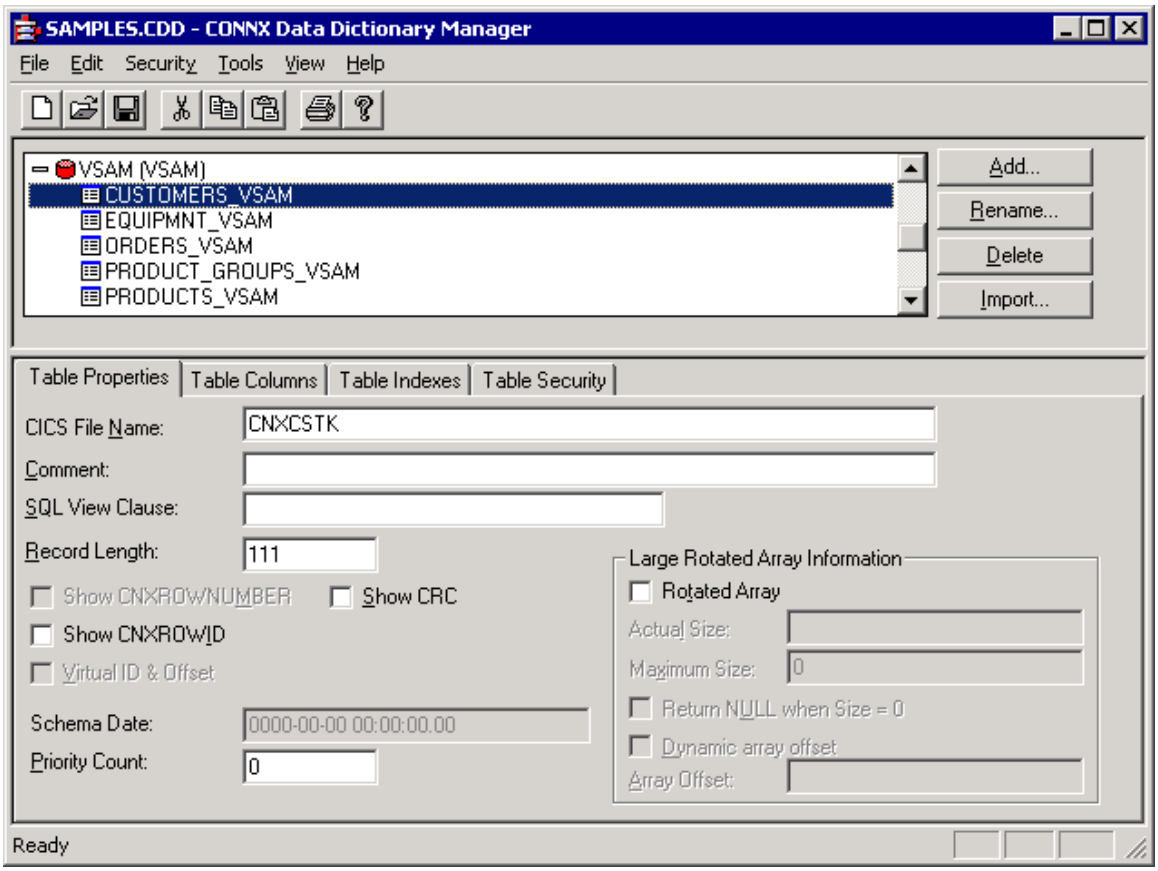
10. Type a password in the **Password** text box.

11. The **TCPIP port number** is set to 6500 by default, but can be configured via the CONNX NX01 transaction. See "To convert the CONNX port number to the default" in the CONNX Installation Guide for configuration information.

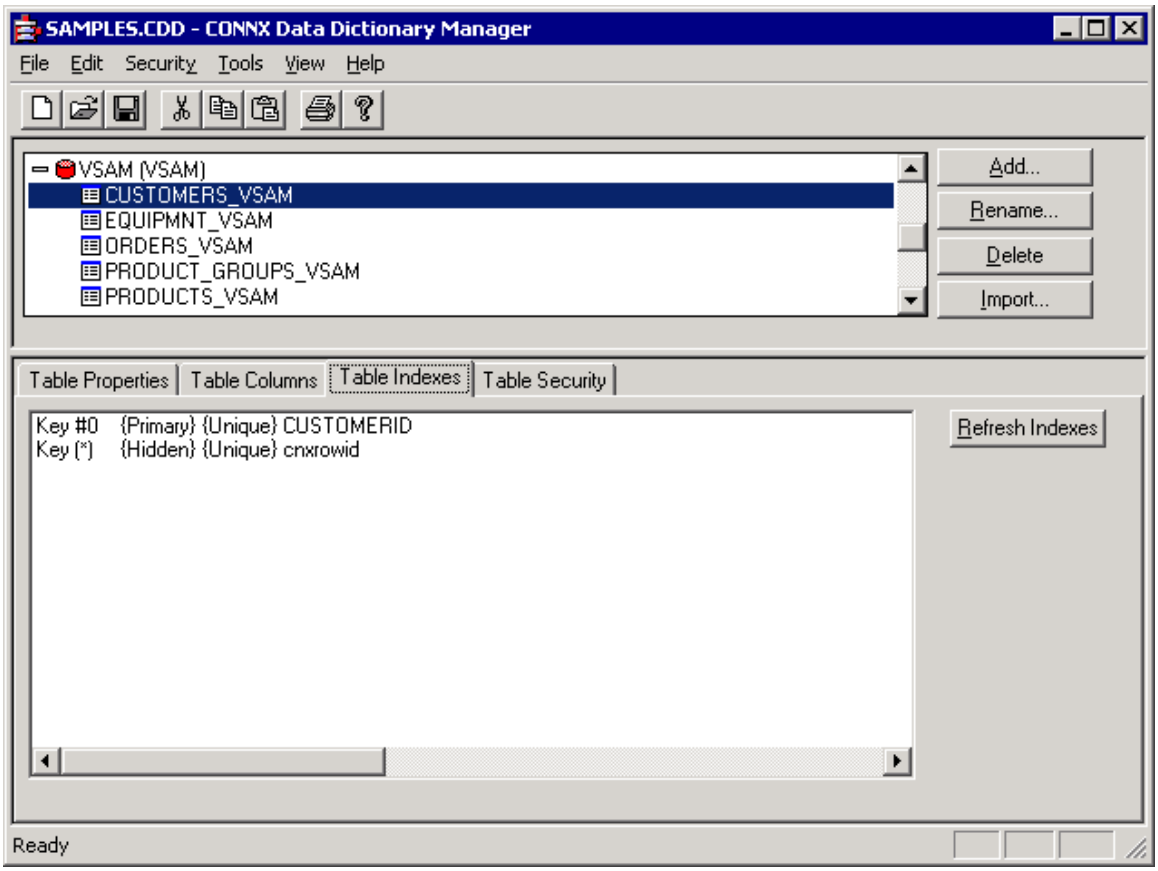
12. Select a **Destination Database** for the imported tables. See Adding a Database Connection for more information.

13. Select each file to import and follow these steps:

    1. Click the **Table Properties** tab in the CONNX Data Dictionary Manager window.
    2. Type the name for the file in the **File Name** text box. For example, the short file name for the CONNX VSAM KSDS sample customer file is CNXCSTK.



3. Tab out of the File Name text box to display the **CONNX Database Logon** dialog box. Click the **OK** button.
4. Click the **Table Indexes** tab to display the key information for the imported VSAM file.
5. If the Table Index information list box is empty, click the **Refresh Indexes** button. If the list box remains empty, no indexes are defined on the imported VSAM file.



6. Repeat steps a) through e) for each file for which there is imported metadata.
7. Save the CDD by selecting the **File** menu and then clicking **Save.**

14. For the CONNX sample files using the CONNX.VVRR.STASK. COPYBOOK prefix, select from this list of member names and file names:

CONNX.VVRR.STASK.COPYBOOK Member Name	File Name
CUSTOMER	CNXCSTK
CUSTOMRE	CNXCSTE
CUSTOMRR	CNXCSTR
EQUIPMNE	CNXEQE
EQUIPMNR	CNXEQR
EQUIPMNT	CNXEQK
ORDER	CNXORK
ORDERE	CNXORE
ORDERR	CNXORR
PRODGRP	CNXPGK
PRODGRPE	CNXPGE
PRODGRPR	CNXPGR

PRODUCT	CNXPRDK
PRODUCTE	CNXPRDE
PRODUCTR	CNXPRDR

Note: *For your site, specify the COBOL copybook PDS/member and the corresponding file name to import test or production metadata.*

CONNX and VSAM - VSE

To import from VSAM COBOL FD (File Definition) files (CICS) - VSE

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears.
3. Select **VSAM COBOL FD** files in the **Import Type** list box.
4. Type a fully qualified library.sublib.member.type name in the **Enter a COBOL FD data set member text box**, for example, **cnxvvrr.copybook.customer.c**

   The COBOL copybook to be imported must reside in a library.sublib and the library must be defined to the TCP/IP file system. This can be done via operating system console commands or via the IPINIT00 batch job. The following examples assume that the sample CONNX COBOL copybooks reside in VSAM-managed library CNXVVRR:

1. **Via VSE console commands,** as follows:

```
msg f7
   AR 0015 1I40I  READY
F7-0102 IPN300I Enter TCP/IP Command
102 def file,pub=CNXV8R8,dlbl=CNXV8R8,type=lib
F7 0100 IPN264I File defined, Dataset: CNXV8R8
```

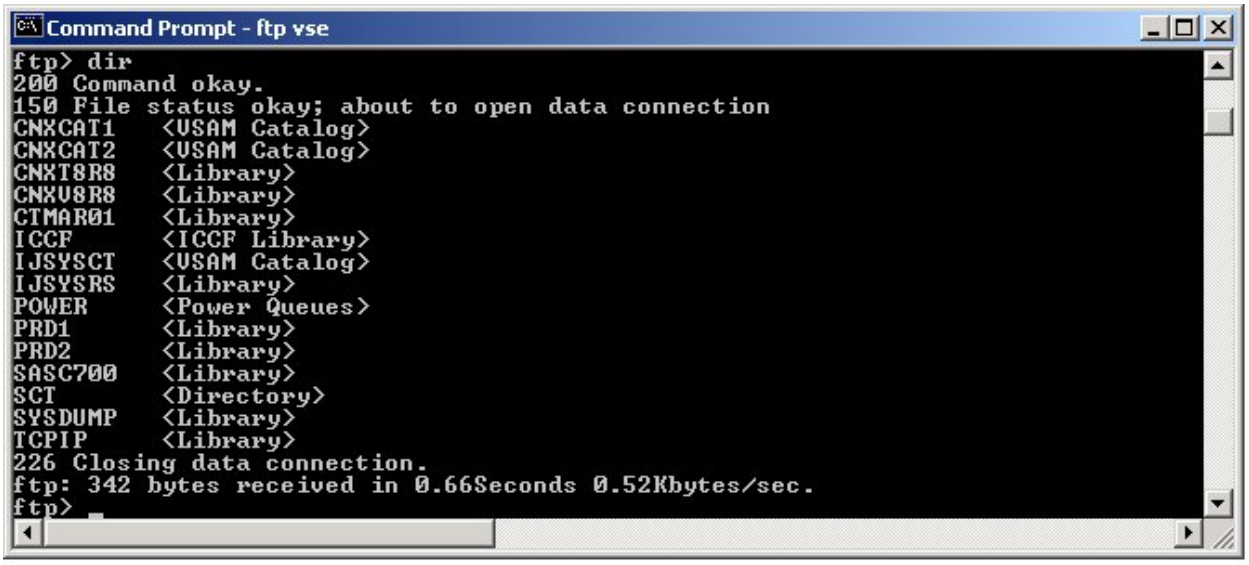
2. **Via batch JCL:** Add the following control record to the IPINIT00 member as discussed above, submit the batch job, and stop/restart TCP/IP:

```
DEFINE FILE,PUBLIC='CNXV8R8', DLBL=CNXV8R8, TYPE=LIBRARY,READONLY=NO
```

5. After defining the CONNX library to TCP/IP, you can import from a COBOL copybook by specifying the fully-qualified library.sublib.member.type name, for example:

```
CNXVVRR.COPYBOOK.CUSTOMER.C
```

6. To verify the entries currently defined for FTP access to the VSE TCP/IP job, start an FTP session using the Windows command prompt, and enter the **dir** command:



7. A COBOL FD specification does not contain the CICS FCT (File Control Table) name; therefore, it must be specified on the Table Properties tab in the CONNX Data Dictionary Manager when the data is returned.

1. All of the record layouts in the specified file are imported.

2. No additional logon information is required.

8. Type a CICS user name in the **User Name** text box.

The CICS user ID must be authorized for FTP access to the target host. The most common way to authorize a VSE userid for TCP/IP is to hardcode it in the TCP/IP initialization data set. For example, edit member IPINIT00 in the TCP/IP ICCF library and add the appropriate user ID and password:

```
DEFINE USER, ID=CONX, PASSWORD=CONNXVSE
```

9. Then submit job IPINIT00 to the POWER reader queue.

10. Additional security strategies are documented in the following manuals published by Connectivity Systems Incorporated:

TCP/IP for VSE Commands Version 1 Release 4

TCP/IP for VSE Installation Guide Version 1 Release 4

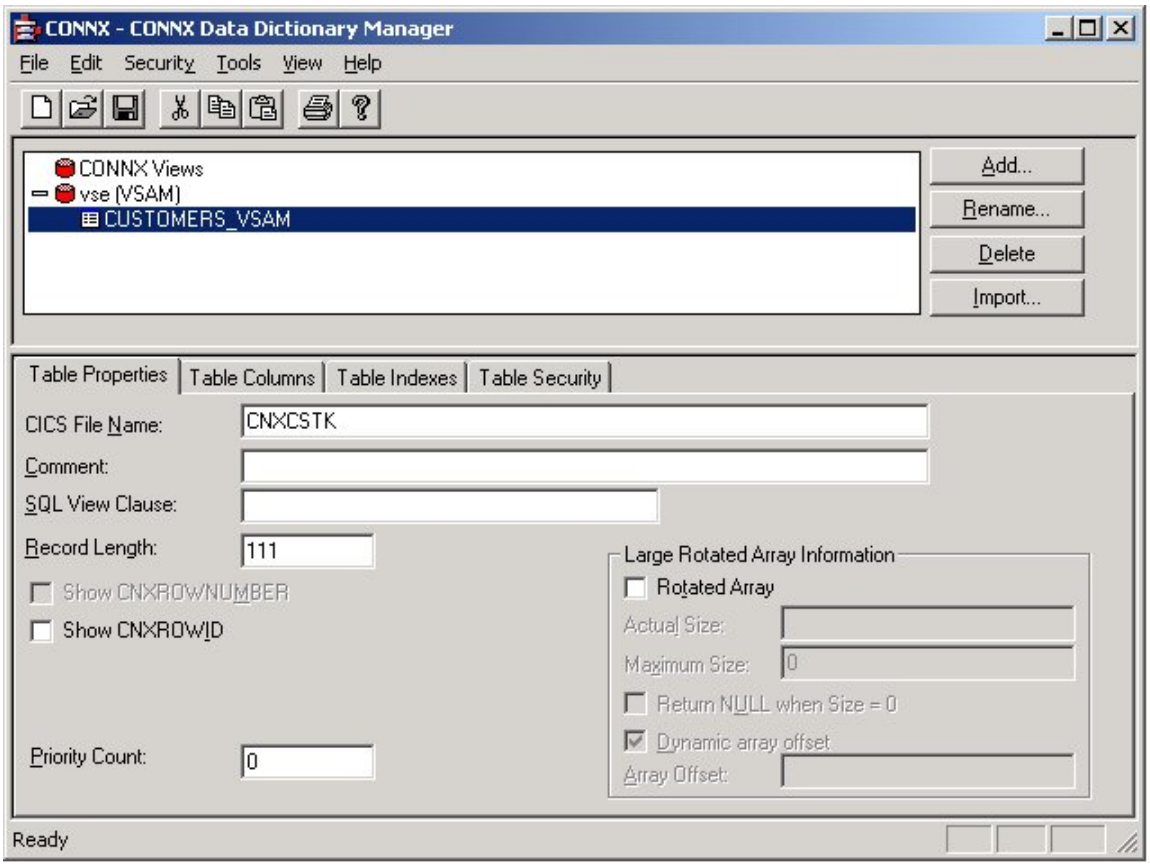
11. Type a **CICS password** in the **Password** text box.

12. The TCPIP port number is set to 6500 by default, but can be configured via the CONNX NX01 CICS transaction. See the CONNX Configuration Manager folder under CONNX Registry File Settings for configuration information.

13. Select a **Destination Database** for the imported tables. See Adding a Database Connection for more information.

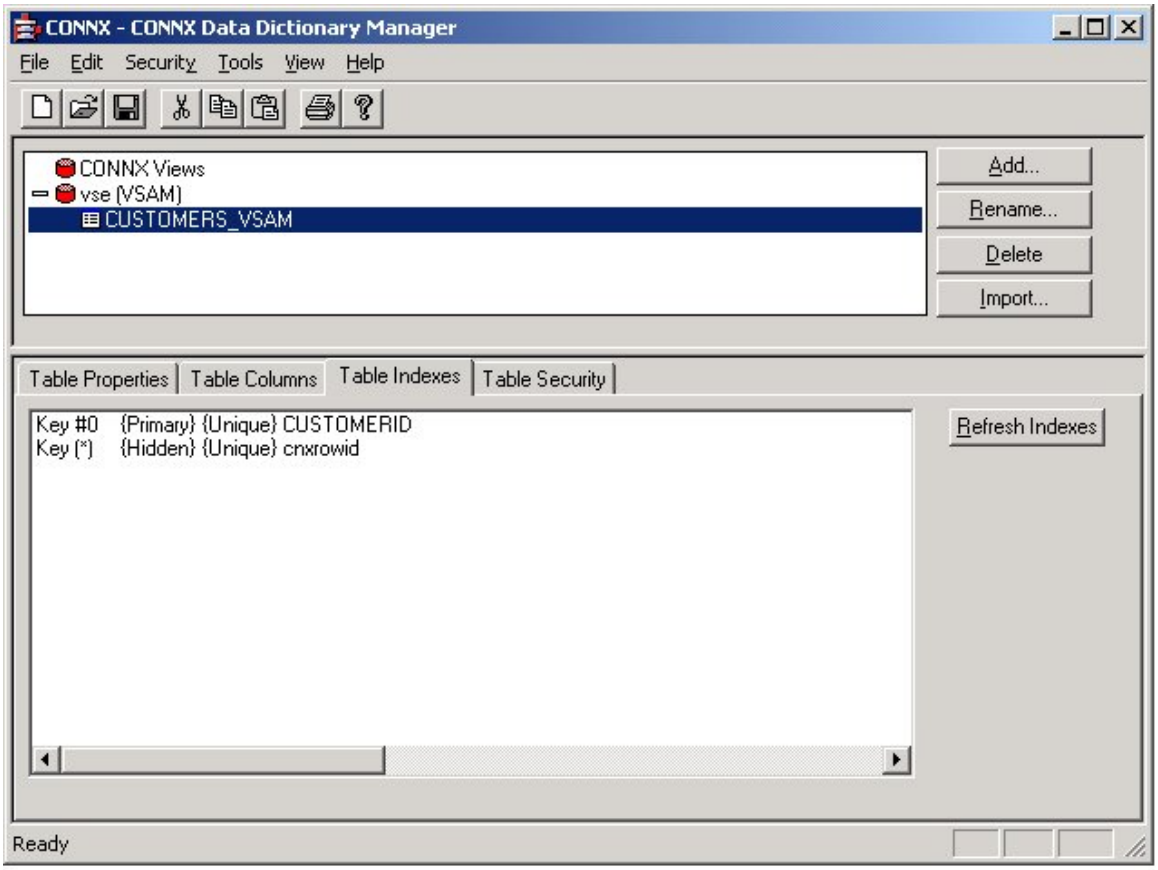
14. Select each file to import and follow these steps:

1. Click the **Table Properties** tab in the CONNX Data Dictionary Manager window.
2. Type the **CICS FCT** name for the file in the **CICS File Name** text box. For example, the CICS short file name for the CONNX VSAM KSDS sample customer file is CNXCSTK.



3. Tab out of the **CICS File Name** text box to display the **CONNX Database Logon** dialog box. Click the **OK** button.

4. Click the **Table Indexes** tab to display the key information for the imported CICS VSAM file.



5. If the Table Index information list box is empty, click the **Refresh Indexes** button. If the list box remains empty, no indexes are defined on the imported VSAM file.
6. Repeat steps a) through e) for each file for which there is imported metadata.
7. Save the CDD by selecting the **File** menu and then clicking **Save.**

15. For the CONNX sample files using the CNXVVRR.COPYBOOK library.sublibrary, select from this list of member names and CICS file names:

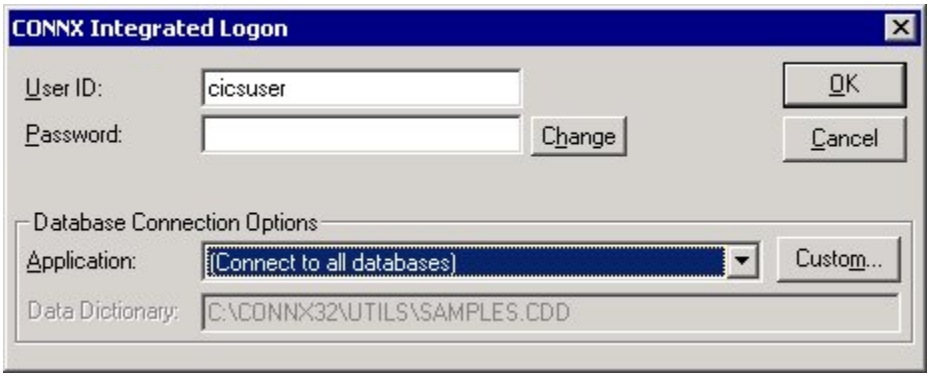
CNXVVRR.COPYBOOK library.sublibrary Member Names	CICS File Name
CUSTOMER.C	CNXCSTK
CUSTOMRE.C	CNXCSTE
CUSTOMRR.C	CNXCSTR
EQUIPMNE.C	CNXEQE
EQUIPMNR.C	CNXEQR
EQUIPMNT.C	CNXEQK
ORDER.C	CNXORK
ORDERE.C	CNXORE
ORDERR.C	CNXORR

PRODGRP.C	CNXPGK
PRODGRPE.C	CNXPGE
PRODGRPR.C	CNXPGR
PRODUCT.C	CNXPRDK
PRODUCTE.C	CNXPRDE
PRODUCTR.C	CNXPRDR

Note: *For your site, specify the COBOL copybook library.sublib.member.type and the corresponding CICS file name to import test or production metadata.*

To use the CONNX Integrated Logon with an ODBC-compliant application

If you are using an ODBC-compliant application, such as InfoNaut™, to start a connection to a CONNX ODBC data source, use your CICS user ID and password.



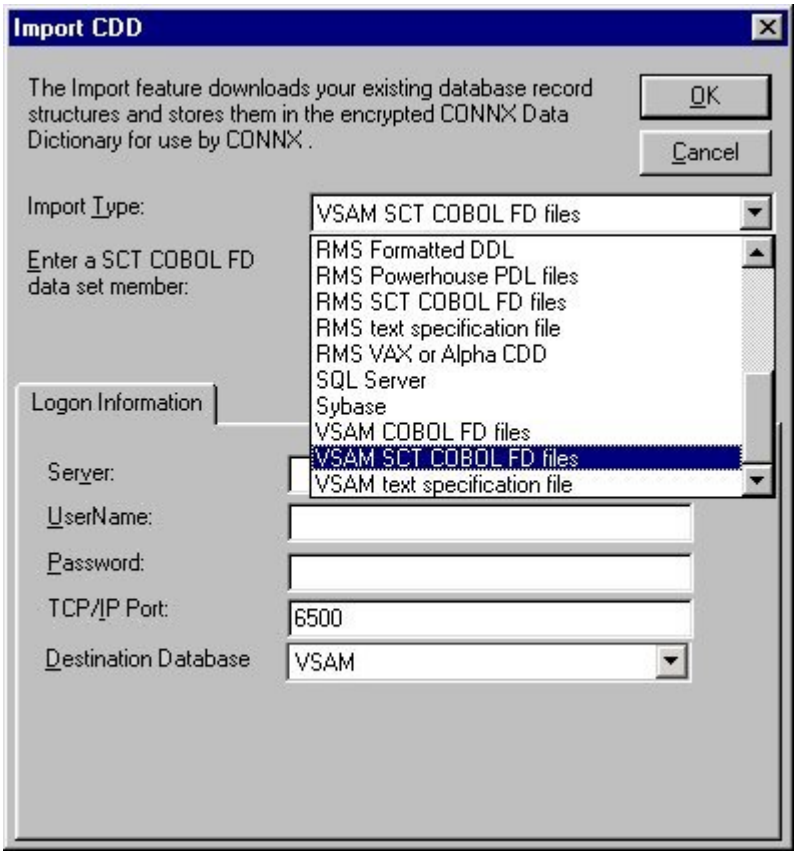
For more information on security for CONNX and VSAM, see CONNX Security.

For more information on InfoNaut, visit the help file in both the InfoNaut product and at http://www.connx.com/products/connx/InfoNaut/CONNX_Infonaut.htm

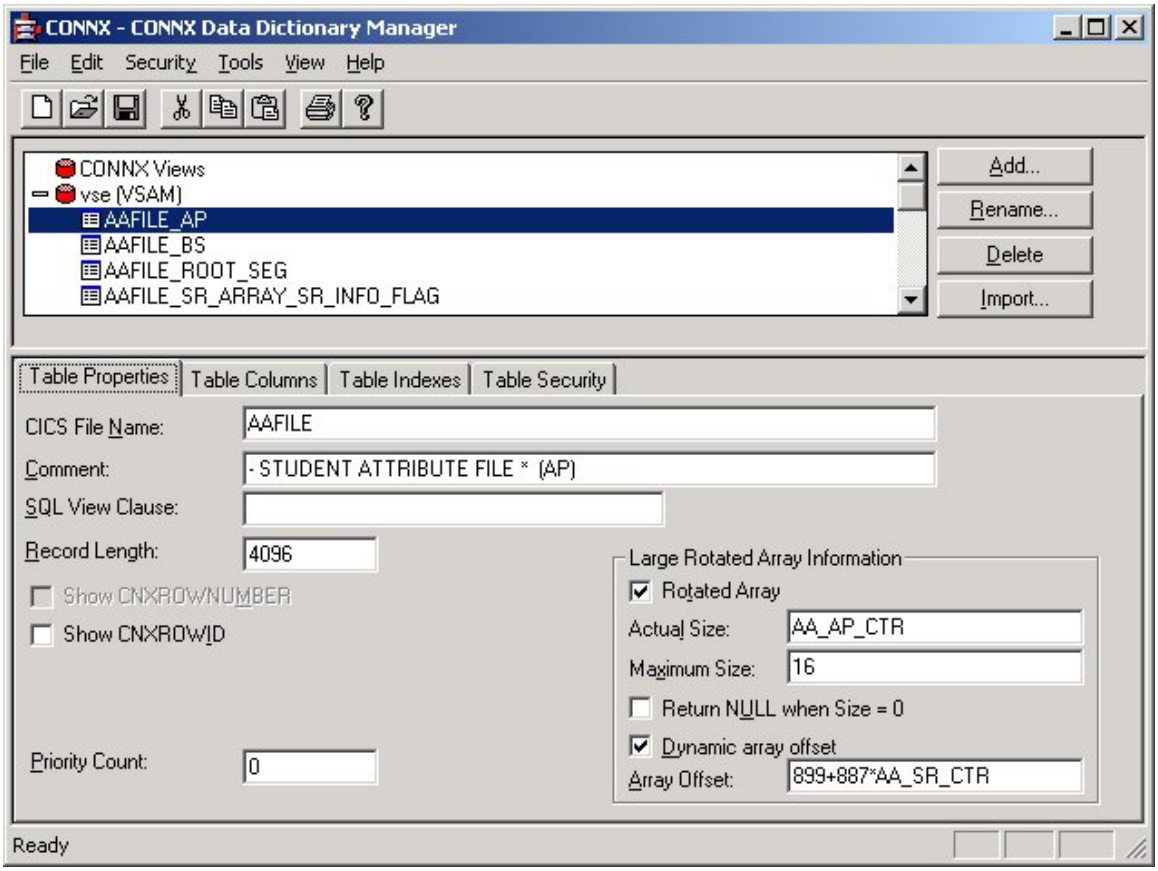
To import from SCT COBOL FD (File Definition) files (CICS) - VSE

This procedure is site-specific for SCT customers. See To import from VSAM COBOL FD (File Definition) files (CICS) - VSE for information on how to authorize and verify FTP access.

1. Click the **Import** button in the CONNX CDD Windows Application window.
2. The Import CDD dialog box appears. Select **VSAM SCT COBOL FD** in the **Import Type** list box.



3. Type a fully qualified **library.sublib.member.type SCT COBOL FD name** in the **Enter an SCT COBOL FD data set member** text box. For example, given an SCT installation library name of SCTLIB and standard sublibraries ADS, FRS, HRS, LMS, SIS, and ZSS, the AAFILE copybook resides in the fully qualified library.sublib.member.type name: SCTLIB.SIS.ACAARC.C.
From the list of imported files, select each to display the properties, columns, and indexes.



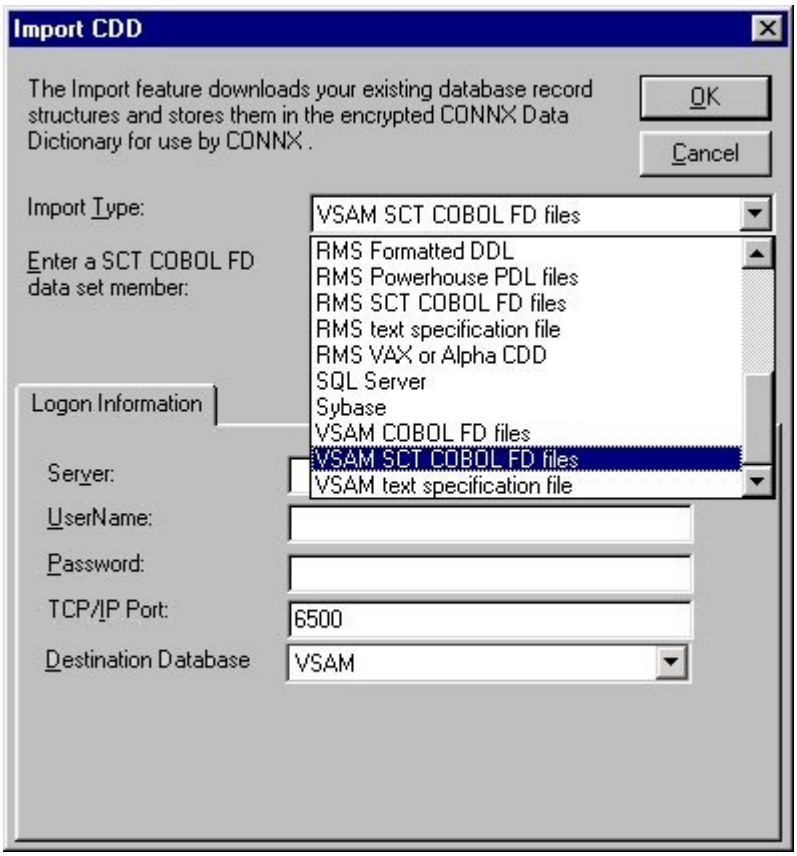
4. Save the CDD by selecting the **File** menu and clicking **Save.**

To perform a VSAM SCT DBD (Database Definition) Overlay Import - VSE

You can overlay the imported COBOL FD field names in your CONNX CDD with the corresponding FOCUS™ DBD names. You can also use the CONNX SCT DBD Overlay Import feature to add comments that correlate to the help screens in your application. Additionally, the comments contain the SCT mnemonic for each field, making it easy to locate a desired field by using the CONNX Find feature.

See To import from VSAM COBOL FD (File Definition) files (CICS) - VSE for information on how to authorize and verify FTP access.

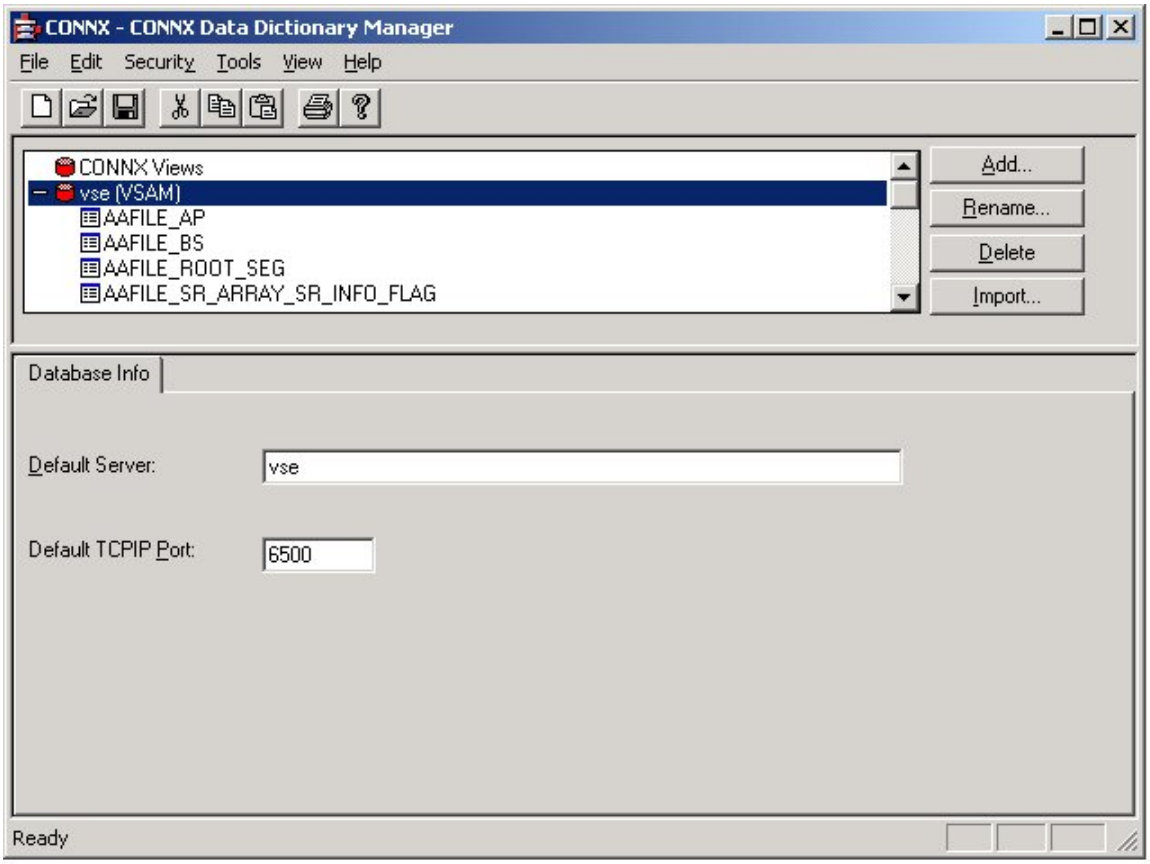
1. Click the **Import** button in the CONNX CDD Windows Application window.
2. The **Import CDD** dialog box appears. Select **VSAM SCT COBOL FD files** in the **Import Type** list box.



3. Type a fully qualified **library.sublib.member.type SCT COBOL FD name** in the **Enter an SCT COBOL FD file name** text box.

   In this example, SCTLIB.SIS.ACAARC.C is used. The SIS files from the specified COBOL FD file are imported.

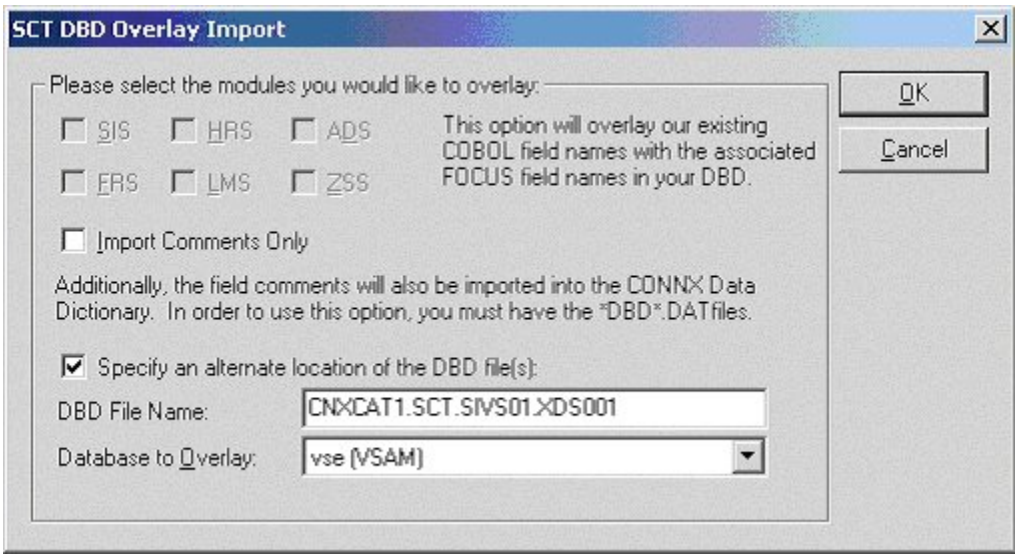
4. Click the **OK** button to import the SCT layouts.



5. The SCT file layouts are imported. The process may take a few minutes depending on the number of files imported.
6. On the **Tools** menu in the CONNX Data Dictionary Manager window, select **SCT DBD Overlay Import.**
7. The SCT DBD Overlay Import dialog box appears. Select the **Import Comments Only** check box to add only the DBD comments currently in your CDD. Importing comments only adds the mnemonic file and file identifier and any comments existing for each field in the DBD file.
8. The SCT Focus DBD overlay files are defined as sequential files in VSAM space. Given an example VSAM user catalog name of CNXCAT1 and the default SCT installation prefixes, the fully qualified SCT Focus DBD overlay files are named as follows:

CNXCAT1.SCT.ADVS02.XDS004	Alumni Development System
CNXCAT1.SCT.FRVS03.XDS002	Financial Records System
CNXCAT1.SCT.HRVS04.XDS003	Human Resource System
CNXCAT1.SCT.NLVS02.XDS005	Loan Management System
CNXCAT1.SCT.SIVS01.XDS001	Student Information System
CNXCAT1.SCT.ZSVS01.XDS000	Z Support Software System

9. Select the **Specify an alternate location of the DBD file** check box and then type the location in the text box below.



10. In this example, the CNXCAT1 user catalog must be defined to the TCP/IP for VSE job as a VSAM catalog. The syntax of the command is:

```
DEFINE FILE,PUBLIC='CNXCAT1',DLBL=CNXCAT1,TYPE=VSAMCAT,READONLY=YES
```

11. The entry for CNXCAT1 can be permanently added to the VSE TCP/IP configuration by updating the initialization JCL member (IPINIT00) with the control record syntax above, submitting the batch job to the VSE POWER batch queue, and stopping and restarting TCP/IP. The CNXCAT1 entry can be temporarily defined to the currently executing version of TCP/IP via VSE console commands, for example:

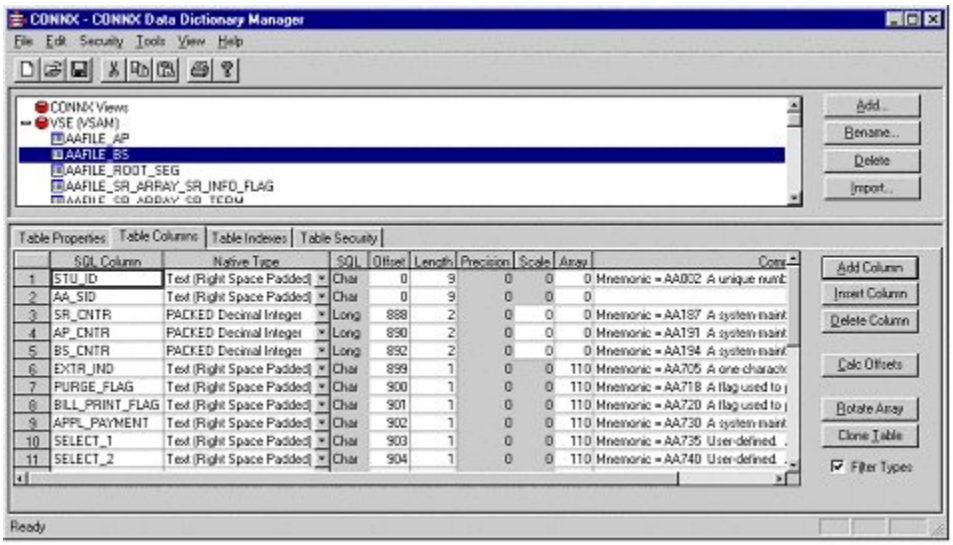
```
msg f7
AR 0015 1I40I READY
F7-0087 IPN300I Enter TCP/IP Command
87 DEFINE FILE,PUBLIC='CNXCAT1',DLBL=CNXCAT1,TYPE=VSAMCAT,READONLY=YES
```

12. To verify the entries currently defined for FTP access to the VSE TCP/IP job, start an FTP session using the Windows command prompt, and enter the dir command:

13. Select the VSAM database to overlay in the **Database to Overlay** list box.

14. Click the **OK** button. Note that SIS modules are used for this example. The conversion to FOCUS field names may take a few minutes, depending on the number of files imported.

15. Once the files are converted, select a table from the list in the upper pane. The COBOL field names in the selected modules are converted to FOCUS field names. Note that comments containing a mnemonic code for each file and field are inserted, and that the field names have changed.


CONNX - CONNX Data Dictionary Manager
File Edit Security Tools View Help
CONNX Views
VSE (VSAM)
AAFILE_AP
AAFILE_BS
AAFILE_ROOT_SEG
AAFILE_SR_ARRAY_SR_INFO_FLAG
Add...
Rename...
Delete
Import...
Table Properties
Table Columns
Table Indexes
Table Security
SQL Column
Native Type
SQL
Offset
Length
Precision
Scale
Array
Comm
1 STU_ID Text (Right Space Padded) Char 0 9 0 0 0 Mnemonic = AA002 A unique numb
2 AA_SID Text (Right Space Padded) Char 0 9 0 0 0
3 SR_CNTR PACKED Decimal Integer Long 888 2 0 0 0 Mnemonic = AA187 A system maint
4 AP_CNTR PACKED Decimal Integer Long 890 2 0 0 0 Mnemonic = AA191 A system maint
5 BS_CNTR PACKED Decimal Integer Long 892 2 0 0 0 Mnemonic = AA194 A system maint
6 EXTR_IND Text (Right Space Padded) Char 899 1 0 0 110 Mnemonic = AA705 A one characte
7 PURGE_FLAG Text (Right Space Padded) Char 900 1 0 0 110 Mnemonic = AA718 A flag used to
8 BILL_PRINT_FLAG Text (Right Space Padded) Char 901 1 0 0 110 Mnemonic = AA720 A flag used to
9 APPL_PAYMENT Text (Right Space Padded) Char 902 1 0 0 110 Mnemonic = AA730 A system maint
10 SELECT_1 Text (Right Space Padded) Char 903 1 0 0 110 Mnemonic = AA735 User-defined
11 SELECT_2 Text (Right Space Padded) Char 904 1 0 0 110 Mnemonic = AA740 User-defined
Add Column
Insert Column
Delete Column
Calc Offsets
Rotate Array
Clone Table
Filter Types
Ready

Creating CONNX VSAM Database CDD Entries Manually

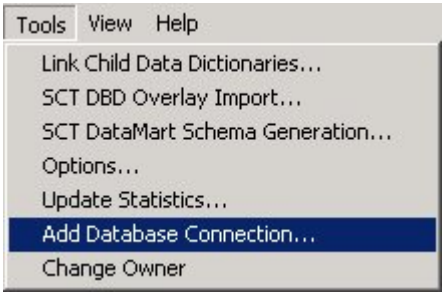
In CONNX, a VSAM database represents a collection of VSAM data sets on a given server. With the CONNX Add Database Connection feature, you can manually create new database connections to VSAM servers on a given port. After the database has been created, you can use the CONNX Import button in the CONNX Data Dictionary Manager window to add tables to the database, or you can manually create new table entries.

Related Topic


To add a CONNX VSAM database CDD entry manually

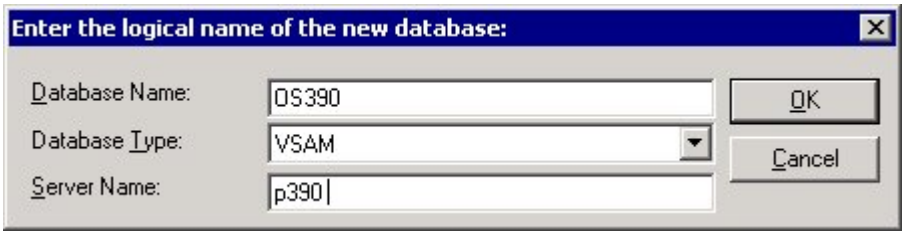
To add a CONNX VSAM database CDD entry manually

1. Select **Add Database Connection** on the **Tools** menu in the CONNX Data Dictionary Manager window.

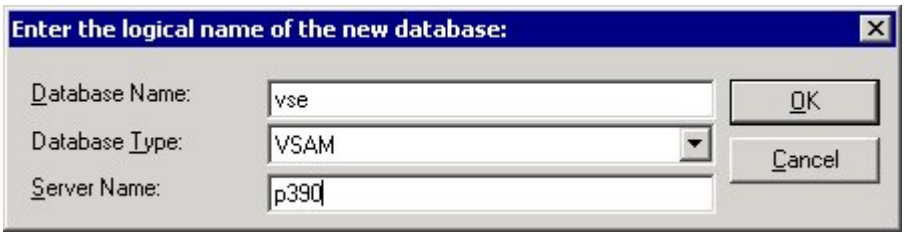


2. The Enter the logical name of the new database dialog box appears.
3. Enter a unique name in the **Database Name** text box.
4. Select **VSAM** from the Database Type list box.
5. Select **VSAM** as the type of data file to create in the Database Type text box.
6. Type the symbolic or numeric TCP/IP address in the Server Name text box. Click the **OK** button.

MVS-OS/390

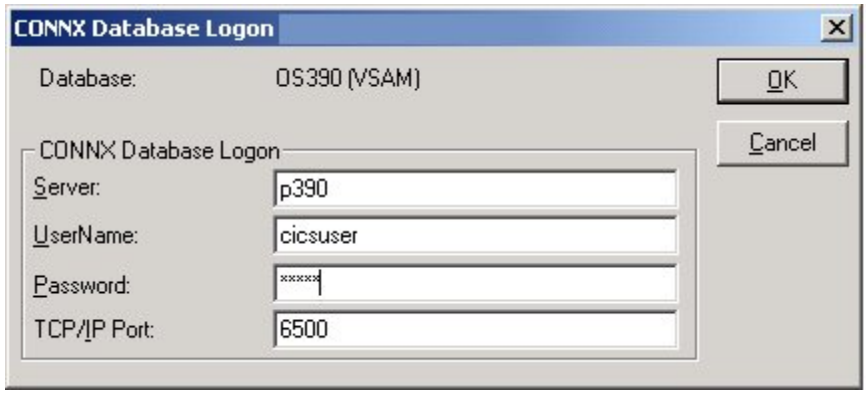


VSE

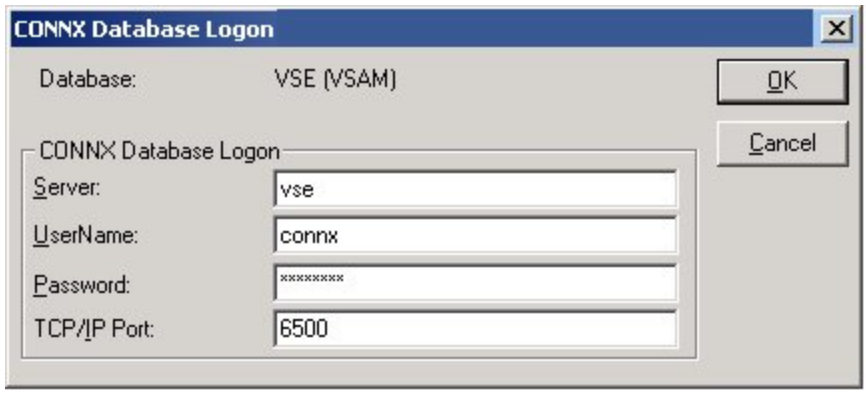


7. The CONNX Database Logon dialog box appears with the default TCP/IP port number. You can change the TCP/IP address and the port number by entering the information in the **Server Name** and **TCPIP Port** text boxes.
8. Type a **CICS user name** in the **User Name** text box.
9. Type a **CICS password** in the **Password** text box. Click the **OK** button.

MVS-OS/390


CONNX Database Logon
Database:
OS390 (VSAM)
OK
Cancel
CONNX Database Logon
Server:
p390
UserName:
cicsuser
Password:
*****
TCP/IP Port:
6500

VSE


CONNX Database Logon
Database:
VSE (VSAM)
OK
Cancel
CONNX Database Logon
Server:
vse
UserName:
connx
Password:
****
TCP/IP Port:
6500

VSAM File Import Configuration Parameters

Configuration Parameters - IMPORTALTINDEXES and FASTPATHMATCH

CONNX for VSAM provides two configuration parameters to expedite or disable the VSAM alternate index file import search process. The default settings for these parameters are optimal for the CONNX VSAM sample files and for the SCT VSAM master files. If your installation does not implement VSAM alternate index files, you can change these parameters to improve the CONNX VSAM file import response time.

The IMPORTALTINDEXES configuration parameter directs CONNX to execute or bypass the alternate index search logic for each imported VSAM file. When IMPORTALTINDEXES is enabled, CONNX searches the CICS RDO file for alternate index entries which point to each imported CICS file name. The default setting for IMPORTALTINDEXES increases CONNX VSAM file import response time in proportion to the number of RDO file entries.

Related Topics

- To disable the IMPORTALTINDEXES configuration option
- To enable the IMPORTALTINDEXES configuration option
- To configure the FASTPATHMATCH configuration option

To disable the IMPORTALTINDEXES option

1. Log on to the CICS region on which the CONNX for VSAM components have been installed, clear the screen, and type:

   ```
   NX01 IMPORTALTINDEXES 0
   ```

2. The expected response is:

   ```
   Entry Added (IMPORTALTINDEXES)=(0)
   ```

3. To confirm that the configuration option has been disabled, clear the screen and type:

   ```
   NX01
   ```

4. The current CONNX for VSAM configuration is displayed, for example:

   ```
   Displaying CONNX Configuration
   (IMPORTALTINDEXES)=(0)
   ```

To enable the IMPORTALTINDEXES configuration option

1. Clear the screen and enter either

```
NX01 IMPORTALTINDEXES 1
```

Or

```
NX01 IMPORTALTINDEXES
```

2. The first command explicitly sets the option to 1, while the second removes the configuration option and thereby restores the default setting of 1. The expected response for the first command is:

```
Entry Added (IMPORTALTINDEXES)=(1)
```

For the second:

```
Removing Configuration Entry (IMPORTALTINDEXES)
Configuration Entry Successfully Removed. (IMPORTALTINDEXES)
```

To configure the FASTPATHMATCH configuration option

CONNX for VSAM provides two configuration parameters to expedite or disable the VSAM alternate index file import search process. The default settings for these parameters are optimal for the CONNX VSAM sample files and for the SCT VSAM master files. If your installation does not implement VSAM alternate index files, you can change these parameters to improve the CONNX VSAM file import response time.

When enabled, the FASTPATHMATCH configuration parameter restricts the CONNX alternate index file name search logic to a two-character prefix based on the imported CICS file name. For example, all CONNX VSAM sample files are defined to the CICS RDO file with a three-character CNX prefix. The CONNX sample VSAM KSDS equipment file is named CNXEQK. The CICS file names for the two alternate indexes for this file are CNXEQI1 and CNXEQI2. The default FASTPATHMATCH setting directs the CONNX VSAM alternate index search logic to start with CICS file names beginning with CN and to stop as soon as a different two-character CICS file name prefix is found.

1. To disable the FASTPATHMATCH option and force a complete search of the RDO file, clear the screen and type:

   ```
   NX01 FASTPATHMATCH 0
   ```

   The expected reponse is:

   ```
   Entry Added (FASTPATHMATCH)=(0)
   ```

2. To enable the FASTPATHMATCH option, clear the screen and type:

   ```
   NX01 FASTPATHMATCH 1
   ```

   The expected response is:

   ```
   Removing Configuration Entry (FASTPATHMATCH)
   Configuration Entry Successfully Removed. (FASTPATHMATCH)
   Entry Added (FASTPATHMATCH)=(1)
   ```

   Or

   ```
   NX01 FASTPATHMATCH
   ```

   The expected response is:

   ```
   Removing Configuration Entry (FASTPATHMATCH)
   Configuration Entry Successfully Removed. (FASTPATHMATCH)
   ```

CONNX for VSAM Configuration Options

IMPORTALTINDEXES	FASTPATHMATCH	REMARKS
Disabled	Disabled	Disables alternate index import logic: fastest setting.
Disabled	Enabled	Same as above.
Enabled	Disabled	Enables alternate index import complete search logic: slowest but most accurate

		setting.
Enabled	Enabled	Default setting: restricts alternate index import logic to CICS short file names with the same two-character prefix as the base CICS short file name. Fast but potentially incomplete. Optimal for CONNX sample and SCT master VSAM files.

VSAM Text File Import Specification

The VSAM text file import specification should be used only if your record layouts **are not** COBOL FD, Formatted DDL, or Powerhouse PDL format.

If you have many record layouts to import from non-standard formats, it is possible to convert them into the CONNX text file import format. This will require that you write an application to convert your existing record layouts to the CONNX text file format specification. If you have only a few small record layouts, it may be faster to manually enter them into a new or existing CONNX Data Dictionary instead of using the text file import.

The text file import specification is described below.

The first line of each record layout should be as follows:

CONNXTABLE, <TableName>, <VSAM File Name>,<Record Length><SQL View Clause>

Note: Inclusion of a SQL View Clause is optional.

One import text file may contain multiple record layouts, each starting with the same header line shown above.

Each subsequent line in the file represents a column in the record layout. The format for each line is as follows:

<column name>, <column length>, <column offset>, <column type>, <column scale>, <column base>, <column fraction>, <column comment>

VSAM Text File Import Syntax and Description

Syntax	Description
<column name>	Name of the column.
<column length>	Length of the column.
<column offset>	Offset of the column.
<column type>	Name for the data type of the column.
<column scale>	Scale of the column (power of 10). A scale of 2 would convert 4.3 to 430.
<column base>	Reserved - must be 0.
<column fraction>	Fraction of the column (negative power of 10). A fraction of 2 would convert 4.3 to .043. A fraction of -2 would convert to 4.3 to 430.
<column comment>	Comments field.

Important: When creating a VSAM text import specification file, it is recommended that you use the column scale syntax rather than the column fraction syntax.

The following is an example of a VSAM import file. It includes an optional SQL View Clause. Inserting a view clause limits the result set. This sample view clause returns only rows where the Company field is not blank. The SQL View Clause text box is located on the Properties tab in the CONNX Data Dictionary Manager window:

```
CONNXTABLE, CompanyTable, [CICS file name], 64, Company <>"
Company, 30, 0, Text (Right Space Padded), 0, 0, 0, This is the Company Field.
```

Title, 10, 30, Text (Right Space Padded), 0, 0, 0, This is the Title Field.

Name, 20, 40, Text (Right Space Padded), 0, 0, 0, This is the Name Field.
Age, 4, 60, Text (Right Space Padded), 0 ,0 ,2, This is the Age Field.

For more information about CONNX for VSAM data types, see IBM Mainframe Data Types.

Related Topic

SQL View Clause Text Box
VSAM View Text File Import Specification

VSAM View Text File Import Specification

The VSAM VIEW text file import specification can be used to populate your CDD with predefined CONNX Views using the VSAM Text Import Option.

The VIEW text file import specification layout is described below.

The **first** line of each view layout must contain CONNXVIEW and the view object name as follows:

```
CONNXVIEW <VIEWOBJECTNAME>
```

The subsequent lines of each view layout must contain the SQL Select statement:

```
SELECT ...
```

The **last** or **footer** line in each view must contain:

```
ENDVIEW
```

One import text file may contain multiple views, each starting with the same header line shown above and with a footer line with the word ENDVIEW.

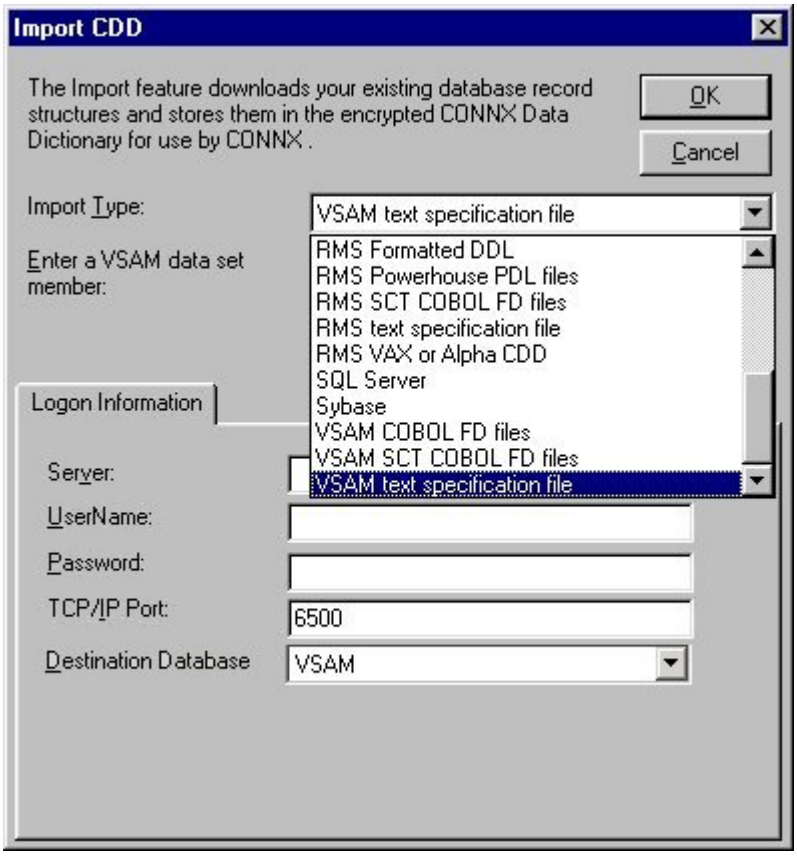
The following is an example of a VSAM VIEW import file.

```
CONNXVIEW, NWORDERS
/*This view was requested by Johnathon Jones on 3/1/2001. He executes this
view daily to see orders for the Northwest Territory. */
SELECT
ORDERS_VSAM.orderid as 'Order' /* Order Number */,
ORDERS_VSAM.customerid as 'Cust Id' /* Customer Identification */,
CUSTOMERS_VSAM.customername as 'Name' /* Name of Customer*/,
CUSTOMERS_VSAM.customerstate as 'ST' /* State Ordered by */,
ORDERS_VSAM.orderdate as 'Ord Date' /* Date Ordered */,
ORDERS_VSAM.productid as 'Product' /* Product number */,
PRODUCTS_VSAM.productname as 'Description' /* Product Description */,
ORDERS_VSAM.productquantity as 'Qty' /* order quantity */,
PRODUCTS_VSAM.productprice as 'Price' /* price per unit */,
(ORDERS_VSAM.productquantity * PRODUCTS_VSAM.productprice) as 'Ext Price' /*
Calculate extended price) */
FROM ORDERS_VSAM, CUSTOMERS_VSAM, PRODUCTS_VSAM /* Tables included in view */
WHERE ORDERS_VSAM.customerid=CUSTOMERS_VSAM.customerid AND
ORDERS_VSAM.productid=PRODUCTS_VSAM.productid and
CUSTOMERS_VSAM.customerstate in ('WA', 'OR', 'MT', 'ID', 'CA') /* Join tables
together and select only Northwest states */
ENDVIEW
```

To import tables or views from a VSAM text file import specification - MVS-OS/390

The FTP process copies the text file import specification off the host.

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **VSAM text specification file** in the **Import Type** list box.



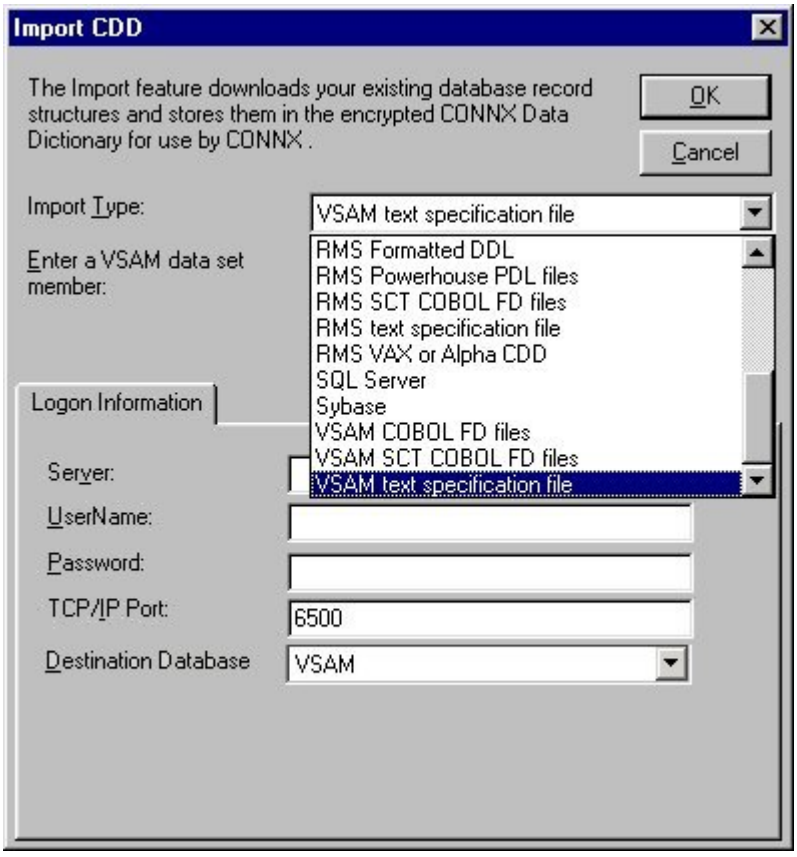
3. Type the full sequential or partitioned data set name for the text file you created and stored on the host in the **Enter a VSAM data set member** text box.
1. Type the server name or IP address, a user name, and password in the corresponding text boxes on the **Logon Information** tab.
2. Port 6500 is listed in the TCPIP Port text box by default. Any change made to the port setting in this text box becomes a permanent change to the port setting of the imported database.
3. Select a **Destination Database** for the imported tables. See Adding a Database Connection for more information.
4. From the list of available tables or views, select each imported table or view and follow these steps:
1. For tables, click the **Table Indexes** tab in the CONNX Data Dictionary Manager window.
2. For views, click the **Syntax** check button located under the **View SQL** tab.
3. Repeat steps a) and b) for each table or view for which there is imported metadata.
4. Save the CDD by selecting the **File** menu and then clicking **Save.**

Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

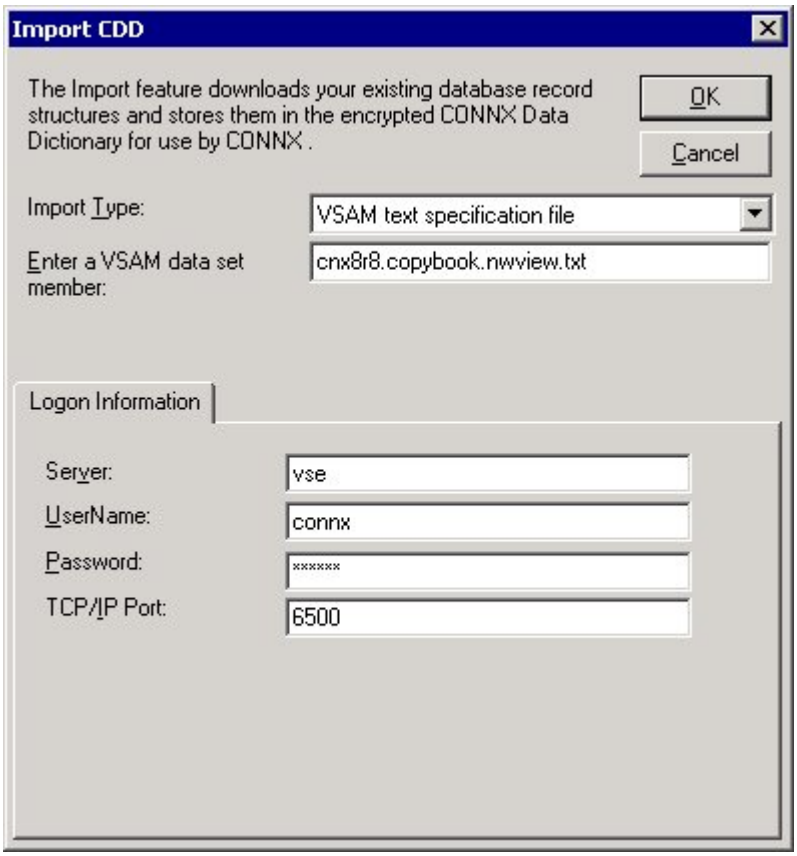
To import tables or views from a VSAM text file import specification-VSE

The FTP process copies the text file import specification off the host.

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The **Import CDD** dialog box appears. Select **VSAM text specification file** in the **Import Type** list box.



3. Type the fully qualified **library.sublib.member.type** name for the text file you created and stored on the host in the **Enter a VSAM data set member** text box.



1. Type the server name or IP address, a user name, and password in the corresponding text boxes on the **Logon Information** tab.
2. Port 6500 is listed in the TCPIP Port text box by default. Any change made to the port setting in this text box becomes a permanent change to the port setting of the imported database.
3. Select a **Destination Database** for the imported tables. See Adding a Database Connection for more information.
4. From the list of available tables or views, select each imported table or view and follow these steps:

1. For tables, click the **Table Indexes** tab in the CONNX Data Dictionary Manager window, and then click the **Refresh Indexes** button.
2. For views, click the **Syntax** check button located under the **View SQL** tab.
3. Repeat steps a) and b) for each table or view for which there is imported metadata.
4. Save the CDD by selecting the **File** menu and then clicking **Save.**

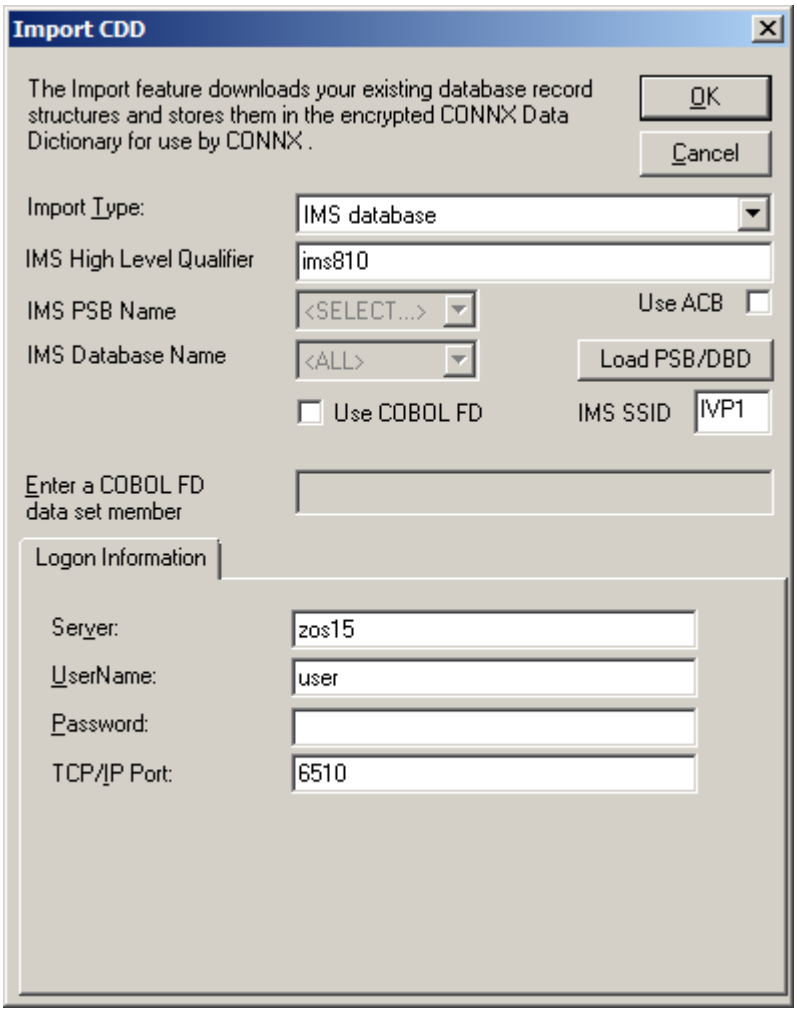
Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

CONNX and IMS

Importing IMS files

This section describes importing IMS files using the IBM® IMS™ Library Integrity Utilities for z/OS. If these utilities are not present on your system, you will need to import using an Index Text Specification file. Please see the section Importing IMS files using Index Text Specification file for more information.

1. Click the **Import** button in the CONNX Data Dictionary Manager window. The **Import CDD** dialog box appears.

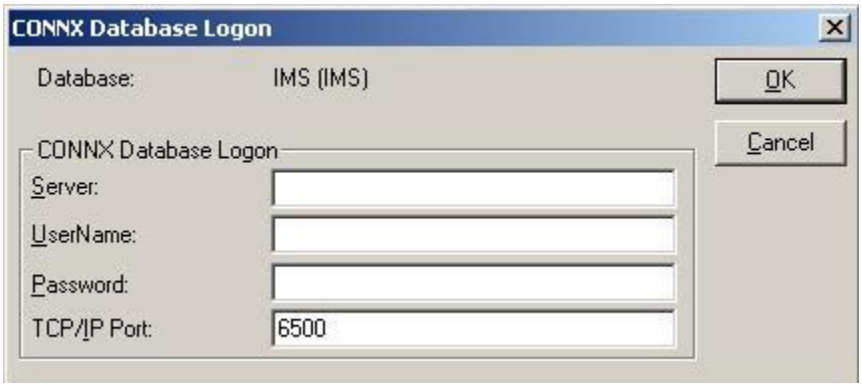


2. Select **IMS database** in the **Import Type** list box.
3. Type the High Level Qualifier (HLQ) for the IMS subsystem you want in the **IMS High Level Qualifier** text box.
4. There are two different IMS sources for gathering CDD import metadata:
   - the ACBLIB data set

     Click the **Use ACB** check box to use the ACBLIB data set for CDD import metadata.
   - the PSBLIB and DBDLIB data sets

     Clear the **Use ACB** check box to use the PSBLIB and DBDLIB data sets for CDD import metadata.

5. Click the **Load PSB/DBD** button. If this is the first time you have imported a file from this HLQ, the Logon Information will be blank. A CONNX Database Logon window appears.

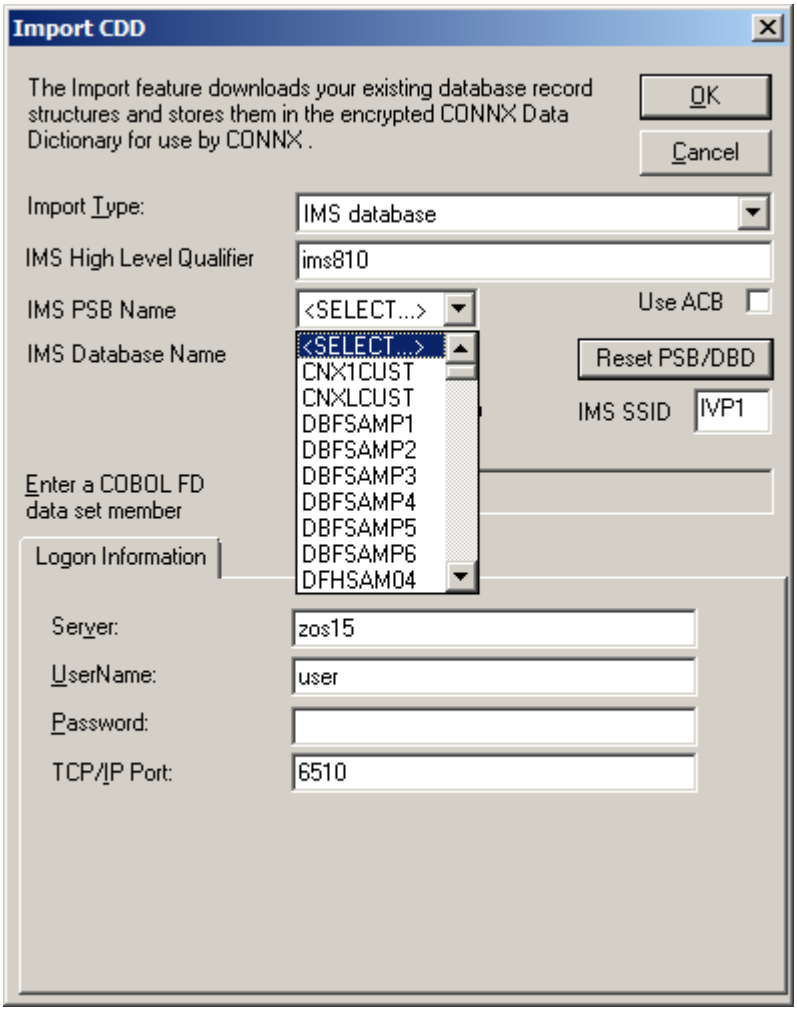


   Enter the Server name in the **Server** text box, a TSO user name and its password in the **User Name** and **Password** text boxes, and the TCP/IP port number in **TCP/IP Port**. Click **OK**. The Logon information appears under the **Logon Information** tab.

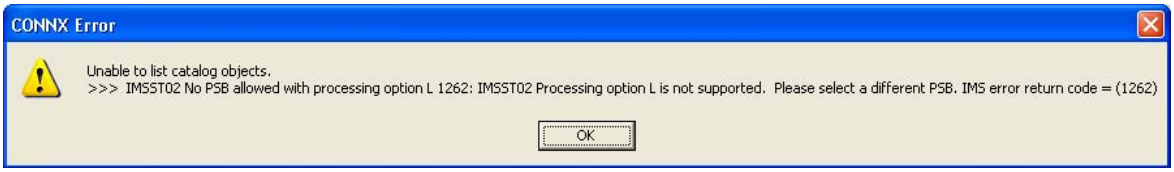
6. The **Load PSB/DBD** button changes to the **Reset PSB/DBD** button.

   At this point, you can either select the PSB first and then the IMS database, or select the IMS database first and then select the PSB.

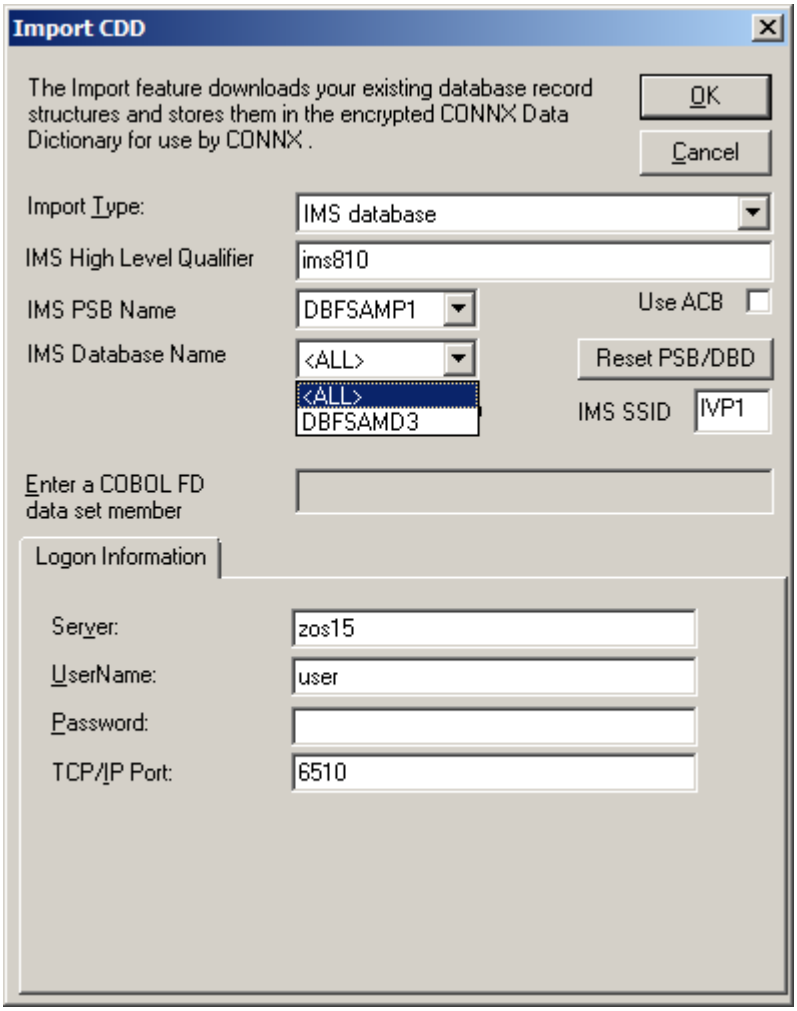
7. If you want to select the PSB first, select the PSB name from the **IMS PSB Name** drop-down entries.



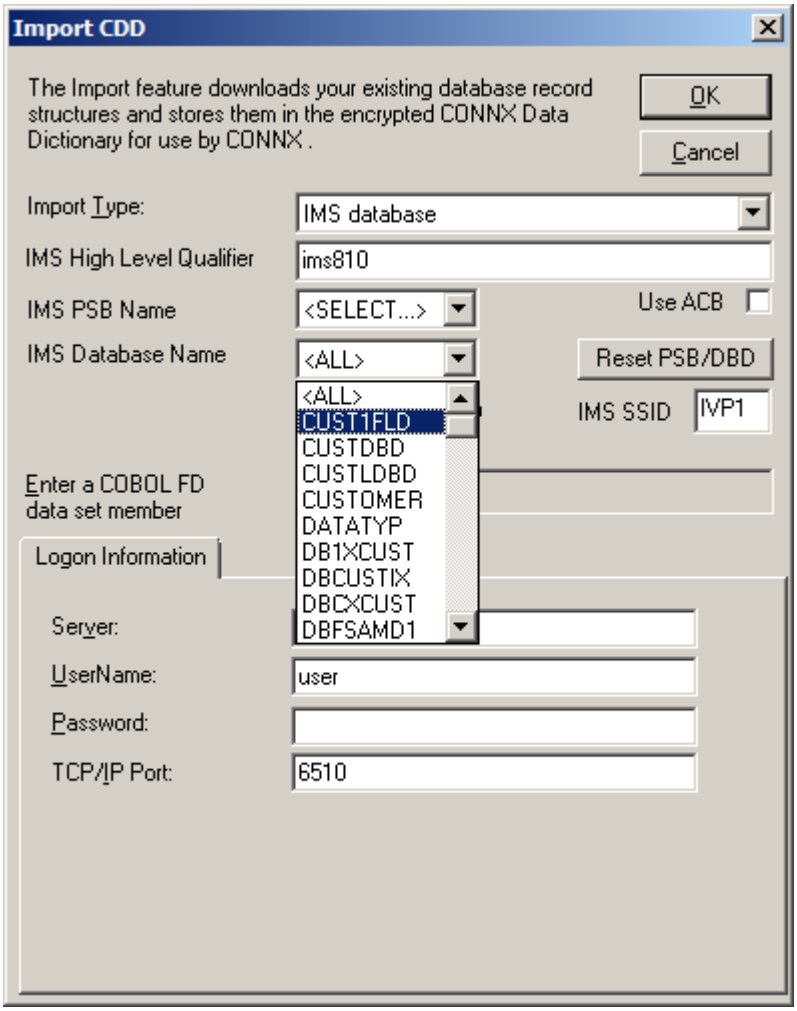
Warning: You can not import a PSB with a processing option of L (load PSBs). If you try to, you will get the following error message:



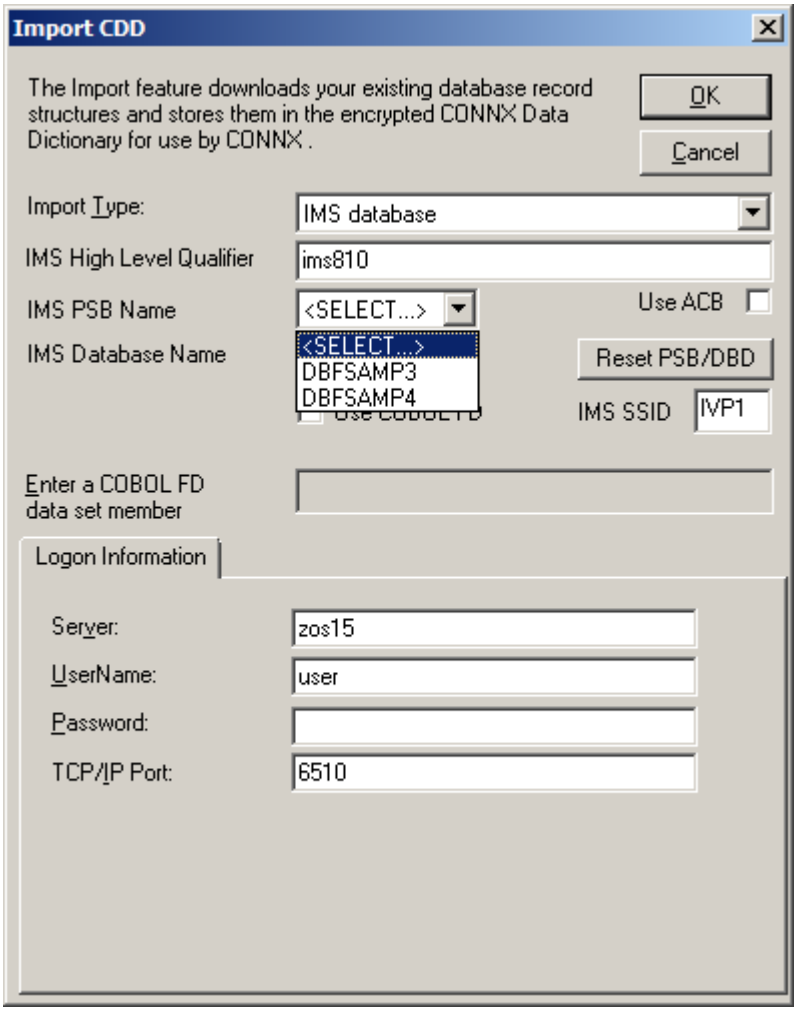
Select the IMS database(s) you want from the **IMS Database Name** drop-down box. The default value is <ALL>.



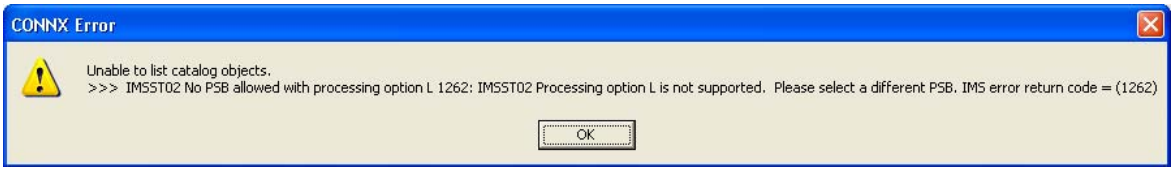
8. If you want to select the IMS database first, select the IMS database name (or <ALL>) from the **IMS Database Name** drop-down entries.



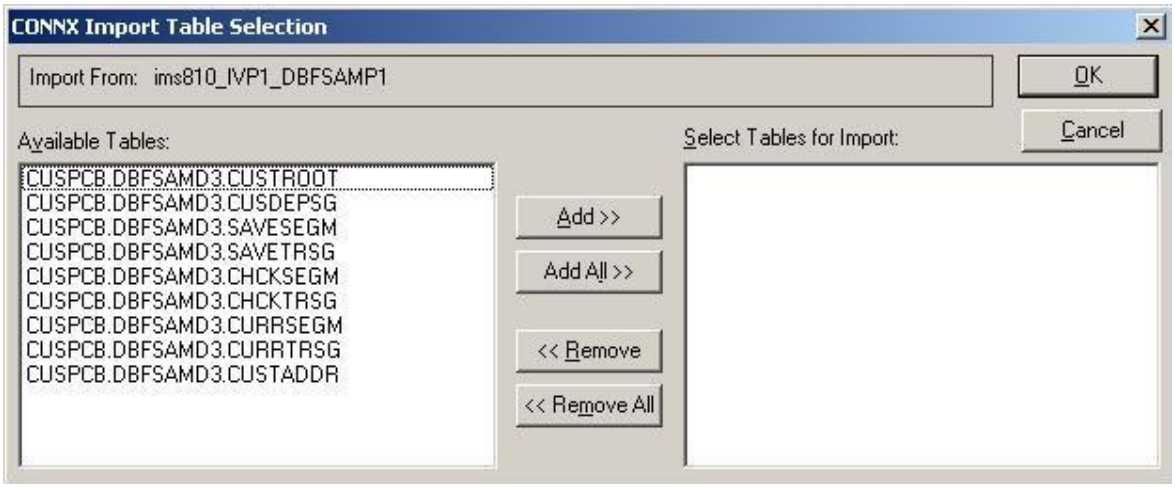
Once you have selected the IMS database you want, select the PSB name from the I**MS PSB Name** drop-down list box.



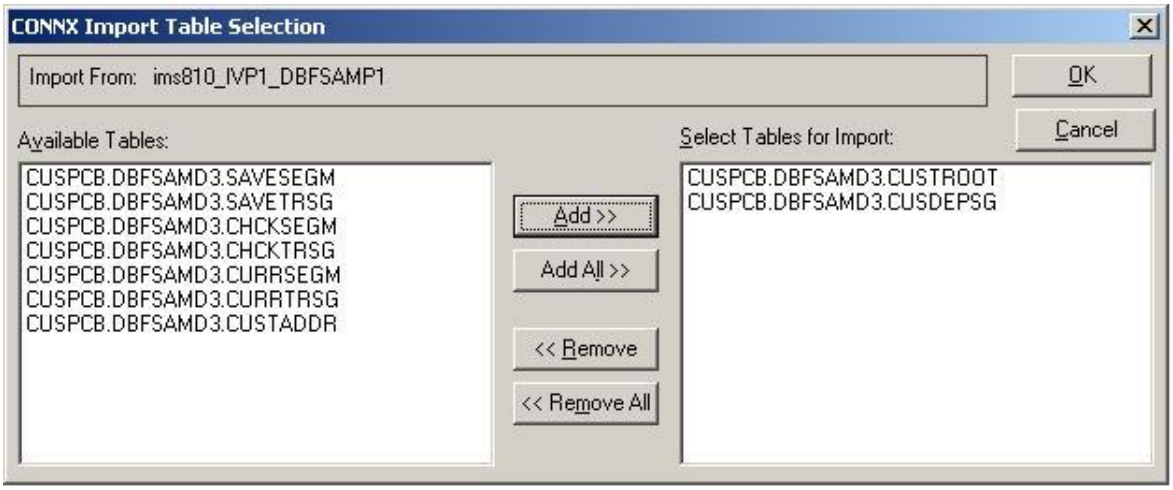
Warning: You can not import a PSB with a processing option of L (load PSBs). If you try to, you will get the following error message:



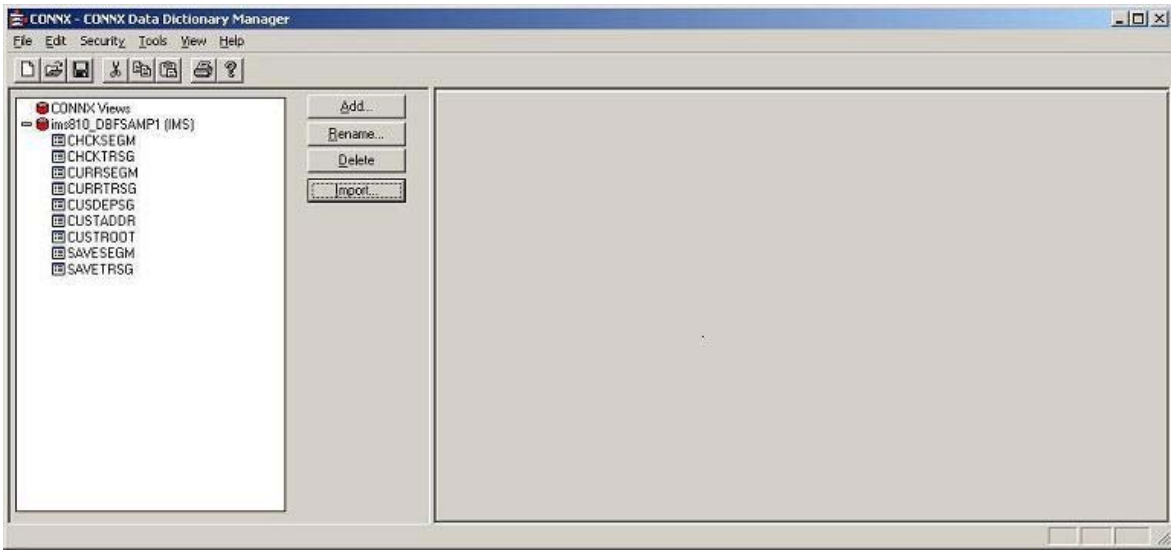
9. Enter the IMS subsystem ID in the **IMS SSID** text box and click **OK**. The **CONNX Import Table Selection** dialog box appears.



10. Select the tables (segments) you want to import from the **Available Tables** list. Click the **Add>>** button. The tables will move from the **Available Tables** list to the **Select Tables for Import** list. If you want all the tables, click the **Add All>>** button to move all of the **Available Tables** to **Select Tables for Import**.

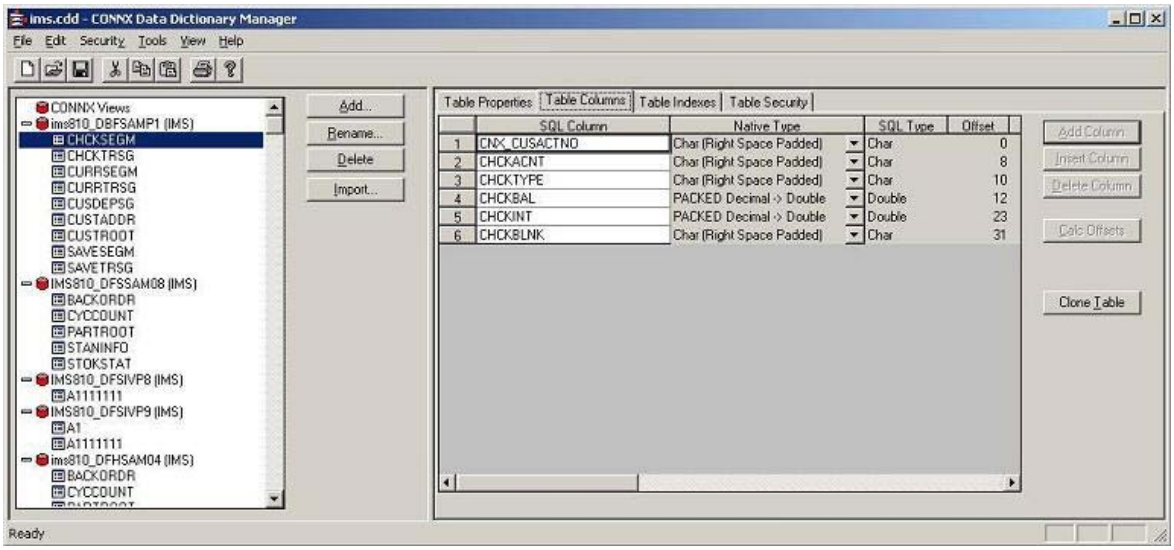


11. Click **OK**. The selected tables appear under the PSB name in the **CONNX Data Dictionary Manager** dialog box.

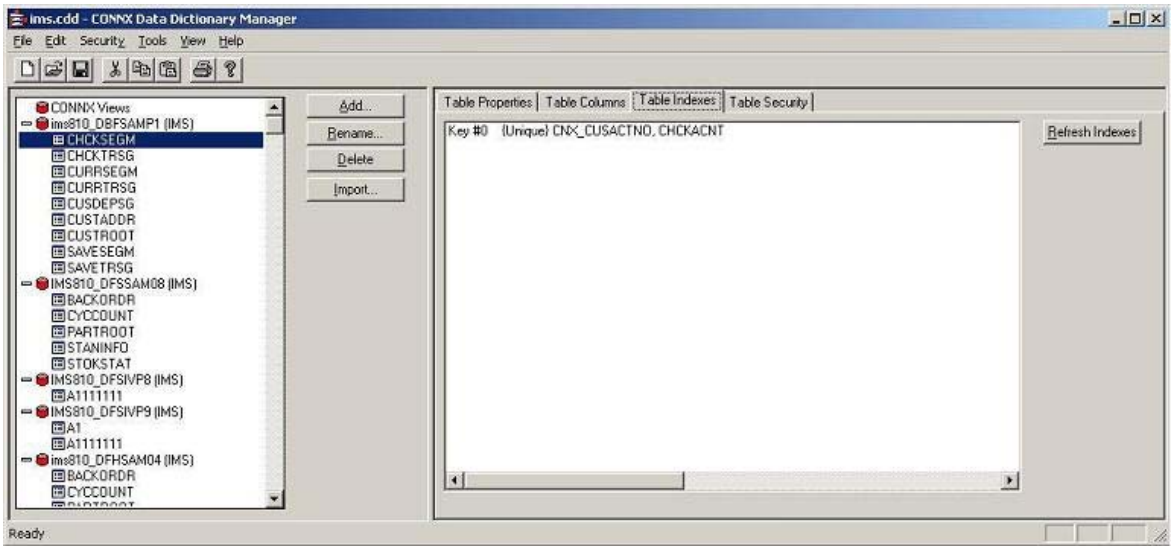


12. If you select one of the tables, the adjoining pane has tabbed table information.
    - You can edit some of the table properties under the **Table Properties** tab.

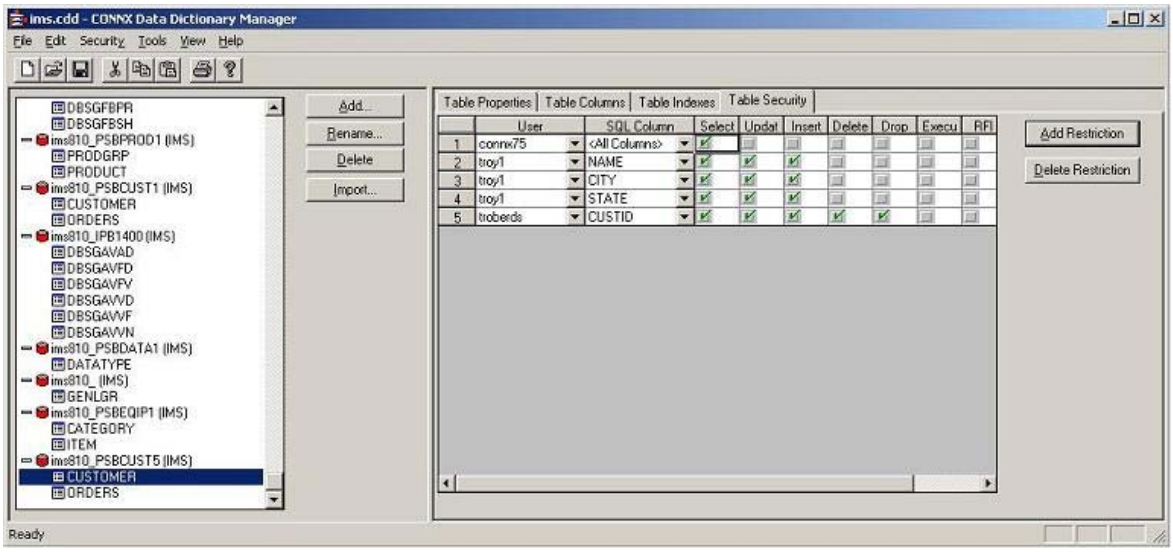
    - You can clone the table under the **Table Columns** tab.



- You can refresh the indexes under the **Table Indexes** tab.



- You can update the table security restrictions under the **Table Security** tab.



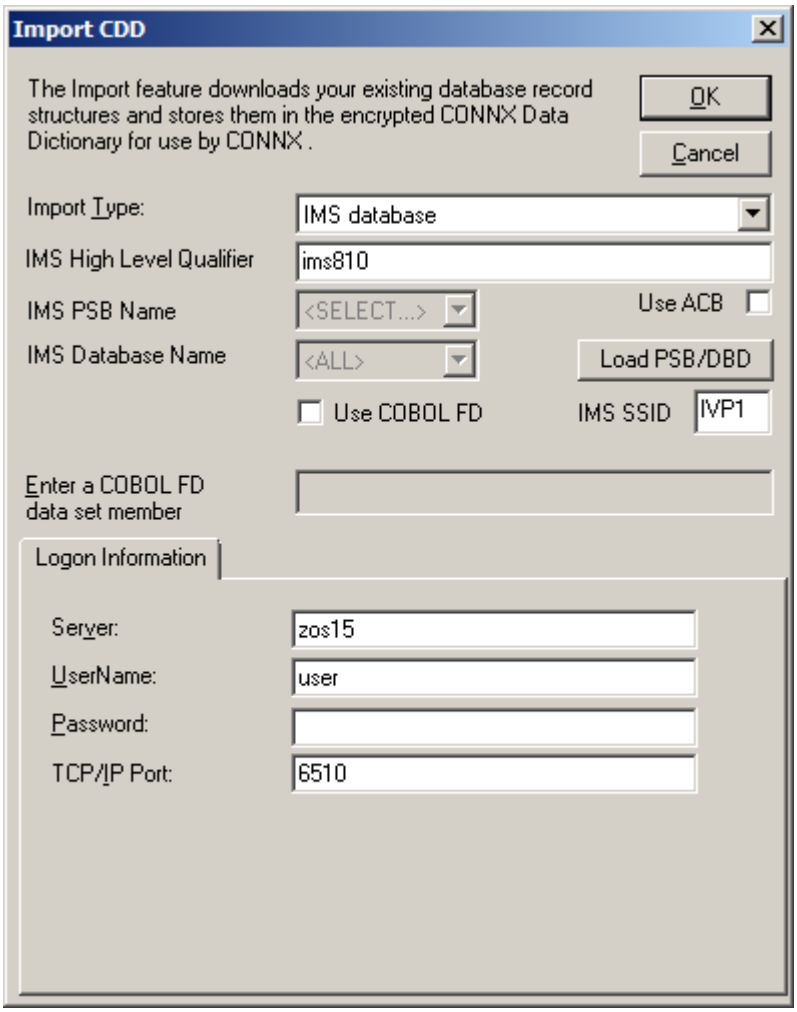
13. Close the dialog box and click **OK** to save the changes.

Importing IMS files using COBOL FD (File Definition) files

This section describes importing IMS files using the IBM® IMS™ Library Integrity Utilities for z/OS along with a COBOL COPYBOOK. If these utilities are not present on your system, you will need to import using an Index Text Specification file. Please see the section Importing IMS files using Index Text Specification file for more information.

It is possible to define a segment with only the key field information and then use the field definitions in a COBOL COPYBOOK
to define the rest of the segment. If this case, you can use the COBOL FD import option.

1. Click the **Import** button in the CONNX Data Dictionary Manager window. The **Import CDD** dialog box appears.



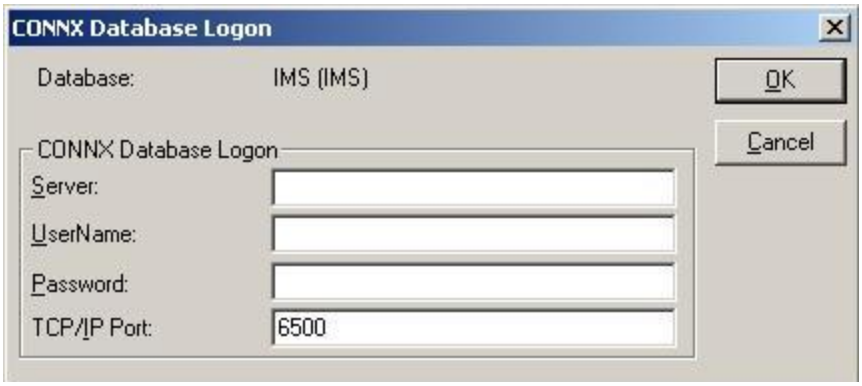
2. Select **IMS database** in the **Import Type** list box.
3. Type the High Level Qualifier (HLQ) for the IMS subsystem you want in the **IMS High Level Qualifier** text box.
4. There are two different IMS sources for gathering CDD import metadata:
   - the ACBLIB data set

   Click the **Use ACB** check box to use the ACBLIB data set for CDD import metadata.

- the PSBLIB and DBDLIB data sets

  Clear the **Use ACB** check box to use the PSBLIB and DBDLIB data sets for CDD import metadata.

5. Click the **Load PSB/DBD** button. If this is the first time you have imported a file from this HLQ, the Logon Information will be blank. A CONNX Database Logon window appears.

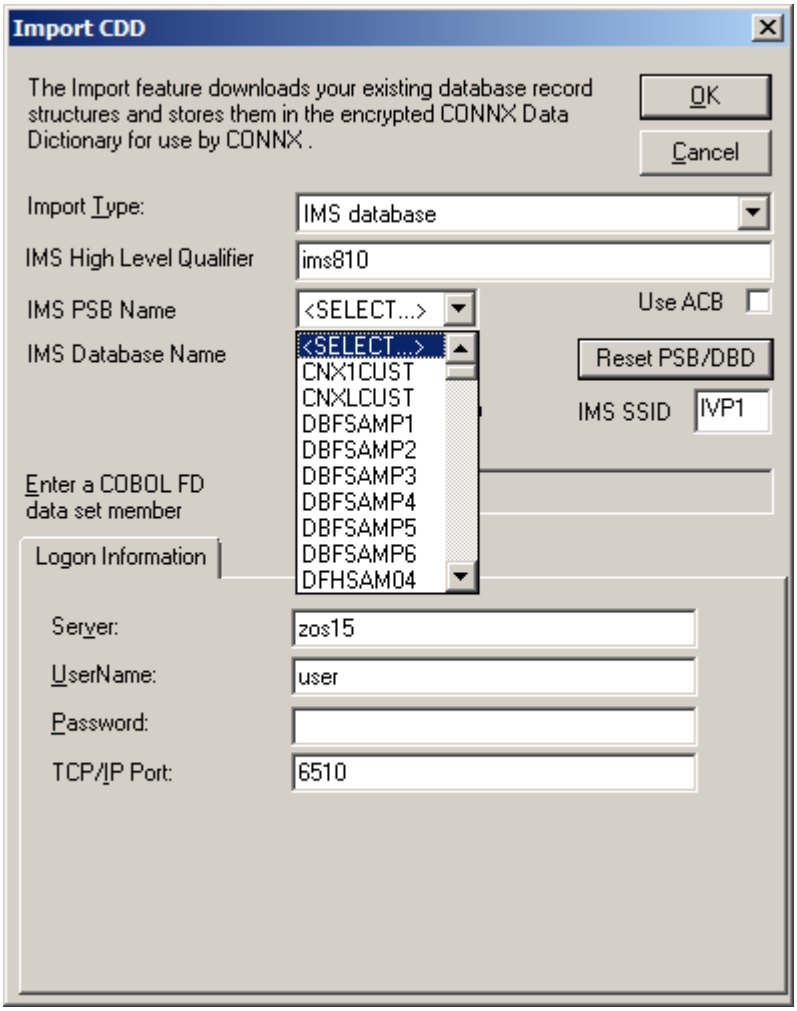


   Enter the Server name in the **Server** text box, a TSO user name and its password in the **User Name** and **Password** text boxes, and the TCP/IP port number in **TCP/IP Port**. Click **OK**. The Logon information appears under the **Logon Information** tab.

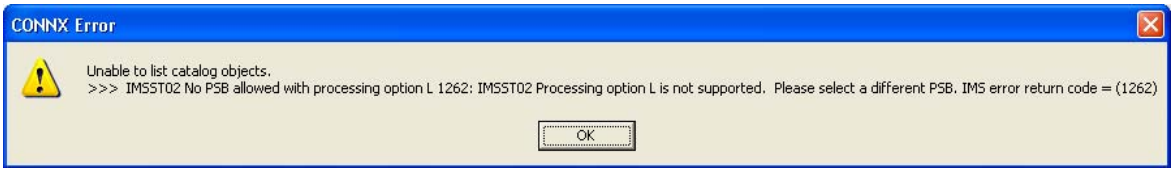
6. The **Load PSB/DBD** button changes to the **Reset PSB/DBD** button.

   At this point, you can either select the PSB first and then the IMS database, or select the IMS database first and then select the PSB.

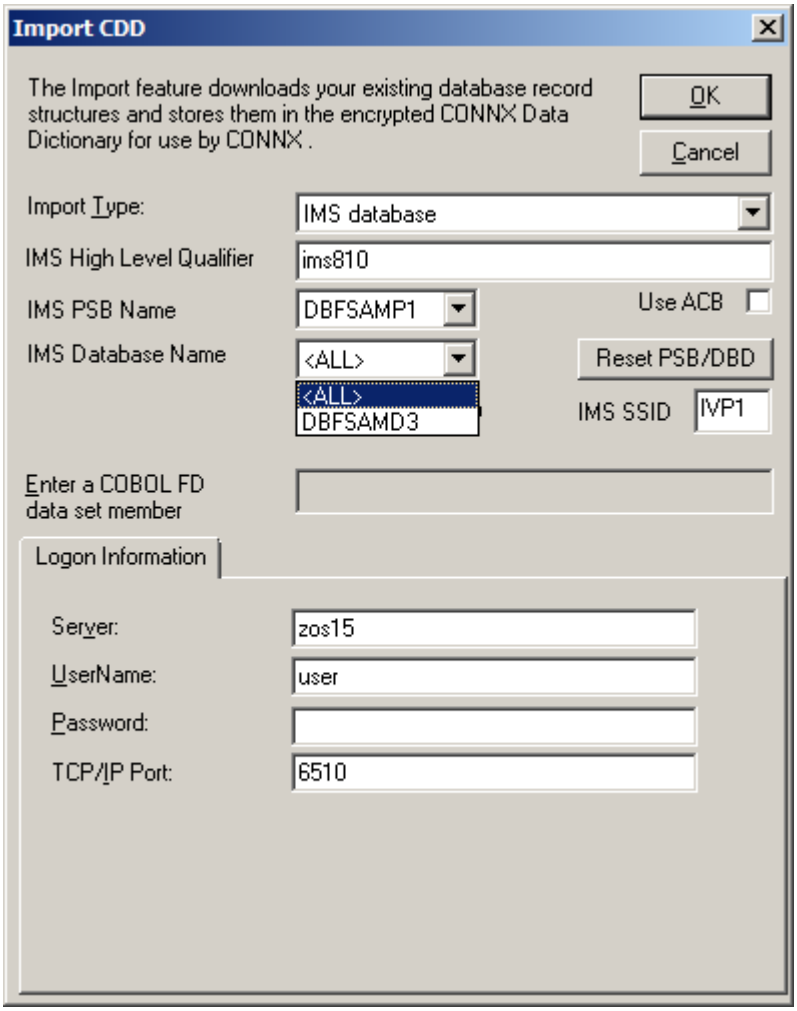
7. If you want to select the PSB first, select the PSB name from the **IMS PSB Name** drop-down entries.



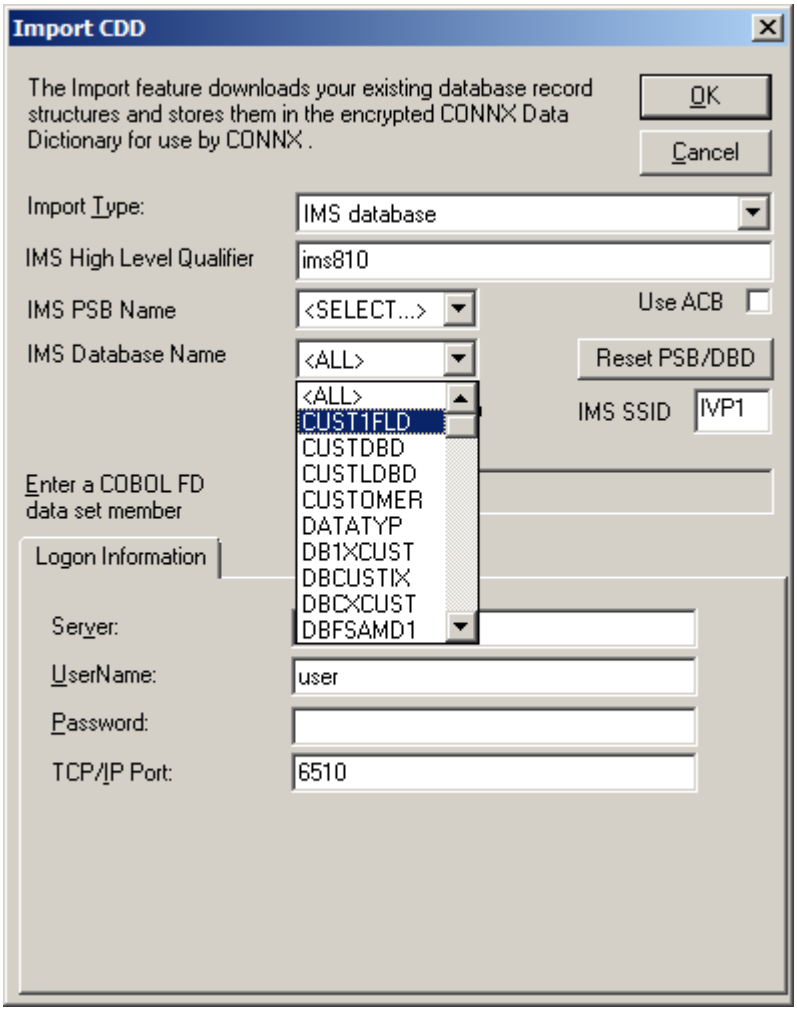
Warning: You can not import a PSB with a processing option of L (load PSBs). If you try to, you will get the following error message:



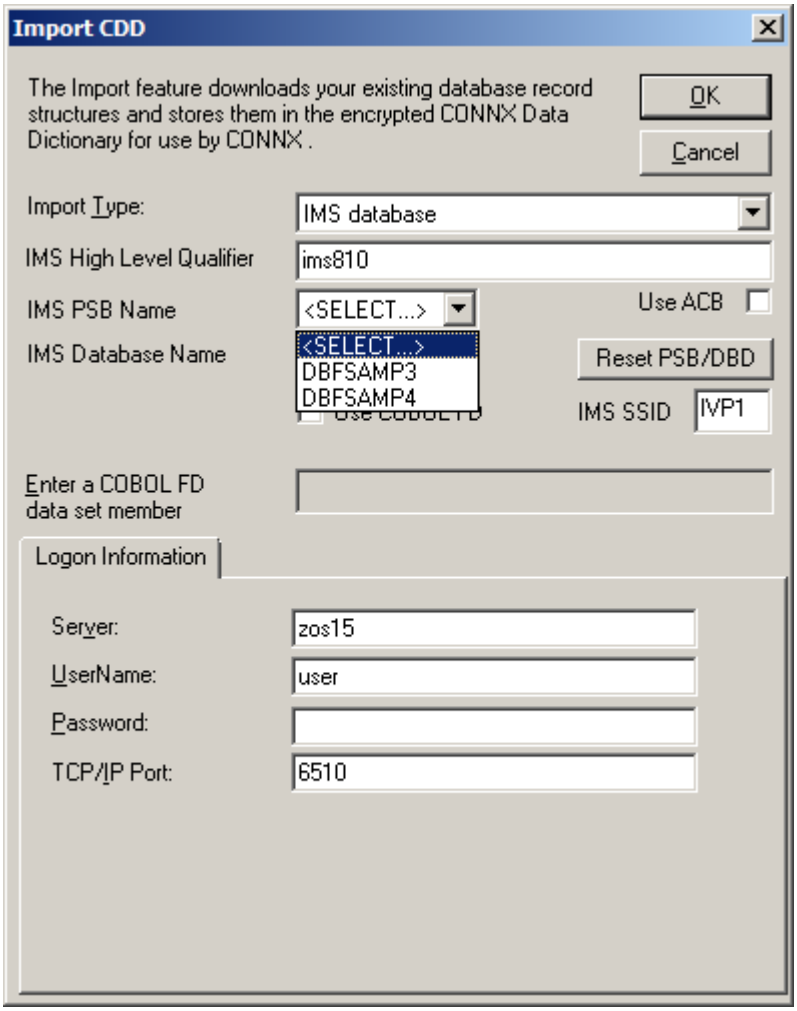
Select the IMS database(s) you want from the **IMS Database Name** drop-down box. The default value is <ALL>.



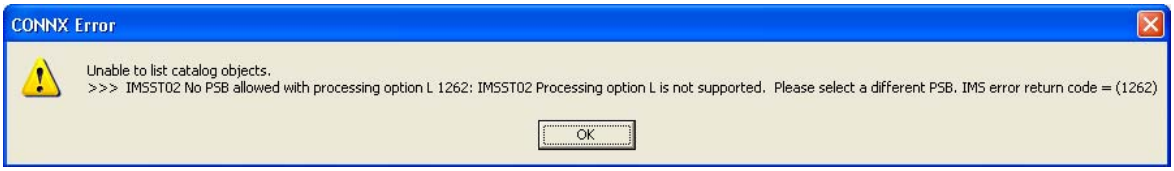
8. If you want to select the IMS database first, select the IMS database name (or <ALL>) from the **IMS Database Name** drop-down entries.



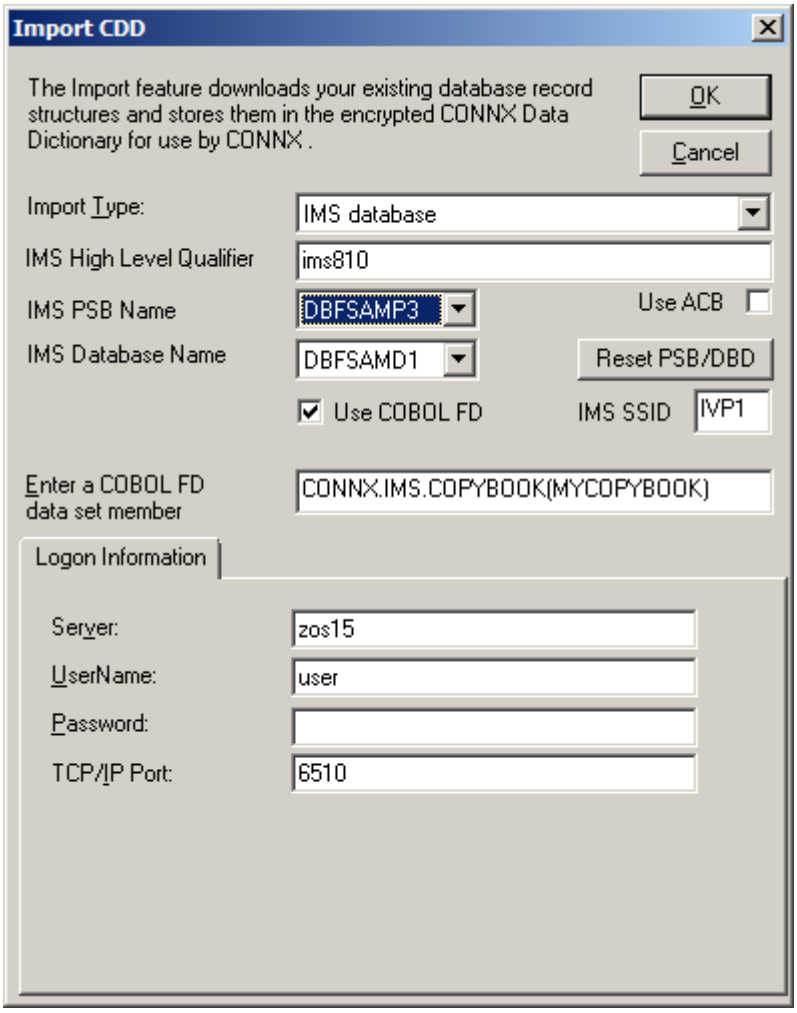
Once you have selected the IMS database you want, select the PSB name from the I**MS PSB Name** drop-down list box.



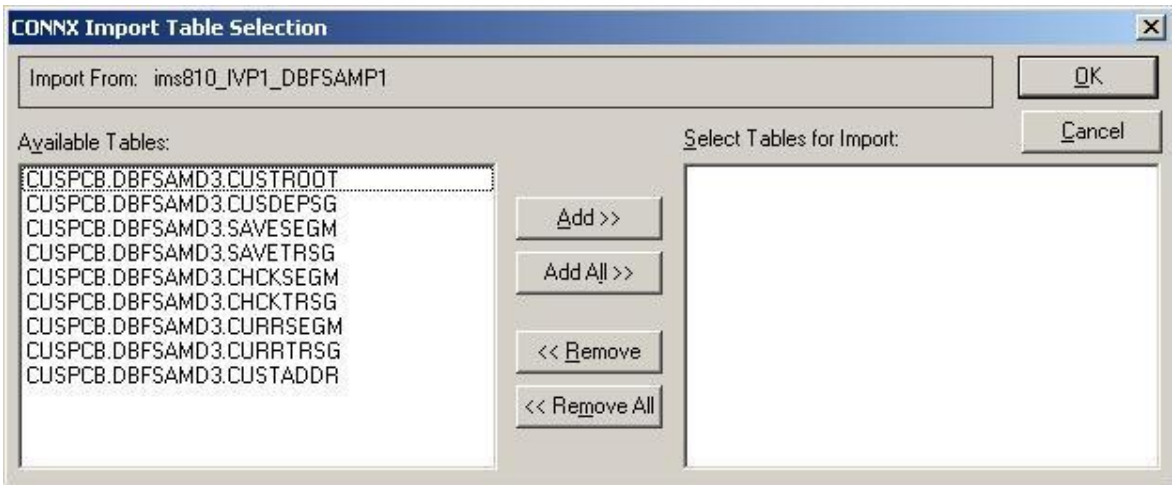
Warning: You can not import a PSB with a processing option of L (load PSBs). If you try to, you will get the following error message:



9. Select the Use COBOL FD check box and enter the fully qualified path to the IMS COBOL FD file.

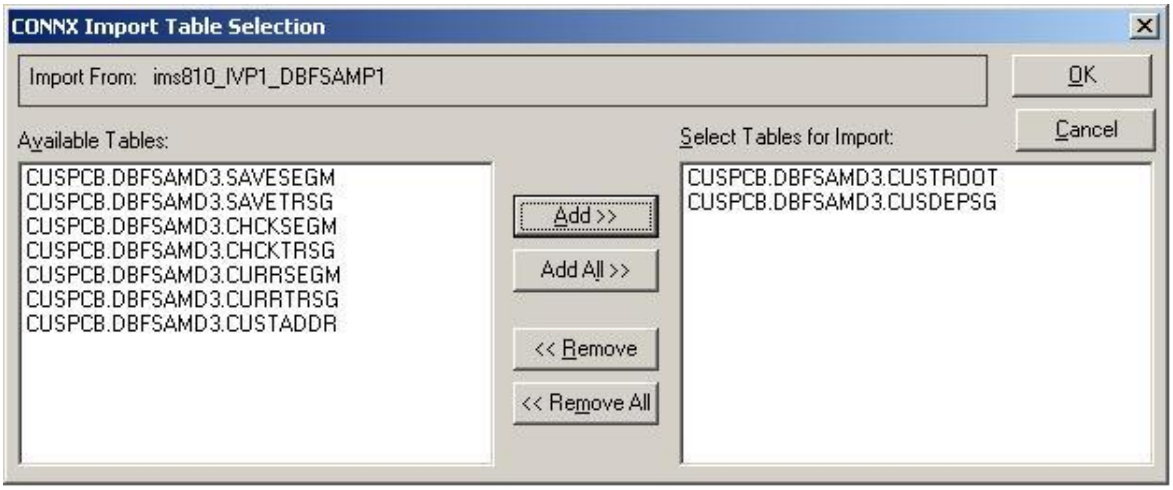


10. Enter the IMS subsystem ID in the **IMS SSID** text box and click **OK**. The **CONNX Import Table Selection** dialog box appears.

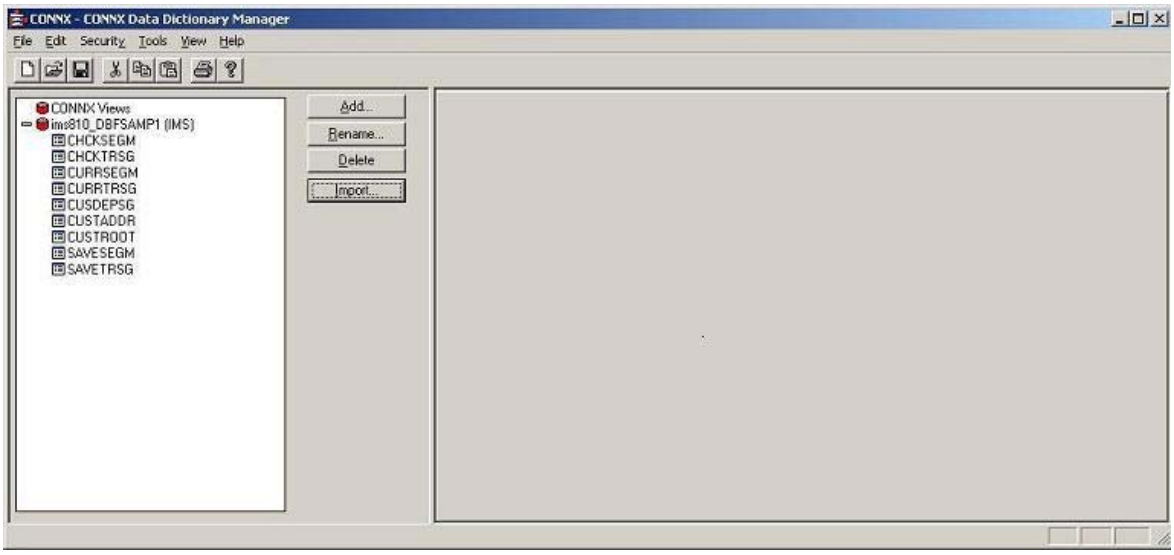


10. Select the tables (segments) you want to import from the **Available Tables** list. Click the **Add>>** button. The tables will move from the **Available Tables** list to the **Select Tables for Import** list. If

you want all the tables, click the **Add All>>** button to move all of the **Available Tables** to **Select Tables for Import**.

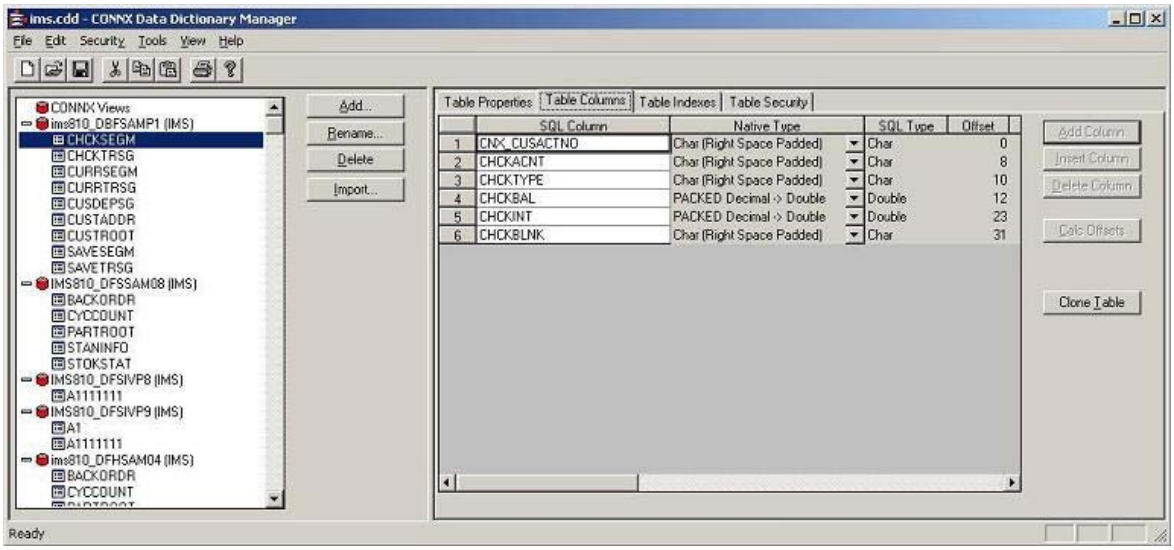


11. Click **OK**. The selected tables appear under the PSB name in the **CONNX Data Dictionary Manager** dialog box.

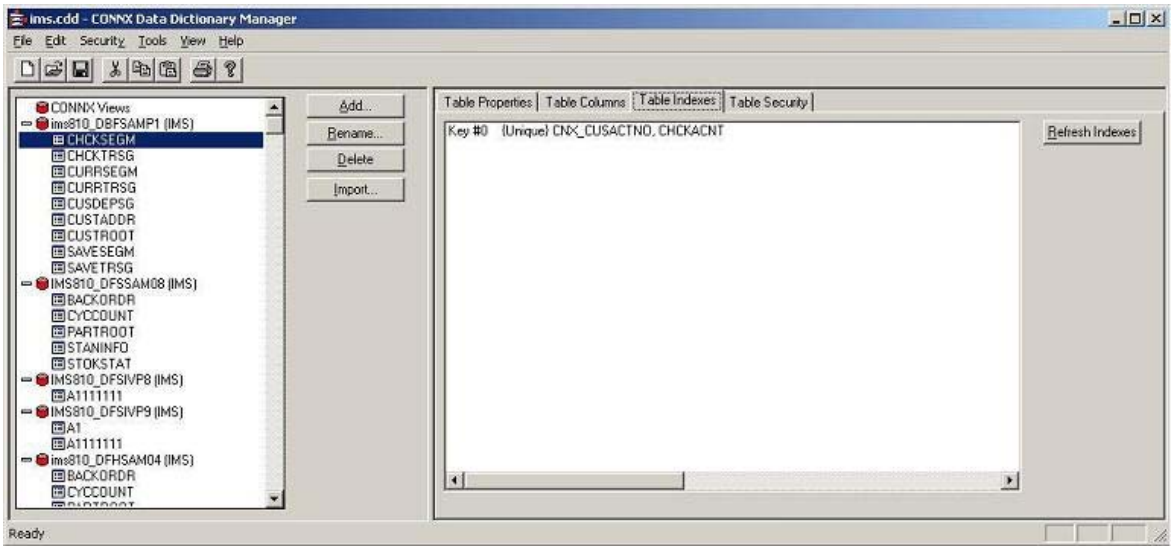


12. If you select one of the tables, the adjoining pane has tabbed table information.
    - You can edit some of the table properties under the **Table Properties** tab.

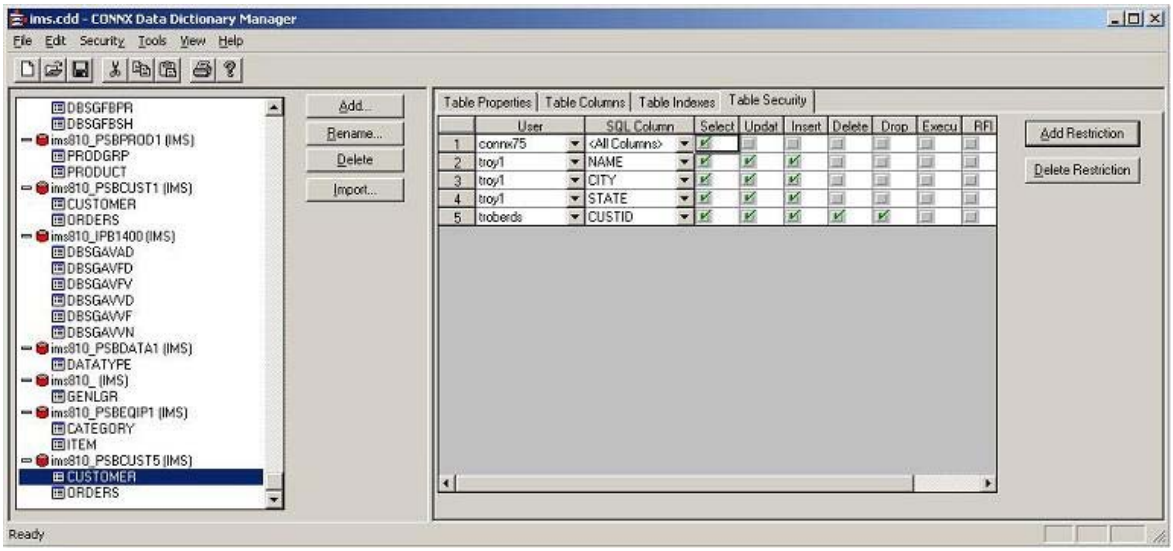
- You can clone the table under the **Table Columns** tab.



- You can refresh the indexes under the **Table Indexes** tab.



- You can update the table security restrictions under the **Table Security** tab.



13. Close the dialog box and click **OK** to save the changes.

IMS Index Text File Import Specification

CONNX for IMS fully supports the IBM® IMS™ Library Integrity Utilities for z/OS. The use of these utilities allows CONNX to automatically extract parent/child relationships between segments, key information and metadata. If the IMS Library Integrity Utilities are not available on a system, CONNX provides a mechanism to provide the necessary information via the combination of a COBOL Copybook and an Index Text Specification file.

The Index Text Specification file should be a Windows comma separated text file with a .txt extension. The format of each line of this file is as follows:

<PSBNAME>,<PCBNAME>,<DBDNAME>,<SEGMENTNAME>,<PARENTSEGMENTNAME>,<KEYNAME>,<KEYDATATYPE>,<KEYOFFSET>,<KEYLENGTH>,<KEYPRECISION>,<KEYSCALE>,<COMMENT>

One import text file may contain records for multiple PSBs. Only the PSB specified on the Import dialog will be displayed when importing.

IMS Index Text File Import Syntax and Description

Syntax	Description
<PSBNAME>	Name of the PSB.
<PCBNAME>	Name of the PCB.
<DBDNAME>	Name of the DBD.
<SEGMENTNAME>	Name of the Segment.
<PARENTSEGMENTNAME>	Name of this segment's parent. If the segment does not have a parent, this field must be blank.
<KEYNAME>	Name of the key field for this segment.
<KEYDATATYPE>	The data type of this segments key field. Possible values are P, C, F and H. Please see the conversion table below for a definition of these types.
<KEYOFFSET>	The starting position of the key field in the segment. This is a 0 based value - the first position is offset 0.
<KEYLENGTH>	The length of the key field.
<KEYPRECISION>	Precision of key field.
<KEYSCALE>	Scale of key field.
<COMMENT>	Column comment

Conversion of IMS field types to COBOL Native Types and SQL Types

IMS Field Type	COBOL Native Type	COBOL Usage Representation	SQL Type
P	DISPLAY_NUMERIC	PIC S9(n)V9(0) COMP-3	Decimal
C	CHARACTER	PIC X(n)	Char

F	BINARY	PIC S9(9) COMP	Integer
H	BINARY	PIC S9(4) COMP	SmallInt

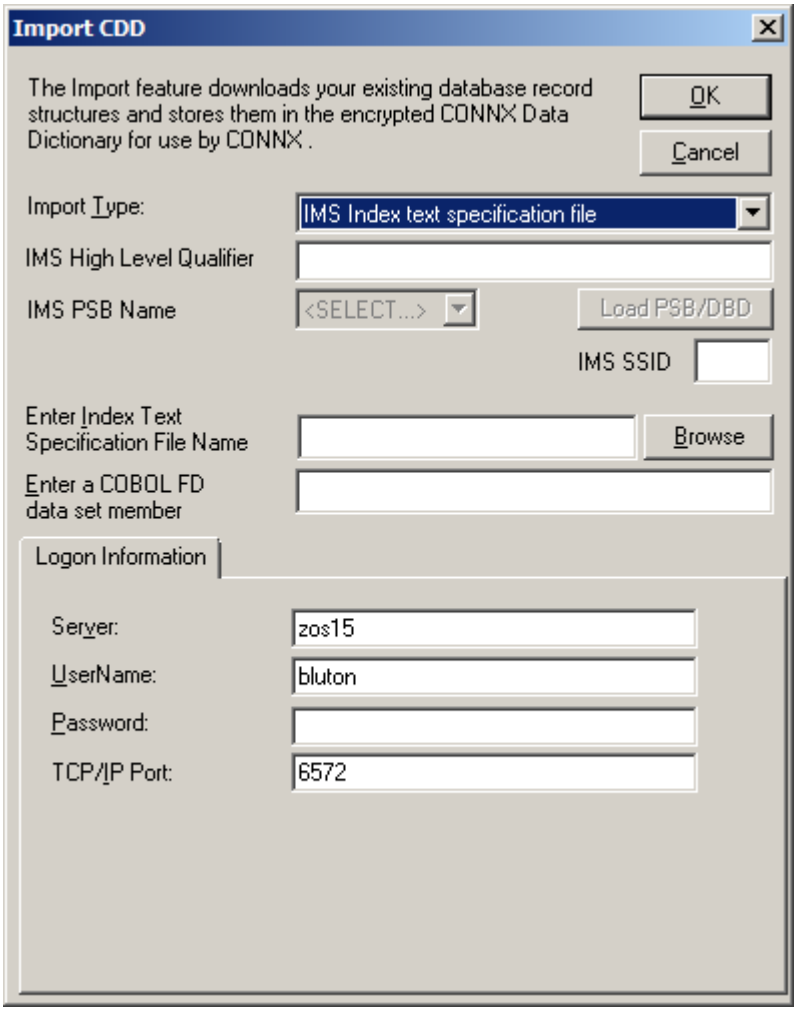
The following is an example of an IMS Index Text Specification file:

```
* this is a sample IMS Index Text Specification file
* lines that start with an '*' are comment lines and are not processed as part of the import
CNX1CUST,CUST1FL,CUST1FLD,CUSTOMER,,CUSTID,C,0,5,0,0
CNX1CUST,CUST1FL,CUST1FLD,ORDERS,CUSTOMER,ORDERID,P,0,4,7,3,This is a comment
```

The first two lines have an '*' in the first column and are therefore commented out. The file contains the definitions for two segments, CUSTOMER and ORDERS. ORDERS is a child of CUSTOMER. On the line for CUSTOMER, the PARENTSEGMENTNAME is empty (represented by the ',,' between the segment name and the key name) because it does not have a parent.

Note: The entire hierarchy for a segment must be represented in this file. If a segment has an entry for the parent segment name, there must also be an entry for the parent segment. If the hierarchy is not represented all the way to the root segment, an error will occur and the segment will not be imported.

To perform an IMS Index text specification file import, select this option from the import screen.



In addition to specifying the HLQ, SSID and PSB name, you must also specify the location and name of the Index Text Specification file as well as the fully qualified name of the COBOL copybook. For information on specifying COBOL copybooks, see the section Importing IMS files using COBOL FD. CONNX will use the information in the specification file rather than calling into the IMS Library Integrity Utilities for this information.

Related Topics

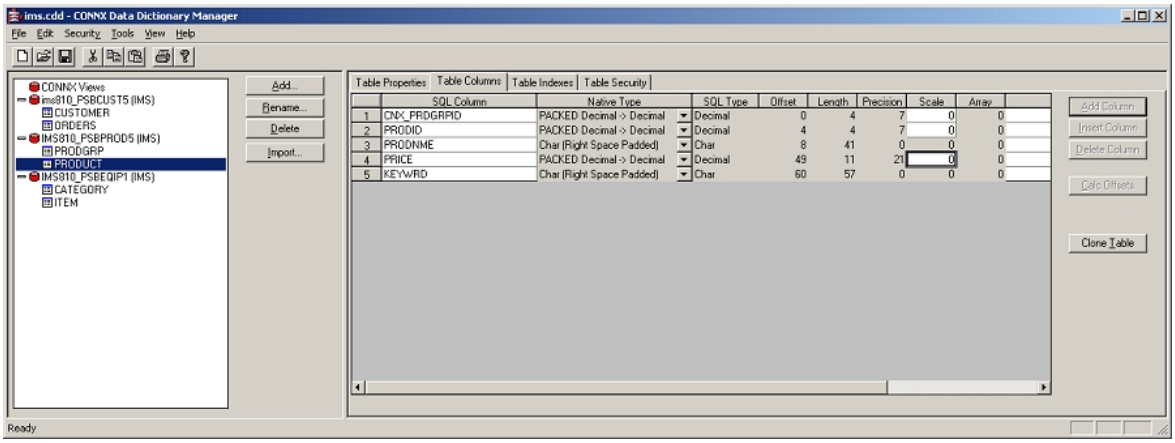
- Importing IMS files
- Importing IMS files using COBOL FD files

IMS Packed Decimal Data Fields

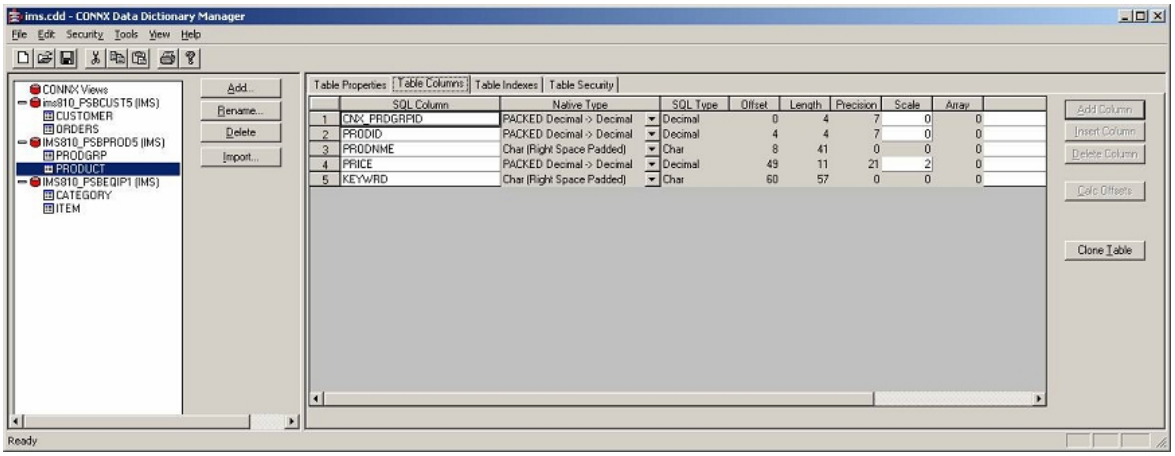
When you import IMS files, tables containing packed decimal fields do not contain the decimal point location information. If the packed decimal fields in your IMS tables contain non-integer values, specify the location of the decimal point using the CONNX Data Dictionary Manager.

To specify the decimal point location in the packed decimal field:

1. In the **CONNX Data Dictionary Manager**, click the IMS table containing the packed decimal field. Click the **Table Columns** tab.



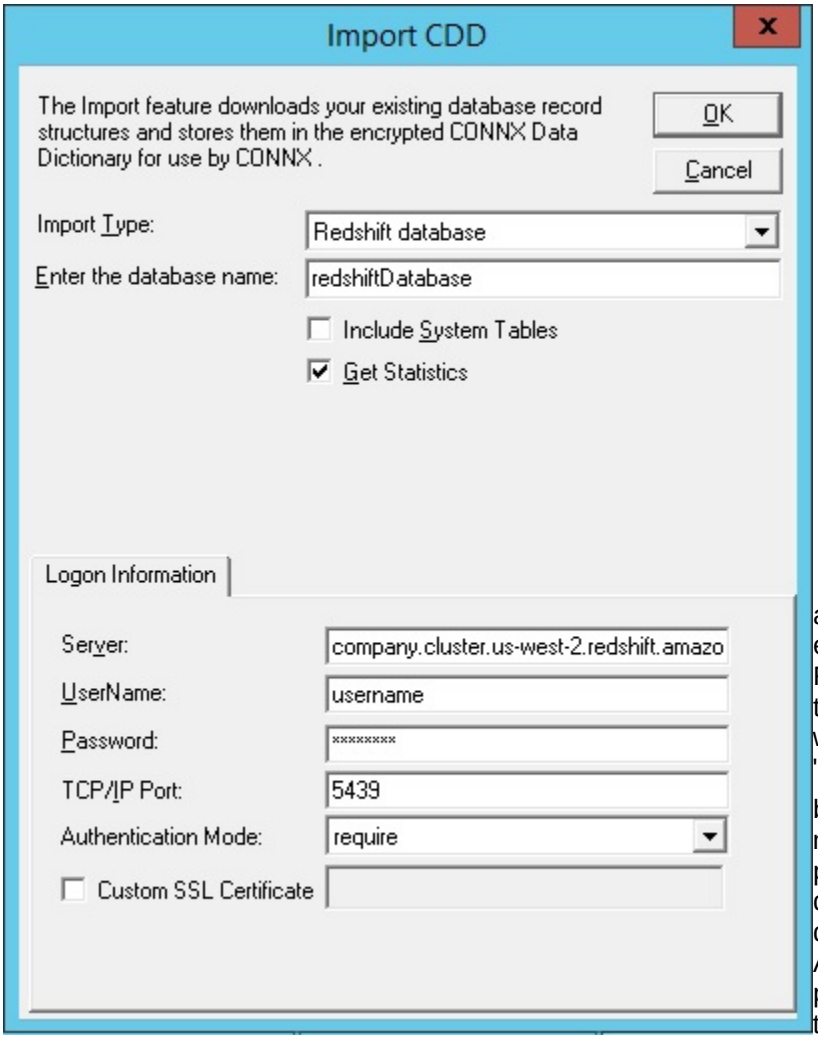
2. Scroll so you can see the both the field row containing the packed decimal data and the scale attribute column.
3. In **Scale** enter the number of decimal places the field contains. The PRICE field contains two decimal places so the **Scale** value has been changed to 2.



4. Save your changes.

Importing Redshift Database Tables

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The Import CDD dialog box appears. Select **Redshift database** from the Import Type list box. This is the name CONNX uses for imports of Redshift data.



3. Type the database that you created for your cluster in the **Enter the database name** text box.
4. Select the **Include System Tables** check box to import table definitions into the CDD.
5. Select the **Get Statistics** check box to identify the table sizes.
6. Type a server location, user name, and password under **Logon Information.**

a. **Server:** The endpoint of the Amazon Redshift cluster. This will be the url to the location of the cluster created with AWS it will likely end with ".redshift.amazonaws.com".

b. **UserName:** The name of a user account that has permission to connect to the database. This value is a database permission, not an Amazon Redshift permission, although you can use the master user account that you set up when you launched the cluster.

c. **Password**: The password for the user account to connect to the database.

d. **TCP/IP Port:** Port 5439 is listed in the TCP/IP Port text box by default. This is the port number that you specified when you launched the cluster. If you have a firewall, ensure that this port is open for you to use.

e. **Authentication Mode:** These are the SSL methods for authentication that can be used with the Redshift connection.

i. **disable:** SSL is disabled and the connection is not encrypted.

ii. **allow:** SSL is used if the server requires it.

iii. **prefer:** SSL is used if the server supports it. Amazon Redshift supports SSL, so SSL is used when you set to prefer.

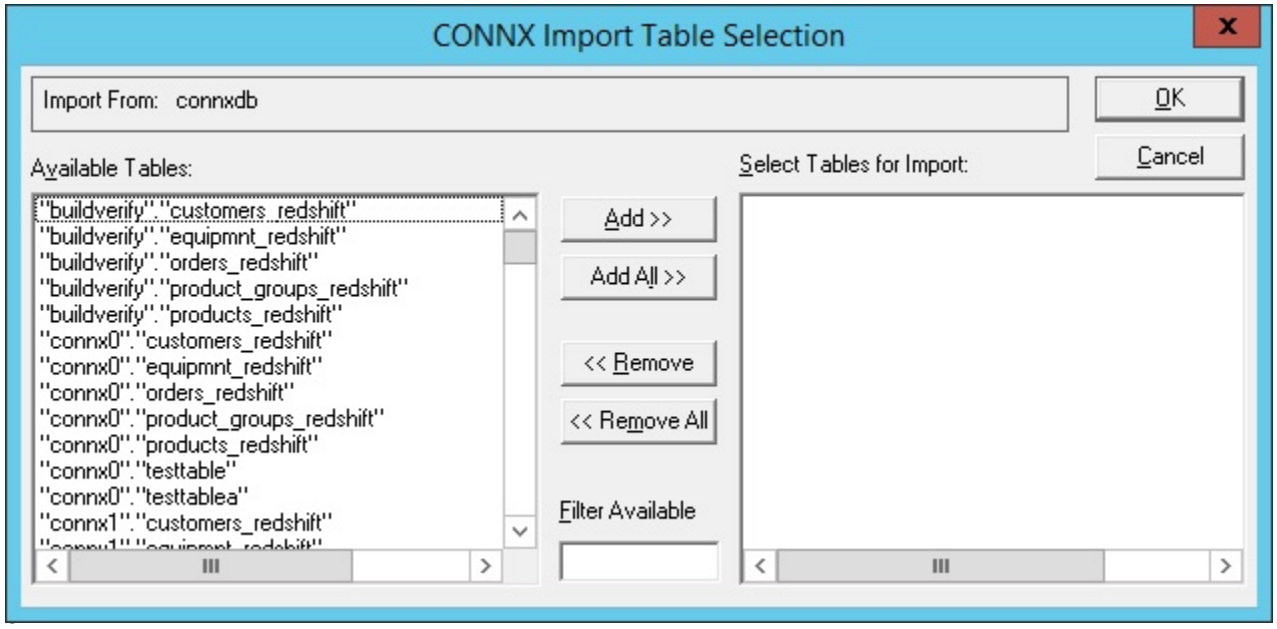
iv. **require:** SSL is required.

v. **verify-ca:** SSL must be used and the server certificate must be verified.

vi. **verify-full:** SSL must be used. The server certificate must be verified and the server hostname must match the hostname attribute on the certificate.

f. **Custom SSL Certificate:** Specify the Amazon Redshift certificate to used with server authentication here. The certificate must be in the root of the CONNX Installation directory. Or it the folder with the SSL Certificate can be specified using the registry key **"CERTPATH"** in the **"CONNX"** registry key.

7. Click the **OK** button. The **CONNX Import Table Selection** dialog box appears with a list of available table names. Click the **Add** or **Add All** button to move the tables to the **Select Tables**

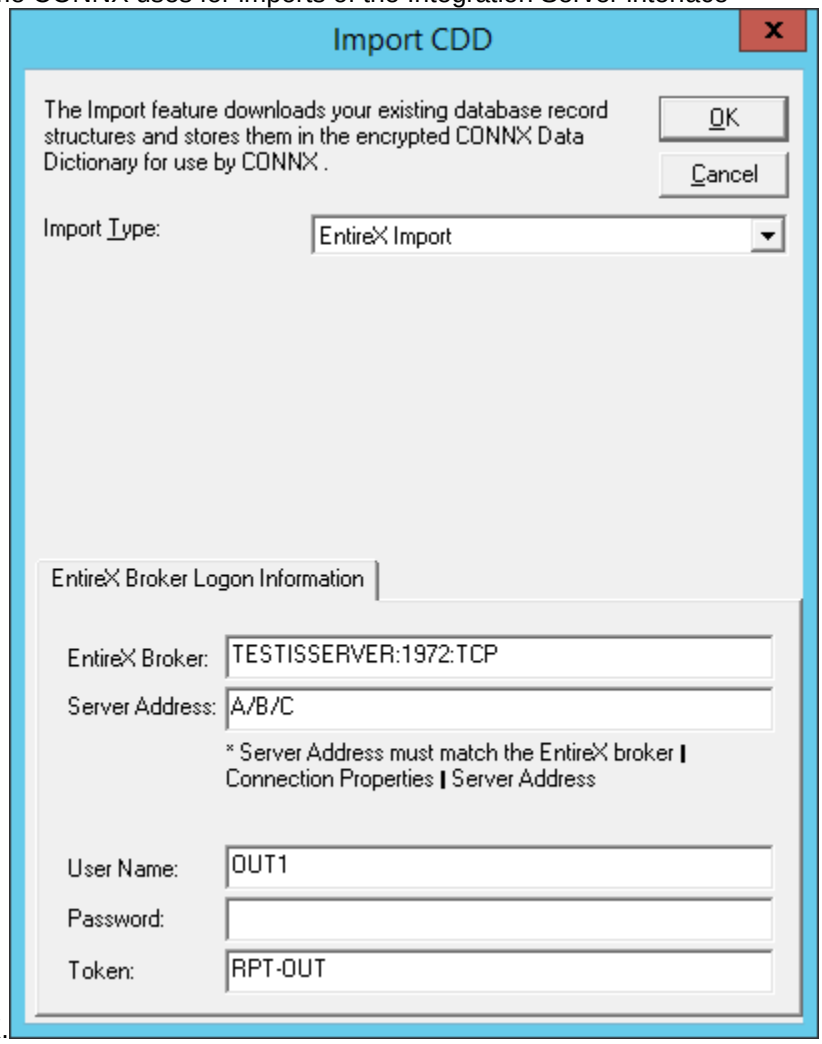
8. 

k the **OK** button to import the selected tables into CONNX. The Redshift database tables are added to the list of accessible objects in the CONNX Data Dictionary Manager window.

CONNX and Integration Server

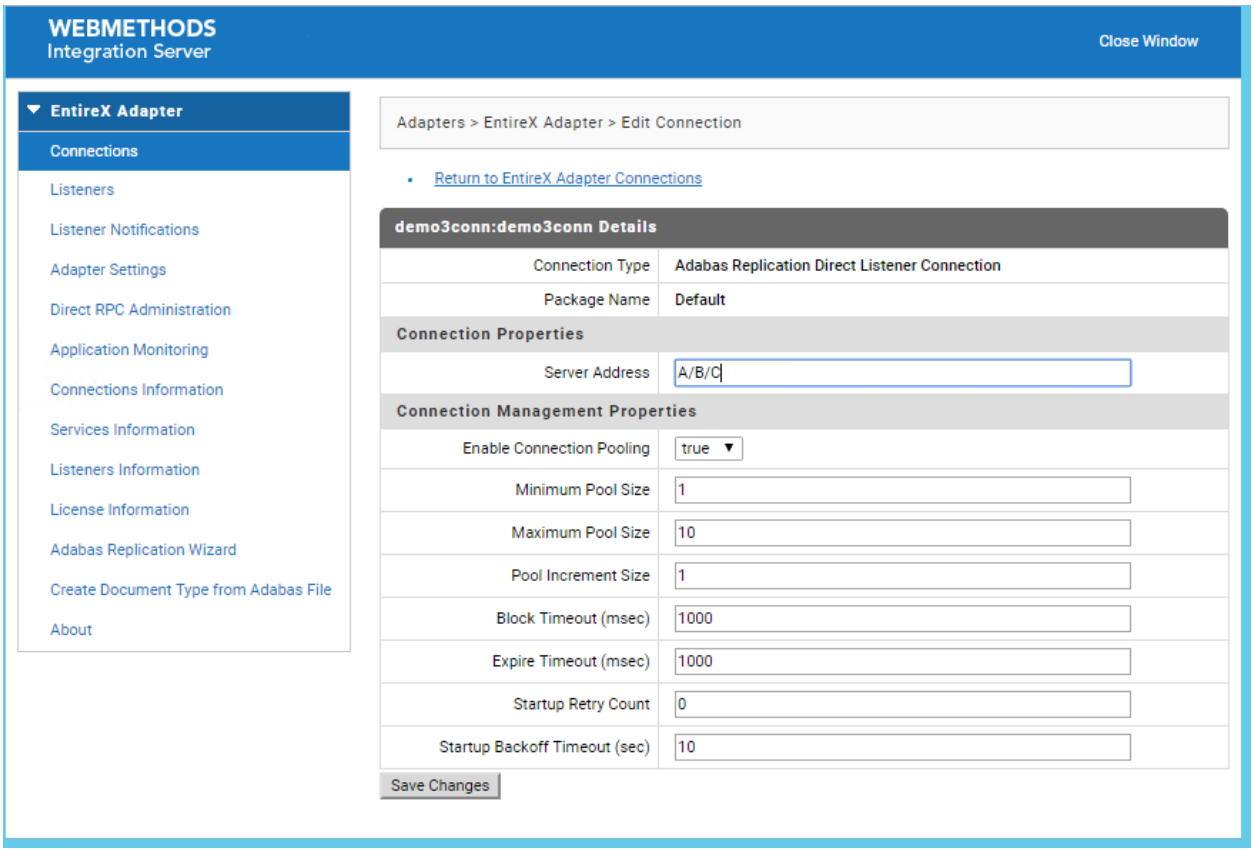
Importing Integration Server - EntireX

1. Click the **Import** button in the CONNX Data Dictionary Manager window.
2. The Import CDD dialog box appears. Select **EntireX Import** from the Import Type list box. This is the name CONNX uses for imports of the Integration Server interface EntireX.



3. **EntireX Broker** field is the Server name : Port number : Tranport protocol. This field used to specify where the EntireX listener is and what it is listening on. The Server name is the name of the server with Integration Server and EntireX installed and configured, the default EntireX listener is 1971, with TCP as the protocol. These can be changed for each EntireX listener that is configured.
4. **Server Address** is used to connect to a specific connection on the EntireX listener. This value is configured on the EntireX Connection Properties -> Server Address. There are three parts to the server address separated by forward slashes. The actual names are not that important, it is only important that the Import Screen Server Address matches the EntireX Connection Properties Server Address. In the example below, the EntireX connection Server Address is A/B/C so that is exactly what goes into the import Server Address - A/B/C.

*__Note__: These values are case sensitive.



5. **User Name, Password and Token** - on the first connection with the EntireX server, these are registered with EntireX. Then the EntireX server knows all following messages that have this User Name, Password and Token are sent from this connection.
6. Click the OK button to import the selected EntireX connection into CONNX. The EntireX database is added to the list of accessible objects in the CONNX Data Dictionary Manager window.

Chapter 3 - CONNX Security

CONNX Security Overview

The CONNX product has a comprehensive security model with the following features:

Established Security Measures Remain Intact

CONNX does not bypass security measures established by a database or operating system. For example, if a user has been given read-only access to a table, the user continues to have read-only access to the table through CONNX.

Secure Management

CONNX view, table, and column security levels can be assigned to an individual user, or to a group of users. Record security levels can be established through CONNX views. (See CONNX Views).

Seven Access Levels

CONNX supports seven access levels: Select, Update, Insert, Delete, Drop, Execute, and RFI.

Select	Read-only access
Update	Update queries
Insert	Insert new data
Delete	Delete data
Drop	Drop tables
Execute	Execute stored procedures
RFI	Enable Referential Integrity

The access levels can be applied to a table object, a view object, or a field within a table or a view.

Integrated Security/Database Account Management

CONNX provides a mechanism that enables users to specify a single CONNX user name and password to access multiple databases that may each require a distinct user name and password. The database-specific identification information is encrypted and stored in the CONNX Data Dictionary.

Secure Data Dictionary

The CONNX Data Dictionary is encrypted to only allow access to authorized users.

Maximum Security Option

This option allows only users defined in the CONNX Data Dictionary to access data, regardless of database permissions.

Read Only Default Access Option

This option restricts all access to all tables in the CONNX Data Dictionary to Read Only. This default security level can be overridden by specifying User or Column level security in the Security Panel for each CONNX object.

Related Topics

Adding Security to a Data Dictionary Entry

To add security to tables and columns

CONNX Users and Groups

Secure Access to Data on Multiple Platforms

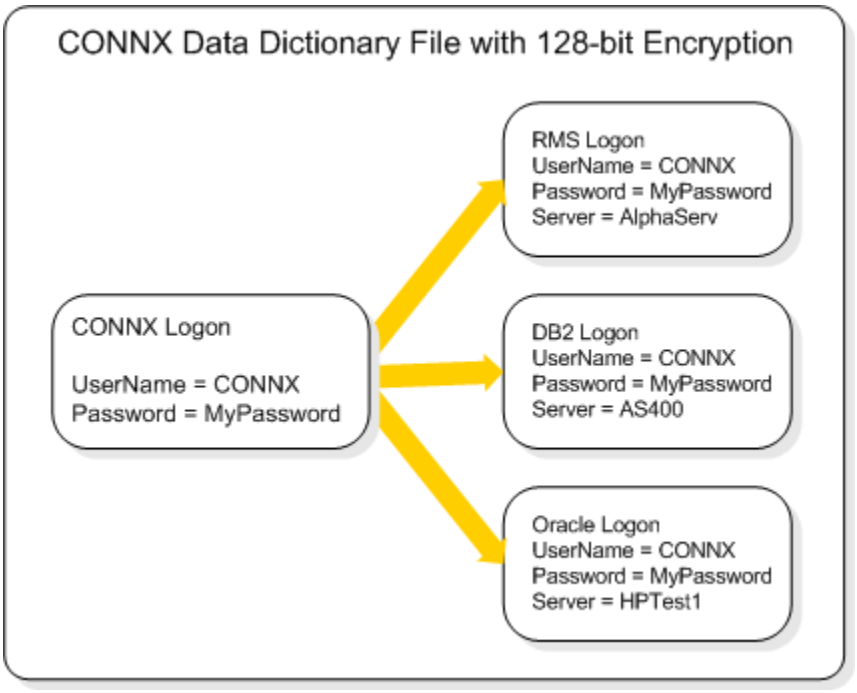
Security can be established in each database, although RMS and DBMS databases do not support column-level security. Such security can only be accomplished through CONNX. Additional security can be established in tables in CONNX using the CONNX Data Dictionary. CONNX can then restrict access at the column or table level for any user or all users. See To add security to tables and columns for information on adding security to tables and columns.

The creation of CONNX Views can also be used to limit access to data. This mechanism enables a database administrator to establish security on a cross-platform view, consisting of tables in different databases. See Security for CONNX Views for more information on CONNX Views.

Adding Security to a Data Dictionary Entry

The CONNX Integrated Security feature simplifies the process of logging on to multiple databases by using the same alias for one or more databases. CONNX prompts for a user name and password. If connecting to a single database, the logon name and password used can be the same as the one used with a source database.

CONNX Logon



To access multiple databases, it is recommended that one of the user names and passwords used with one of the databases be used as the CONNX user name and password. An entirely new user name and password may also be created for accessing the CONNX Data Dictionary (CDD).

The security levels in all CDD entries can be modified to protect specific types of data. The access rights of individuals or groups can also be modified within the CDD. Users or groups can be added or removed, passwords can be changed, and security levels can be added to specific views, columns, or tables within each type of database.

Related Topics

- CONNX Security Overview
- To add security to tables and columns
- CONNX Users and Groups

To add security to tables and columns

1. Select a table or view from the list box in the CONNX Data Dictionary Manager window.
2. Click the **Table Security** or **View Security** tab in the lower pane, and then click the **Add Restriction** button.
3. To add a restriction to a table or view, select the name of the user or group to restrict in the **User** list box.
4. To add a restriction to a column, select the name of the column to which you want to restrict access in the **SQL Column** list box. Select **<All Columns>** if the security entry is to apply to the entire table. Select **<Everyone>** and **<All Columns>** on all tables in order to maintain the highest levels of security.
5. Select the check boxes in the remaining columns to define access rights for each user or group as described in the following table.

CONNX CDD Definition of Access Rights

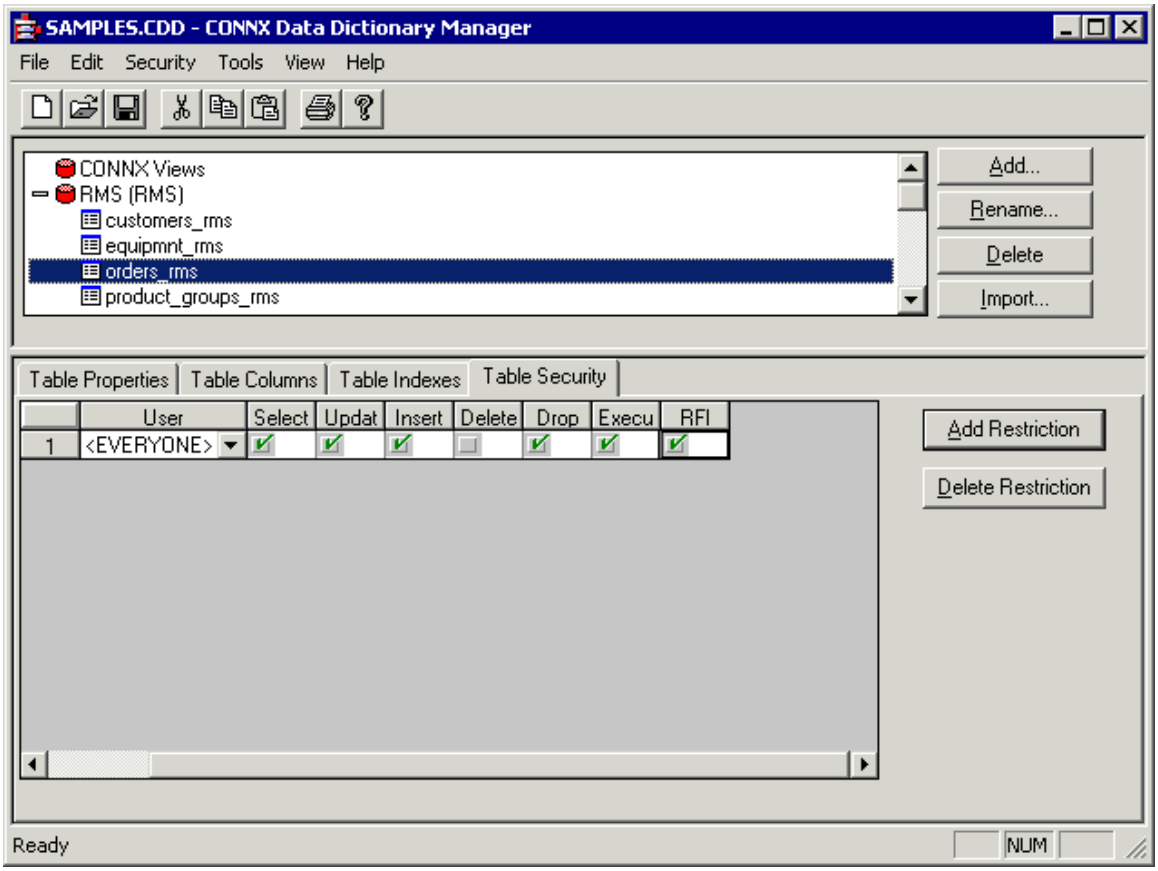
Access	Definition
Select	Read only - can run Select queries.
Update	Can update queries and can modify existing data, but not add new data.
Insert	Can insert new data.
Delete	Can delete data.
Drop	Can drop tables and keys to remove completely.
Execute	Can execute stored procedures.
RFI	Can enable referential integrity. (Not available in this release.)

Related Topics

- CONNX Security Overview
- Adding Security to a Data Dictionary Entry
- Accessing multiple databases
- Managing Applications

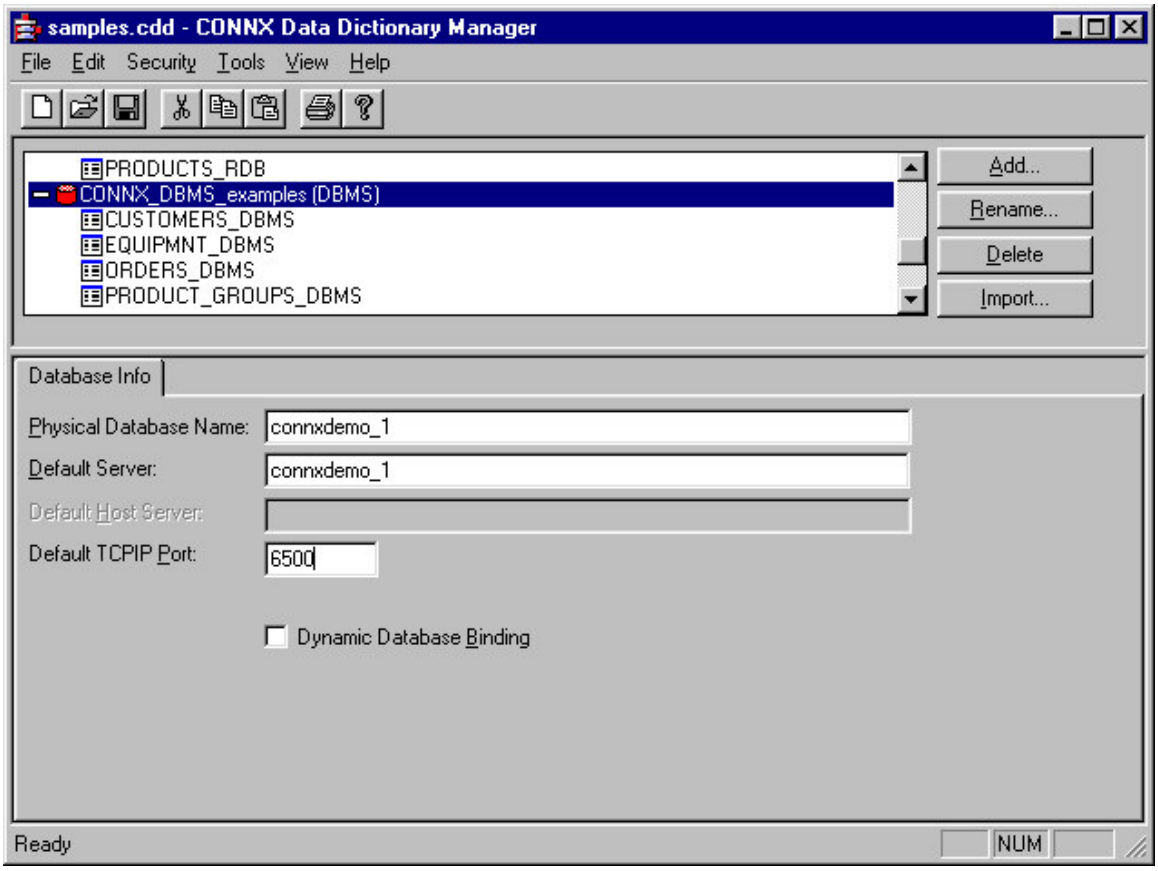
To delete an existing security restriction

1. Select a table or view from the list available in the CONNX Data Dictionary Manager window.
2. Click the **Table Security** or the **View Security** tab in the lower pane, select an existing restriction, and then click the **Delete Restriction** button.



To identify the server name

1. Click a database object in the CONNX Data Dictionary Manager window.
2. The current default server appears in the **Default Server** text box.

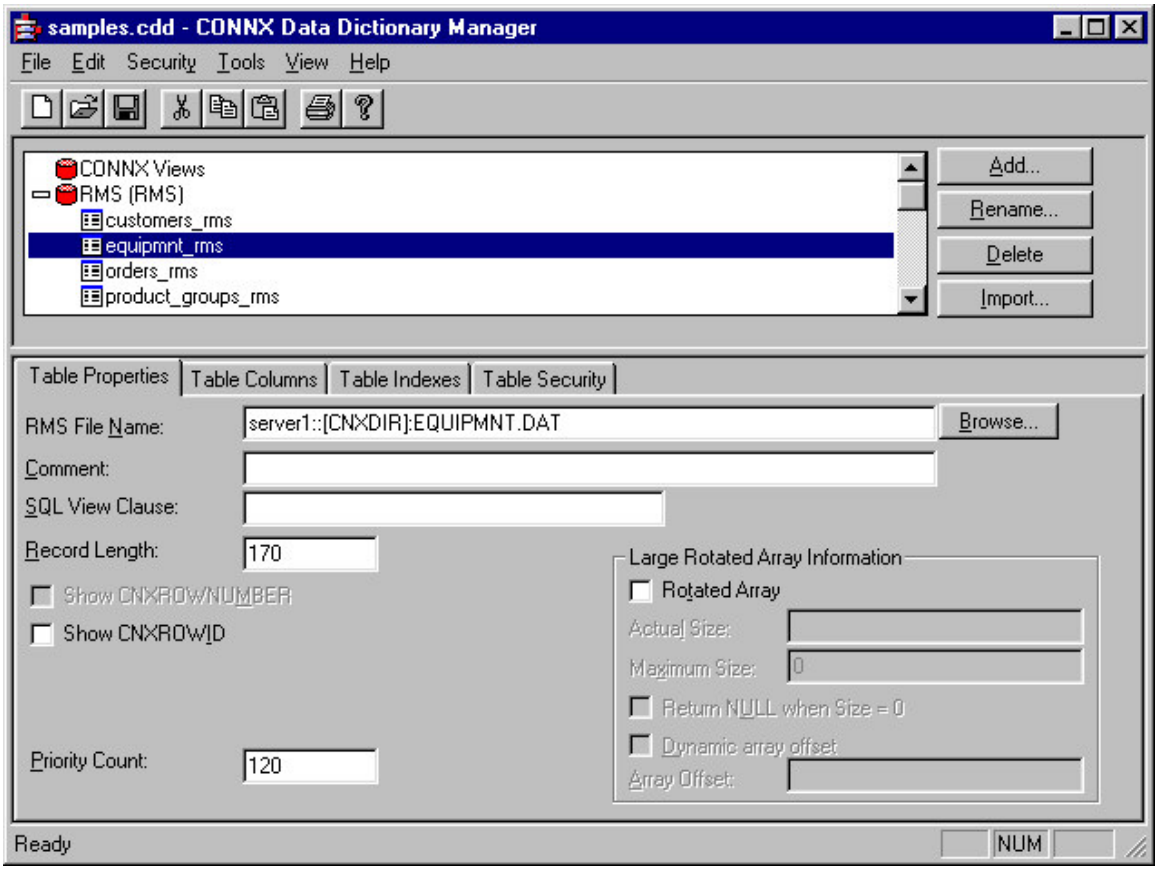


3. Select a database object to return to the list of available CDDs.

To override the default server name (RMS only - for multiple servers)

1. Click on the name of the RMS database object in the CONNX Data Dictionary window.
2. Type the server for the data file before the path and file name in the **RMS File Name** text box. The syntax is as follows:

```
server::[path]:file name
```



3. Select **Save** on the **File** menu to save the server name.

Important: *Specifying the server name in the RMS File Name text box should only be done in rare cases, as it severely decreases the performance of any access to that file. Use of this procedure is not recommended.*

Users and Groups

CONNX Users and Groups

Users can be managed within the CDD in two ways: as individual users and as members of user groups. It should be noted, however, that user groups cannot belong to other user groups.

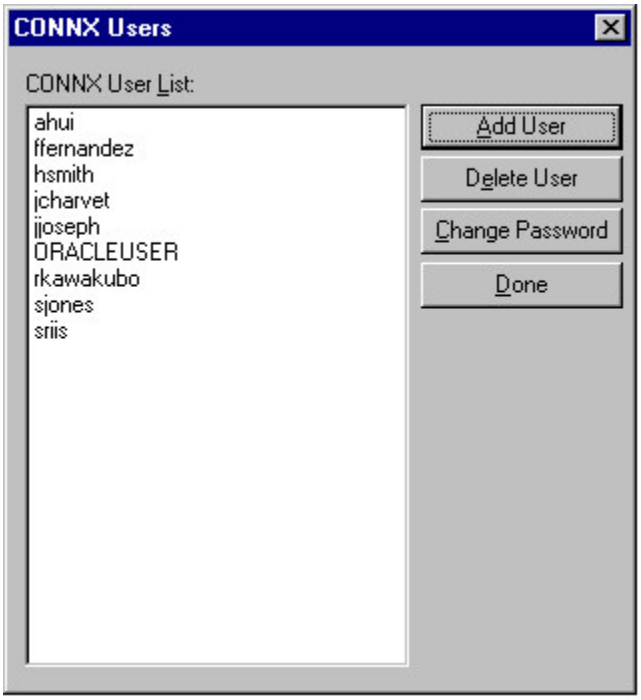
Related Topics

- To add a new user
- To change a user password
- To add a new group
- To add new users to a group
- To remove users from a group
- To delete an existing security restriction

To add a new user

Note: In order to create and drop users, the administrator of the task must belong to the CONNX Data Dictionary group "connx system admin" which appears automatically in the CONNX Groups and CONNX Group Users dialog boxes.

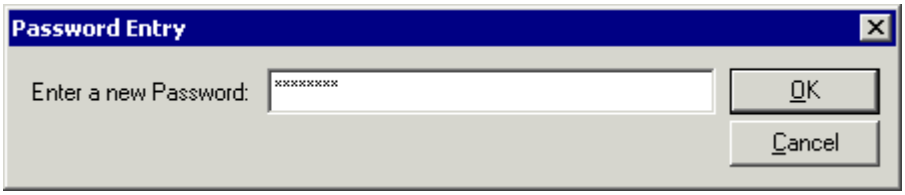
1. Click **Users** on the **Security** menu in the CONNX Data Dictionary Manager window.
2. The **CONNX Users** dialog box appears. Click the **Add User** button in the **CONNX Users** dialog box.



3. The **User Creation** dialog box appears. Type the user name of the new user in the **Enter new Username** text box.
4. Click the **OK** button, and then click the **Done** button in the **CONNX Users** dialog box.

To change a user password

1. On the **Security** menu in the CONNX Data Dictionary Manager window, click **Users**.
2. The **CONNX Users** dialog box appears. Select a user from the **CONNX User List**, then click the **Change Password** button in the **CONNX Users** dialog box.
3. The **Password Entry** dialog box appears. Type the new password in the **Enter Password** text box.

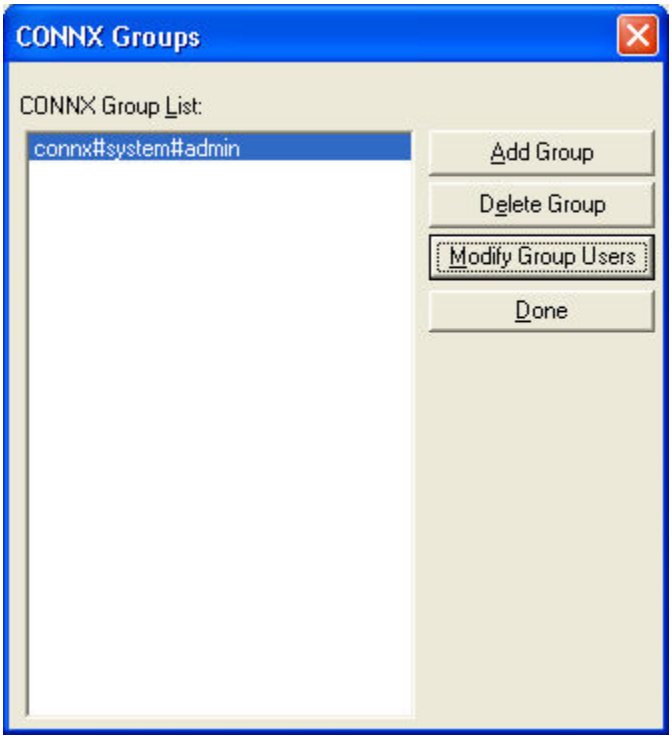


4. The **Password Confirmation** dialog box appears. Retype the password, and then click the **OK** button.
5. Click the **Done** button in the **CONNX Users** dialog box to return to the CONNX Data Dictionary Manager window.

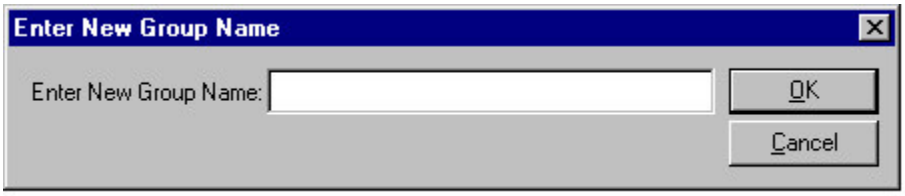
To add a new group

Note: In order to create and drop users, the administrator of the task must belong to the CONNX Data Dictionary group "connx system admin" which appears automatically in the CONNX Groups dialog box.

1. On the **Security** menu in the CONNX Data Dictionary Manager window, click **Groups**.
2. The **CONNX Groups** dialog box appears. Click the **Add Group** button in the **CONNX Groups** dialog box.



3. The **Enter New Group Name** dialog box appears. Type the name of the new group in the **Enter New Group Name** text box.

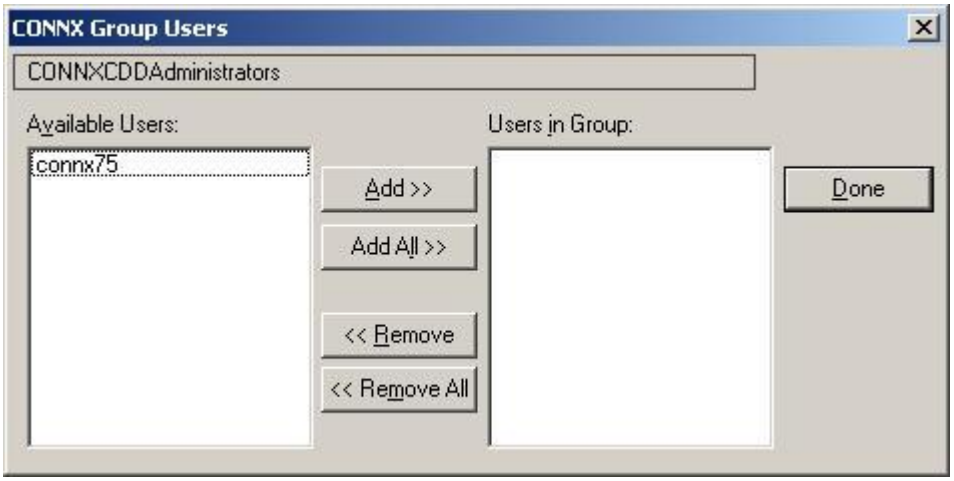


4. Click the **OK** button, and then click the **Done** button in the **CONNX Groups** dialog box.

To add new users to a group

Note: In order to create and drop users, the administrator of the task must belong to the CONNX Data Dictionary group **CONNXCDDAdministrators** *which appears automatically in the CONNX Groups and CONNX Groups Users dialog boxes.*

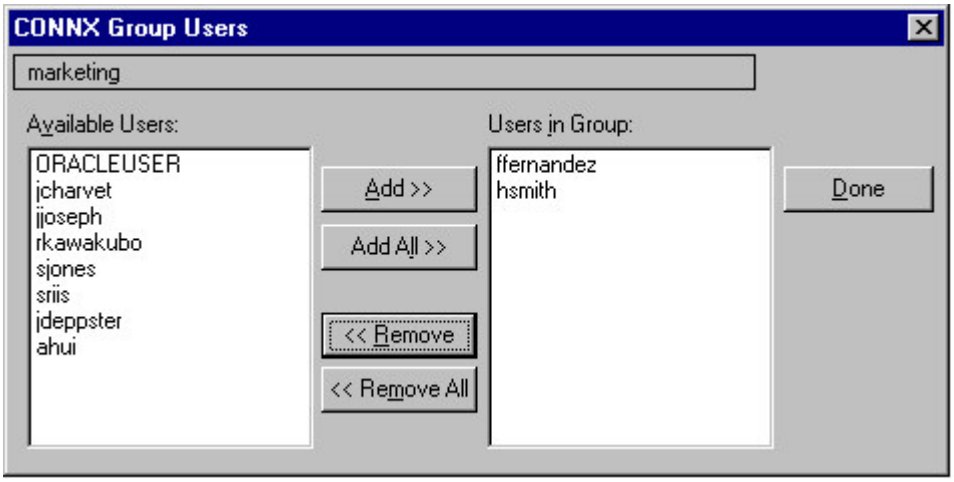
1. On the **Security** menu in the CONNX Data Dictionary Manager window, click **Groups**.
2. The **CONNX Groups** dialog box appears. Select a group from the **CONNX Group List** list box, then click the **Modify Group Users**.
3. The **CONNX Group Users** dialog box appears. Select an available user or users from the list box on the left, then click the **Add** or **Add All** button to add the user or users to the group.



4. Click the **Done** button in the **CONNX Group Users** dialog box, and then click the **Done** button in the **CONNX Groups** dialog box.

To remove users from a group

1. On the **Security** menu in the CONNX Data Dictionary Manager window, click **Groups**.
2. The **CONNX Groups** dialog box appears. Select a group from the **CONNX Group List** list box, then click the **Modify Group Users** button.
3. The **CONNX Group Users** dialog box appears. Select a user under the **Users In Group** list box on the right, then click the **Remove** or **Remove All** button.



4. Click the **Done** button in the **CONNX Group Users** dialog box, and then click the **Done** button in the **CONNX Groups** dialog box.

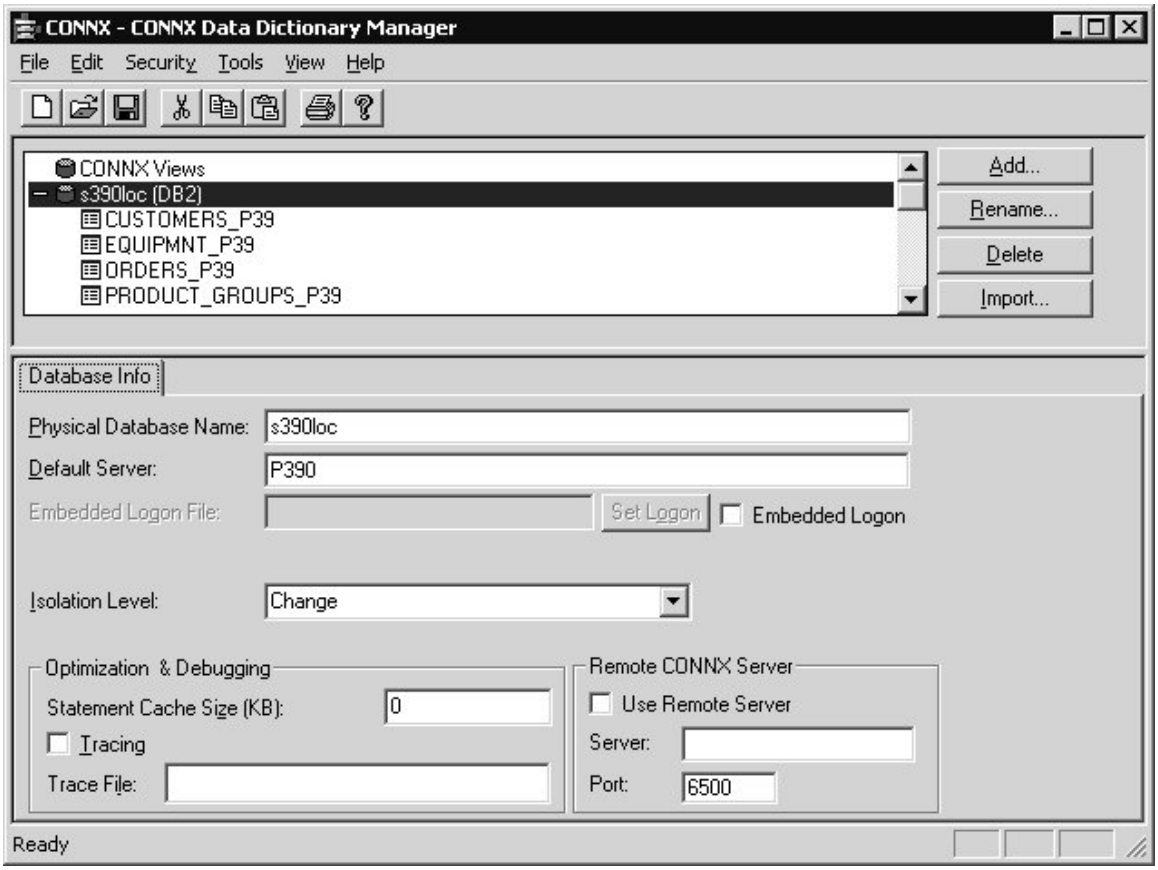
CONNX Embedded Logon

To use the CONNX Embedded Logon

The CONNX Embedded Logon feature provides a mechanism that enables end users to be identified with separate CONNX logons, but to use a single logon for a particular database.

This feature is useful in Web application and in other n-tier applications. The advantage of using an embedded logon is that the end user does not know the user ID and password for the underlying database. Security for the end user is controlled solely through the CONNX Data Dictionary.

1. In the CONNX Data Dictionary Manager window, select a database from the list in the upper pane.
2. The **Database Info** tab appears.



3. Click the **Embedded Logon** check box. The **Embedded Logon File** text box is enabled.
4. Enter a **UNC file path** as the location for the logon file. CONNX creates and stores the logon file at that location.

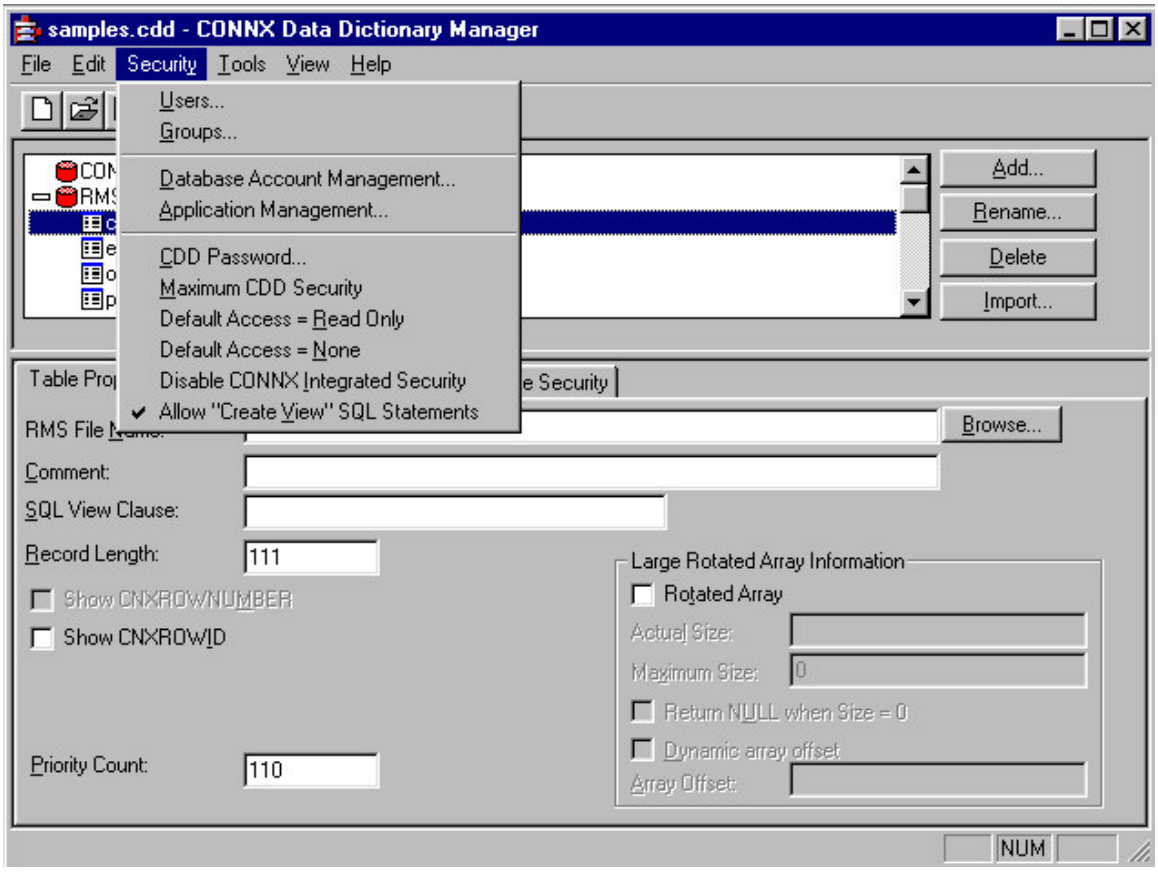
Security for CONNX Views

To disable the creation of views

View-creation capability is automatically enabled in CONNX and can be performed with the SQL statement CREATE VIEW. This capability can be a security risk if views are being created without the supervision of the system administrator. View creation, however, can be disabled, limiting the ability to administrator use only.

To disable the creation of views

1. Select **Allow "Create View" SQL Statements** on the **Security** menu in the CONNX Data Dictionary Manager window.



2. The feature is disabled and views cannot be created by anyone other than the system administrator.

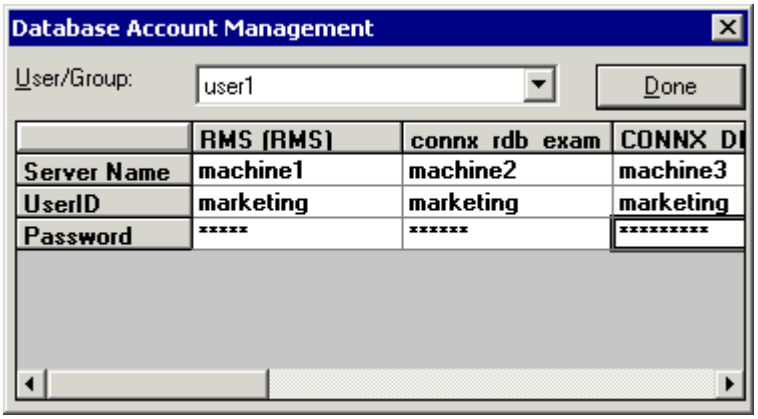
Accessing multiple databases

When using CONNX to access multiple databases, a separate logon is required for each database. To simplify the logon process, CONNX incorporates an integrated logon model requiring only one user name and password, and stores the logon information for all accessed databases.

When a user first logs in, their user name and password are entered and then stored in the CONNX Data Dictionary (CDD) for future use. If the password is changed on the host machine, CONNX prompts for the new password and then automatically updates and stores it in the CDD. The CDD is encrypted with a 128-bit key that guarantees protection of vital information.

To modify user names and password

1. On the **Security** menu in the CONNX Data Dictionary Manager window, click **Database Account Management**.
2. Select a user or group name from the **User/Group** list box in the Database Account Management window.



3. Select the type of database. Use the scroll bar to view the entire selection.
4. Select a user or group name or password to modify and then type the modified user name or password.
5. Click the **Done** button.

To change database owner name

In cases where additional security measures or rapid ownership changes are required, the database administrator can change the database owner name.

1. On the **Tools** menu, select **Change Owner.**

2. The **Select new CONNX table owner** dialog box appears.

3. Select a new owner from the **User** list box, and then click the **OK** button. The database owner changes to the selected user.

Disabling integrated security

If connecting to only one database, the CONNX Integrated Security feature may be turned off.

To disable Integrated Security

- Select the **Security** menu in the CONNX Data Dictionary Manager window and then click **Disable CONNX Integrated Security.**

To log on in the future, type the user name, password, and server name used for the database.

Application Management

Managing Applications

The CONNX Application Management feature is designed for companies that are accessing several different database types through CONNX. Application Management enables the definition of a subset of data sources that can be accessed from the list of currently available data sources. When defining a CONNX application name, a user can determine which databases are required by each application and type of use. When setting up a data source, administrators can specify applications and thereby define what is seen by type of database, rather than by table. CONNX only connects to the databases specified, instead of connecting to all of the databases in the CONNX Data Dictionary (CDD).

You can add applications, delete applications, add databases to an application, remove databases from an application, and modify the lists of available databases used within an application.

The benefits of this feature include savings on user licenses and increased security, since the administrator can establish the types of databases available on each client computer.

Related Topics

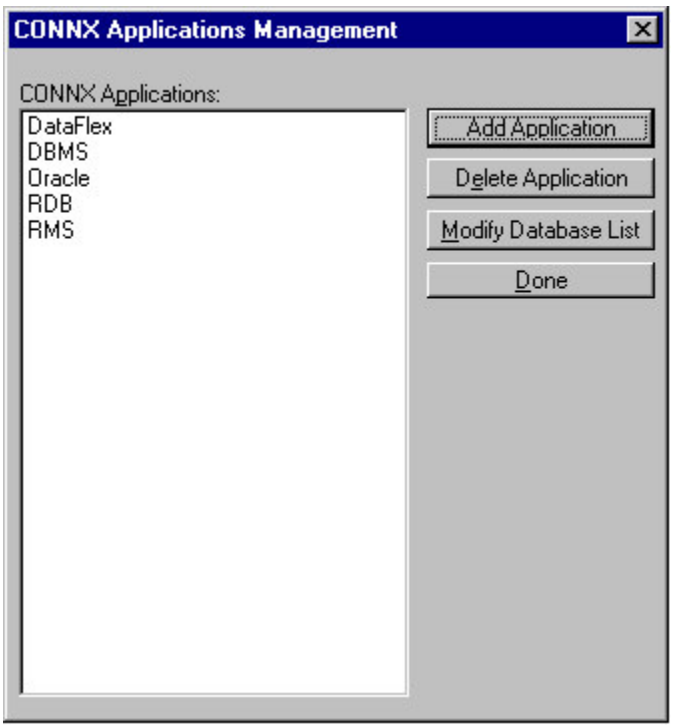
To add an application

To remove an application

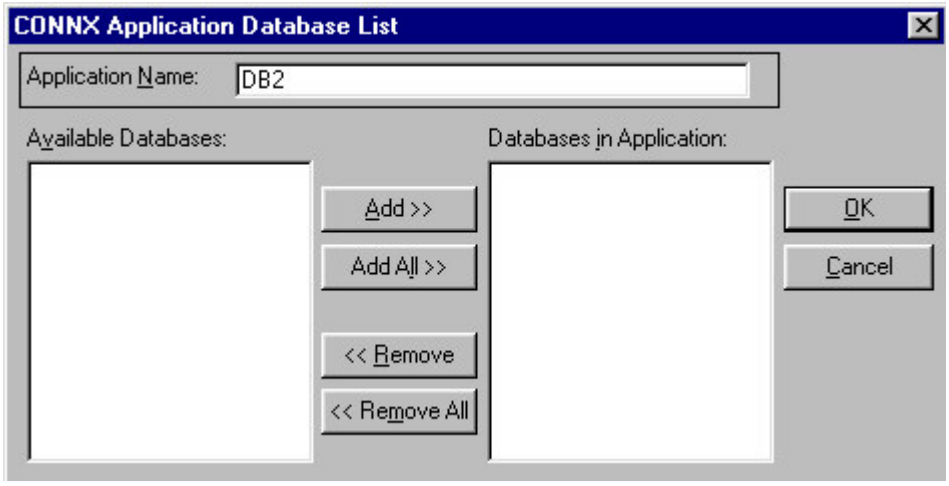
To remove a database from use within an application

To add an application

1. Select **Security** on the menu bar in the CONNX Data Dictionary Manager window, and then click **Application Management**.
2. The CONNX Application Management dialog box appears. Click the **Add Application** button in the **CONNX Applications Management** dialog box.



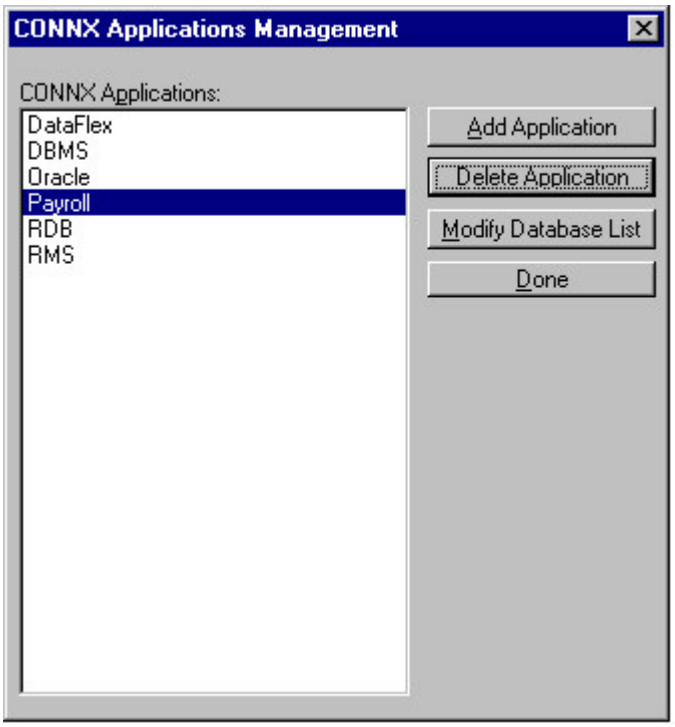
3. The **CONNX Application Database List** dialog box appears. Type an application name in the **Application Name** text box.



4. Select the required databases in the list box on the left, and then click the **Add** or **Add All** button.
5. Click the **OK** button to confirm the addition of the application databases and to connect to the specified databases.

To remove an application

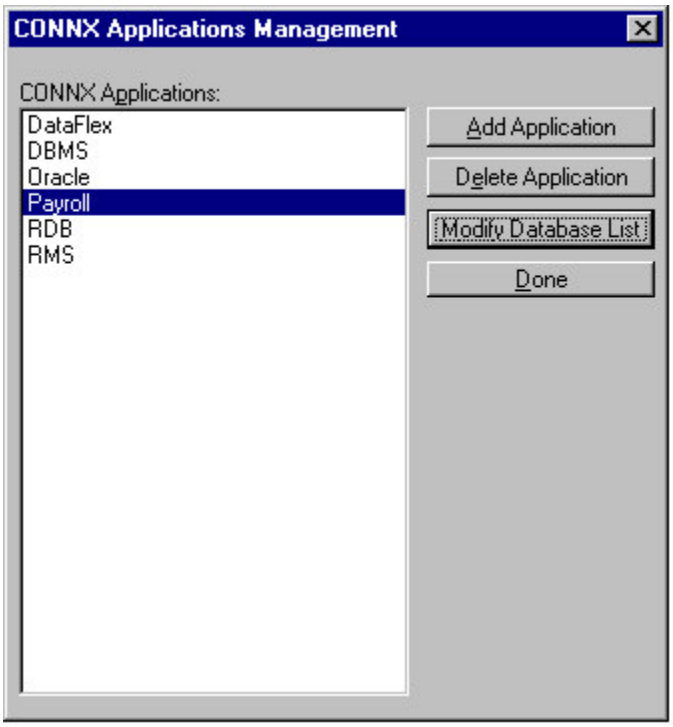
1. Select **Security** on the menu bar in the CONNX Data Dictionary Manager window, and then click **Application Management**.
2. The **CONNX Applications Management** dialog box appears. Select the name of the application to remove.



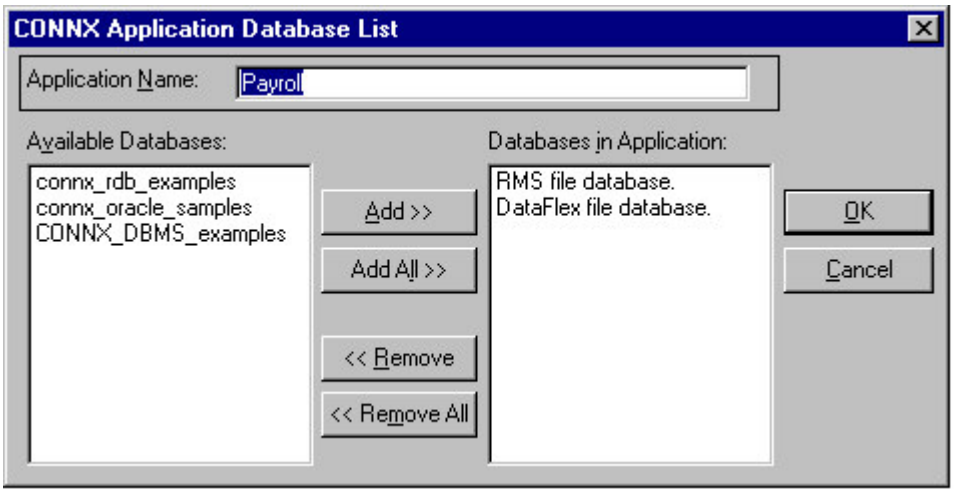
3. Click the **Delete Application** button to remove the application, and then click the **Done** button to return to the CONNX Data Dictionary Manager window.

To remove a database from use within an application

1. Select **Security** on the menu bar in the CONNX Data Dictionary Manager window, and then click **Application Management**.
2. The **CONNX Applications Management** dialog box appears. Select the application to modify, and then click the **Modify Database List** button.



3. The **Application Database List** dialog box appears with the list of available application databases.



4. Select the databases to remove from the list box on the right, and then click the **Remove** or **Remove All** button.
5. Click the **OK** button to confirm the removal of the selected databases from the application, and then click the **Done** button in the **CONNX Applications Management** dialog box to return to the CONNX Data Dictionary Manager window.

Maximum CDD Security

Maximum security

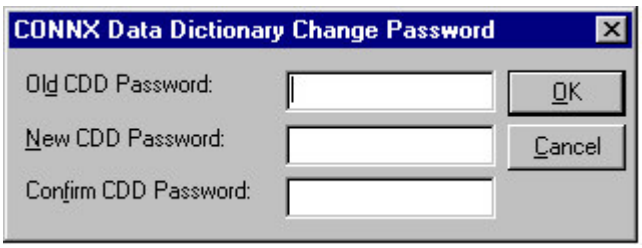
When the Maximum CDD Security option is established, CONNX refuses all logon attempts if the user is not defined in the CONNX Data Dictionary. Even if a valid database user name and password are specified, CONNX does not allow them to log on to the CDD.

Related Topic

- CONNX Security Overview
- Adding Security to a Data Dictionary Entry
- CONNX Users and Groups
- Accessing multiple databases
- Managing Applications
- To establish the maximum CDD security option

To establish the maximum CDD security option

1. Select **Security** on the menu bar in the CONNX Data Dictionary Manager window, and then click **Maximum CDD Security.**
2. The **CONNX Data Dictionary Change Password** dialog box appears. Click **OK**, and then type your old CDD password. If logging in for the first time, proceed to the next step.



3. Type a new CDD password, then retype to confirm the password.
4. Click the **OK** button to return to the CONNX Data Dictionary Manager window.

CONNX and VSAM: CICS Server-Side Security

CICS/VSAM Host / Client Security Overview

Default Option - Figure #1:

CONNX for CICS / VSAM is implemented as a Windows PC client and a set of CICS programs, including a Listener (Transaction NX00 / Program CNXRUN) and a Server (Transaction NXS0 / Program CNXVSAM). When a user connects from a client application through a supported interface (ODBC, JDBC, OLE DB or .NET), the client CONNX user ID / password is mapped to a CICS user ID / password via the CONNX Data Dictionary.

The encrypted CICS user ID / password is sent via TCP/IP to the CONNX CICS TCP/IP Listener Transaction NX00, which decrypts the user ID / password and executes the CICS VERIFY PASSWORD command, which in turn invokes the installed external security manager software (RACF, ACF/2, CA-Top Secret) to verify the user ID / password. Refer to steps 1 through 3 in Figure 1.

If the user ID / password sent from the CONNX PC client is valid, the CONNX CICS / TCP/IP listener transaction NX00 starts CONNX server transaction NXS0 using the CICS command:

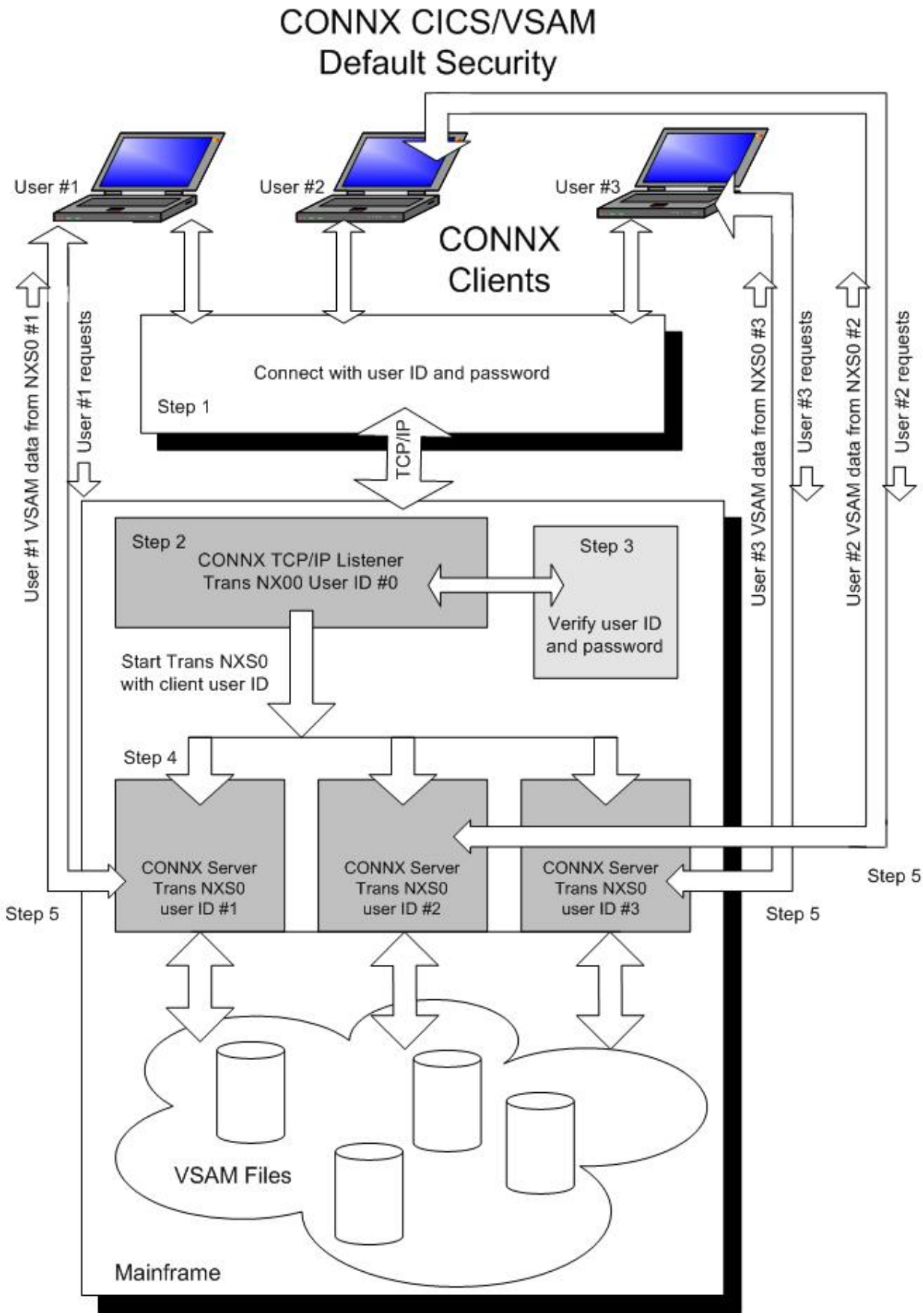
```
START TRANSID (NXS0) USER(user ID)
```

Where user ID is the CICS user ID sent from the CONNX client. In order for this command to succeed, the user ID which started the NX00 transaction (user ID #0) must have surrogate authority defined for the CICS client user ids (user IDs #1, #2, and #3). This prerequisite is documented in the CONNX User Reference Guide. If the start of transaction NXS0 succeeds, then transaction NXS0 sets up a separate TCP/IP connection to the invoking CONNX PC client.

CONNX PC Client requests and VSAM data responses flow back and forth directly from the CONNX PC client to the dedicated NXS0 transaction. Refer to steps 4 and 5 in Figure 1. For the default case, the host-side (RACF, ACF/2, CA Top Secret) security rules defined for the NXS0 user IDs (#1, #2, or #3) determine the type of VSAM file access granted to the CONNX PC client.

The default security option observes the host-side security rules defined for CICS user ID / password verification, and for each CICS user ID or group of user IDs for VSAM file access. In order for these rules to be enforced, each CICS user ID sent from the CONNX PC client must be attached to a separate NXS0 transaction via the START TRANSID command. This option requires that the user ID that starts transaction NX00 must have surrogate authority to start transaction NXS0 for every CICS user ID sent from the CONNX client.

Figure 1:


CONNX CICS/VSAM
Default Security
User #1
User #2
User #3
CONNX
Clients
Connect with user ID and password
Step 1
TCP/IP
User #1 VSAM data from NXS0 #1
User #1 requests
User #3 VSAM data from NXS0 #3
User #3 requests
User #2 VSAM data from NXS0 #2
User #2 requests
Step 2
CONNX TCP/IP Listener
Trans NX00 User ID #0
Step 3
Verify user ID
and password
Start Trans NXS0
with client user ID
Step 4
CONNX Server
Trans NXS0
user ID #1
CONNX Server
Trans NXS0
user ID #2
CONNX Server
Trans NXS0
user ID #3
Step 5
Step 5
Step 5
VSAM Files
Mainframe

Alternate Option - Figure #2:

The host-side security rules for user ID / password verification and dataset access can be enabled or disabled for CONNX client-server connections via a CONNX environment variable (CNXNOPREAUTHORIZE). Setting CNXNOPREAUTHORIZE to a non-zero value instructs the CONNX CICS TCP/IP Listener and Server programs to bypass CICS user ID / password verification. Transaction NX00 starts CONNX server transaction NXS0 using the CICS command:

```
START TRANSID (NXS0)
```

If the start succeeds, then client requests and VSAM data flow from the CONNX PC client to the NXS0 transaction and back via a dedicated TCP/IP socket connection. In this case, the host-side security rules defined for the NX00 user ID (#0) attach to each NXS0 transaction started on behalf of a CONNX client, and determine the type of VSAM file access granted to the CONNX PC client.

This security option bypasses the need to define surrogate user ID relationships, but all dataset access derives from the single user ID which starts transaction NX00. An advantage to this approach is that host-side VSAM data set security rules for CONNX PC clients need only be defined for one user ID.

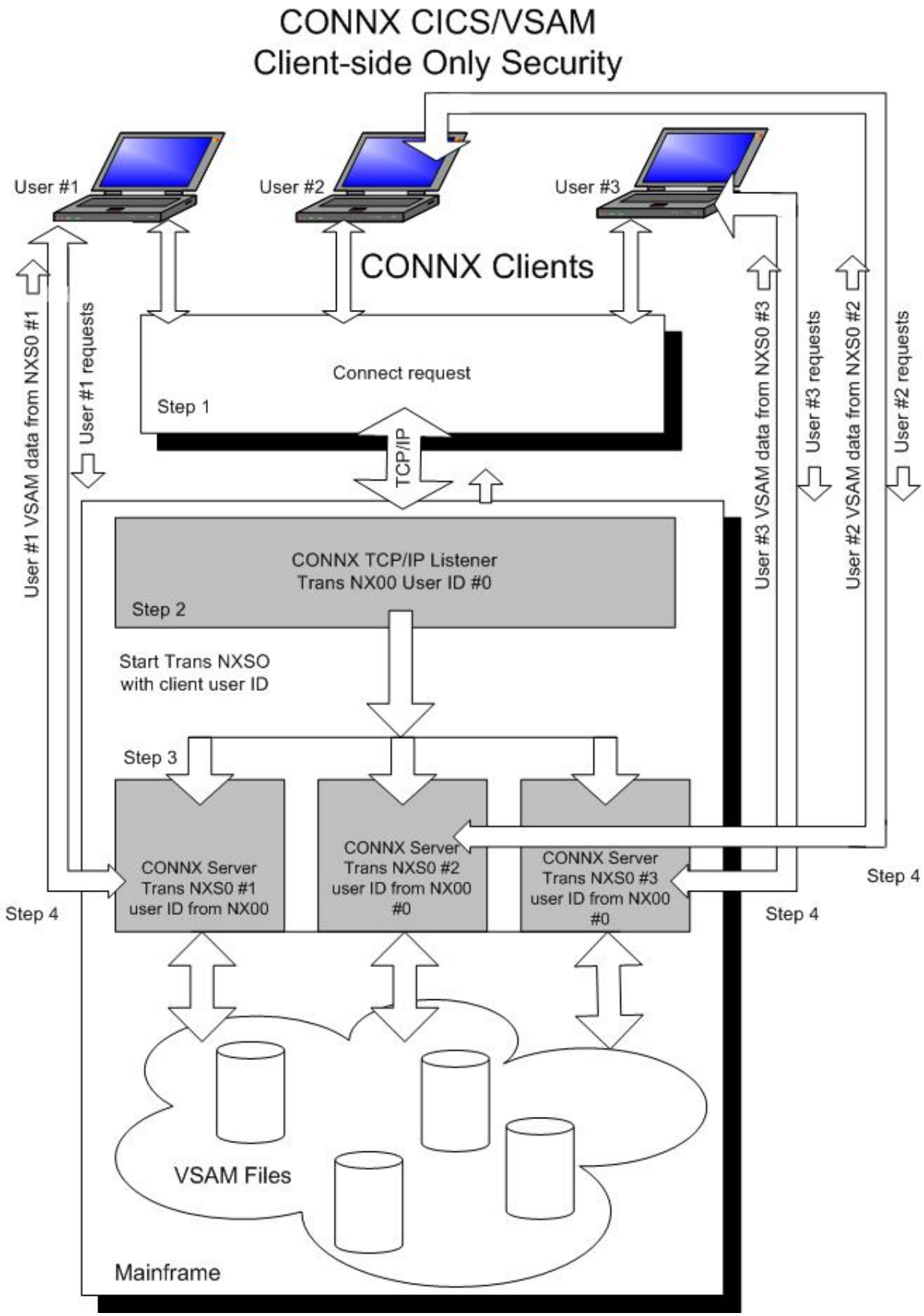
CONNX Client-Side Security Enhances Host-Side Security

For both options, the CONNX Administrator should consider taking advantage of the client-side security features implemented in the CONNX Data Dictionary (CDD). The first line of defense is only to import selected VSAM files into the CDD. Additionally, the CONNX Administrator can restrict access to VSAM files based on CONNX user IDs and groups defined via the security menu features in the CONNX Data Dictionary Manager.

In most cases, a physical VSAM file is composed of multiple logical files or sub-files. The CONNX Data Dictionary Manager can be used to define and enforce security rules based on these logical files, as well as on the underlying physical data sets. Further, column- and row-level security can be implemented by defining one or more CONNX views against the imported VSAM physical or logical files, and by authorizing individual or groups of CONNX user IDs to execute the CONNX client-side views.

The CONNX client-side approach to security complements and enhances most host-side security products, such as RACF, ACF/2, or CA-Top Secret, which implement access rules on a per-physical VSAM file basis. In all cases, the security rules defined in the CONNX Data Dictionary take precedence over the host-side security rules. For more information on CONNX client-side security, refer to the other topics described in the CONNX Security section.

Figure 2:


CONNX CICS/VSAM
Client-side Only Security
User #1
User #2
User #3
CONNX Clients
User #1 VSAM data from NXS0 #1
User #1 requests
Connect request
Step 1
TCP/IP
User #3 VSAM data from NXS0 #3
User #3 requests
User #2 VSAM data from NXS0 #2
User #2 requests
CONNX TCP/IP Listener
Trans NX00 User ID #0
Step 2
Start Trans NXSO
with client user ID
Step 3
CONNX Server
Trans NXS0 #1
user ID from NX00
CONNX Server
Trans NXS0 #2
user ID from NX00
#0
CONNX Server
Trans NXS0 #3
user ID from NX00
#0
Step 4
Step 4
Step 4
VSAM Files
Mainframe

CONNX TCP/IP Listener and Server Security

The CONNX TCP/IP CICS VSAM listener is invoked by the CONNX NX01 CICS transaction. When you log on to CICS and start the listener by entering

```
NX01 START
```

the CONNX NX01 transaction starts the NX00 CICS listener transaction (program CNXRUN). The CONNX CICS TCP/IP listener program waits for incoming connect requests from the CONNX PC clients. When a connect request is received, the listener program issues a START TRANSID for NXS0 (program CNXVSAM, the CONNX CICS/VSAM server). The CONNX CICS/VSAM server connects back to the invoking CONNX PC client on a new TCP/IP socket.

The security privileges of the CONNX listener derive from the userid which invokes transaction NX01(CNXCFG) to start transaction NX00(CNXRUN). Program CNXRUN is a TCP/IP listener, which acts as a broker to start transaction/program NXS0/CNXVSAM. If the NX01 transaction is automatically started at CICS startup, the CONNX listener program (CNXRUN) inherits the security attributes of the CICS default user ID, as defined in the CICS System Initialization Table (SIT), or by the run-time DFLTUSER startup parameter.

When the CONNX PC user enters a user ID/password in the client logon dialog box, both values are encrypted and sent to the CONNX Listener (CNXRUN). The CONNX Listener decrypts the user ID and password; if the user ID and password are non-blank, the listener issues a CICS VERIFY PASSWORD command. If the command succeeds, the listener starts the NXS0 transaction with the decrypted USER ID parameter. If the user ID sent from the CONNX client is non-blank and different from the user ID which started the NX01 and NX00 transactions, the initial (NX01/NX00) user ID must have **surrogate user** ID privileges to start the NXS0 transaction. If the surrogate user ID privilege is not defined, CICS returns a NOTAUTH (not authorized) condition, and the CONNX server is not started. If the initial user ID and the user ID sent by the client match, the surrogate user ID privilege requirement is met, since all CICS user IDs are surrogates of themselves.

To invoke the CONNX TCP/IP CICS VSAM Listener

1. Log on to CICS. (See Step 5 in the CONNX Installation Guide.)
2. Start the listener by typing

   ```
   NX01 START
   ```

3. The CONNX NX01 transaction starts the NX00 CICS listener transaction (program CNXRUN).

The CONNX CICS TCP/IP listener program waits for incoming connect requests from the CONNX PC clients. When a connect request is received, the listener program issues a START TRANSID for NXS0 (program CNXVSAM, the CONNX CICS/VSAM server). The CONNX CICS/VSAM server connects back to the invoking CONNX PC client on a new TCP/IP socket.

A description of the user ID and terminal ID is excerpted from the following CICS document:

Title: *CICS Application Programming Reference*

Document Number: SC33-1688-31

Build Date: 01/18/00 11:28:00 Build Version: 1.3.0

Book Path: /home/publib/epubs/book/dfhjap43.boo

http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/DFHJAP43/CCONTENTS

The excerpt included below can be found at the following Web address:
http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/DFHJAP43/1.232

"USERID (data-value)

Specifies the userid under whose authority the started transaction is to run, if the started transaction is not associated with a terminal (that is, when TERMID is not specified). This is referred to as *userid1.*

If you omit both TERMID and USERID , CICS uses instead the user ID under which the transaction that issues the START command is running. This is referred to as *userid 2.*

By using either *userid1* or *userid2*, CICS ensures that a started transaction always runs under a valid user ID , which must be authorized to all the resources referenced by the started transaction.

CICS performs a surrogate security check against *userid2* to verify that this user is authorized to *userid1.* If *userid2* is not authorized, CICS returns a NOTAUTH condition. The surrogate check is not done here if USERID is omitted. "

Further information on CICS Security is available in:

Title: *CICS RACF Security Guide*

Document Number: SC33-1701-32

Build Date: 06/23/00 12:52:23 Build Version: 1.3.0

Book Path: /home/publib/epubs/book/dfhjat53.boo

http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/dfhjat53/CCONTENTS

For information on the CICS VERIFY PASSWORD command, refer to the following:

http://publibz.boulder.ibm.com/cgi-bin/bookmgr_OS390/BOOKS/DFHJAP43/1.241

Example of CICS Surrogate UserID Creation and CONNX Login

1. Using standard RACF commands, create a new CICS user ID = CNXDEV01.
2. Define two profiles to RACF:

```
rdefine surrogat cnxdev01.dfhinstl uacc(none)
rdefine surrogat cnxdev01.dfhstart uacc(none)
```

3. Activate the SURROGAT class:

```
setropts classact(surrogat) raclist(surrogat)
```

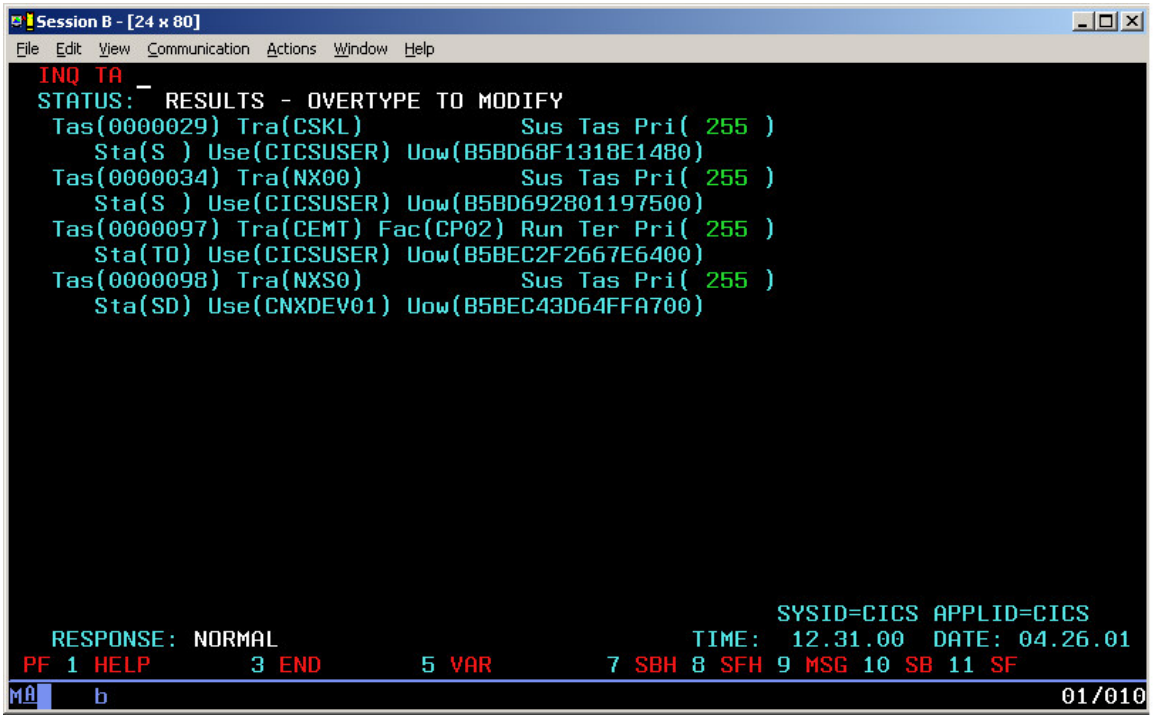
4. Execute RACF permit commands to authorize the default CICS USER ID = CICSUSER to the surrogate user profiles:

```
permit cnxdev01.dfhinstl class(surrogat) id(cicsuser)
permit cnxdev01.dfhstart class(surrogat) id(cicsuser)
```

5. Refresh the SURROGAT class:

```
setropts raclist(surrogat) refresh
```

6. Open the CONNX Data Dictionary Manager window. Select the **Import** button. Type the user ID **CNXDEV01** and a valid password, IP address and port.
7. Note that CONNX Listener Transaction NX00 was previously started by USERID = CICSUSER; Transaction NXS0 is started by transaction NX00 with USERID=CNXDEV01.



Note: *Refer to the following links for more information on CICS and RACF security:*

http://publibz.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/DFHJAT53/2.5

http://publibz.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/DFHJAT53/2.5.2

ODBC Driver Definition

On a Windows platform, the CONNX ODBC (Open Database Connectivity) Driver is a dynamic-link library administered by the Microsoft ODBC data source administrator. On a non-Windows platform (Linux, HP-UX, Solaris, AIX, etc.), the CONNX ODBC Driver is implemented as a shared library, and can be administered by any ODBC-compliant driver manager. Applications can access data located in a remote systems through the ODBC driver. The CONNX driver processes the ODBC function calls, submits requests to the appropriate data source, and then returns the results.

Coupled with the CONNX Data Dictionary, the ODBC driver provides a means of using many popular off-the-shelf querying tools and application development tools. The driver works with ODBC-compliant software, which increases its flexibility when used by companies with a range of front-end applications.

The ODBC interface enables applications to access data in database management systems using Structured Query Language (SQL) as a standard. (SQL is a widely accepted industry standard for defining data, manipulating data, data management, access protection, and transaction control. SQL uses tables, indexes, keys, rows, and columns to define storage locations.)

ODBC Driver Architecture

The architecture of the ODBC Driver has four components:

- **Application**
  Performs processing and calls ODBC functions to submit SQL statements and retrieve results.
- **Driver Manager**
  Loads drivers when requested by an application. Included with Microsoft Windows.
- **Driver**
  Software that processes the ODBC function calls, submits the SQL requests to a specific data source, and returns results to the application. If needed, the driver changes an application's request so that it conforms to the syntax supported by the associated database.
- **Data Source**
  The data the user wants to access and its associated operating system, DBMS, and the network platform (if any) used to access the DBMS.

CONNX ODBC Conformance

CONNX is an ODBC 3.51 Driver. The CONNX driver supports all core, level 1, and level 2 functions, including transaction functions, such as COMMIT and ROLLBACK. CONNX is also compatible with ODBC 3.x applications when accessed through the normal ODBC driver manager, and OLE DB/ADO 2.x applications.

Features of the ODBC Driver

Because the CONNX ODBC driver uses the CONNX Data Dictionary (CDD), there are functions that can be used to enhance performance.

The CDD utilizes keys and key segments that enable the ODBC driver to perform SQL optimization automatically. During a query, the driver uses the key information to access the requested data. This feature dramatically reduces the need for sequential reads of files.

The advanced optimization feature driver also enables the use of ranges during a keyed lookup, promoting faster retrieval of information.

The CDD and the ODBC driver also support the use of segmented keys and perform rapid keyed retrievals, requiring only part of a key. Segmented keys are an important element in many business applications. CONNX automatically utilizes a segmented key if the leftmost field in the key is included.

Related Topics

ODBC Driver Definition

ODBC Driver Architecture

CONNX ODBC Conformance

ODBC Programming Considerations

ODBC Driver

The CONNX ODBC driver can be used successfully with off-the-shelf products and by those who wish to write applications that call the driver.

If you are planning to write applications that call ODBC drivers, refer to the Microsoft Open Database Connectivity Software Development Kit (ODBC SDK), Programmer's Reference, for additional technical information about the ODBC driver manager, the ODBC function calls, and ODBC technology concepts. The kit, ISBN #1572315164, is published by Microsoft Press, which can be reached at 1-800-677-7377.

Record Locking

Record locking in RMS, IBM DB2, CICS/VSAM, DBMS, and Oracle database tables is handled automatically based on the type of SQL statement sent to the SQL server. UPDATE and DELETE statements lock the record at the start of the action, and immediately release the lock when complete. SELECT statements do not lock records, but can still read locked records in RMS using the RMS GET REGARDLESS flag.

During the record-locking procedure, a READLOCK is put on all transactions going forward. The application gets the latest copy of the record and prevents other users from modifying the record. It then issues an update and a COMMIT command, after which it retrieves the most recently saved record to make sure it matches.

Related Topics

- DB2
- DataFlex
- RMS
- Oracle Rdb
- DBMS
- Oracle
- VSAM

Linking Programs

The ODBC import library ODBC.LIB is used when linking programs with CONNX. The import library is included in the ODBC driver installation disk, which is provided in the Microsoft® ODBC SDK.

Configuring the ODBC Data Source Using a Provider String

Before an application program can communicate with the data source, configuration information must be provided. The data source comprises accessible data, its associated operating system, the database management system (or file system), and the network platform used to access the database management system.

Use the following provider string to configure the ODBC data source:

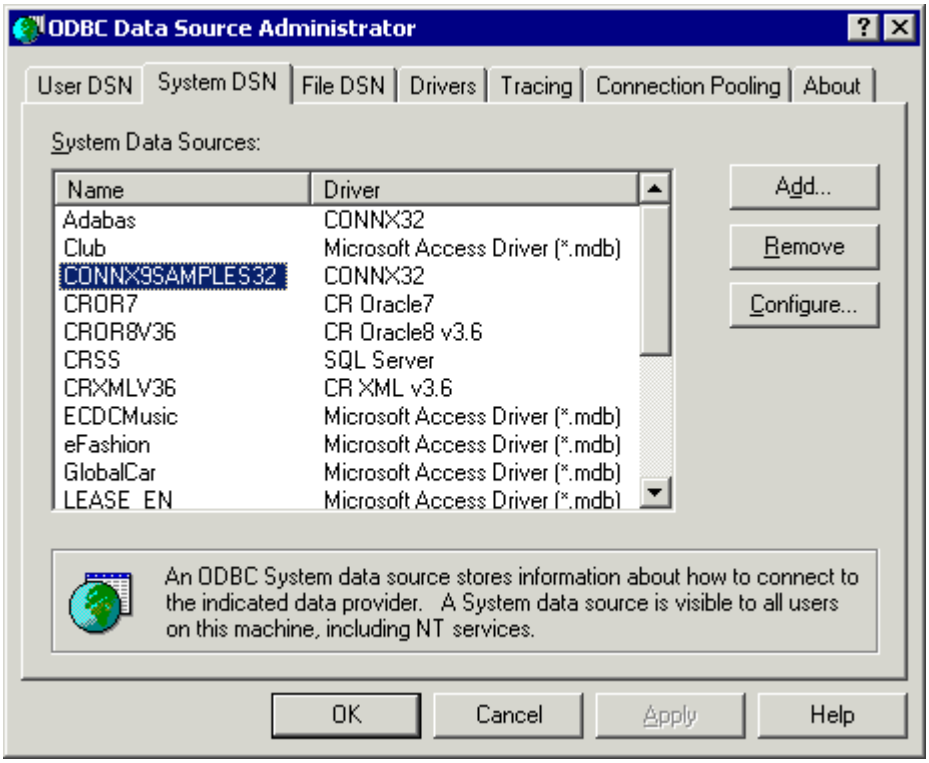
```
"driver=CONNX32;dd=C:\connx32\utils\CONNX.cdd;uid=theuser;pwd=thepass;
  "
```

Once the data source is configured, CONNX can be used with front-end applications (Microsoft Access, Microsoft Excel, Crystal Reports, and so on). Consult the user reference guide supplied with the front-end application to learn how to access ODBC data sources within the application or view the Quick Reference Cards included on your CONNX CD-ROM.

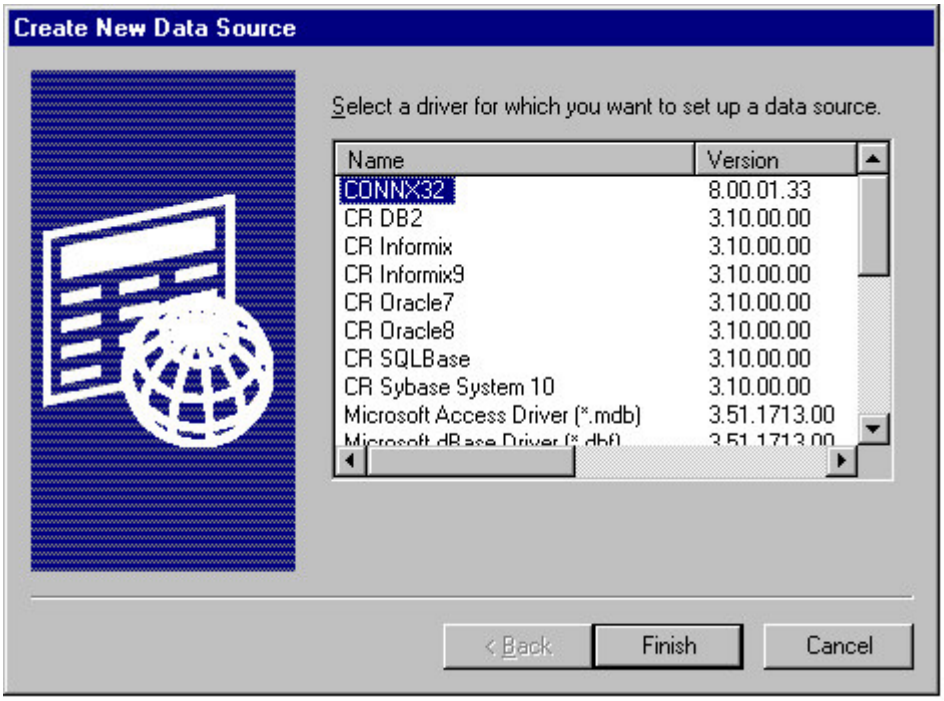
To configure the data source name for the CONNX ODBC driver

Note: The ODBC Data Source Administrator dialog box shows three different categories of data sources: user DSN, system DSN, and file DSN. User DSNs are available for the user currently logged onto the computer. System DSNs are available to all users of this computer. File DSNs are available to all users on a network if placed on a network drive.

1. Click the **Start** button, and then point to **Settings.** In **Control Panel,** click **Administrative Tools,** and then double-click the **Data Sources (ODBC)** icon.
2. The **ODBC Data Source Administrator** dialog box appears. Click the **Add** button to create a new data source.

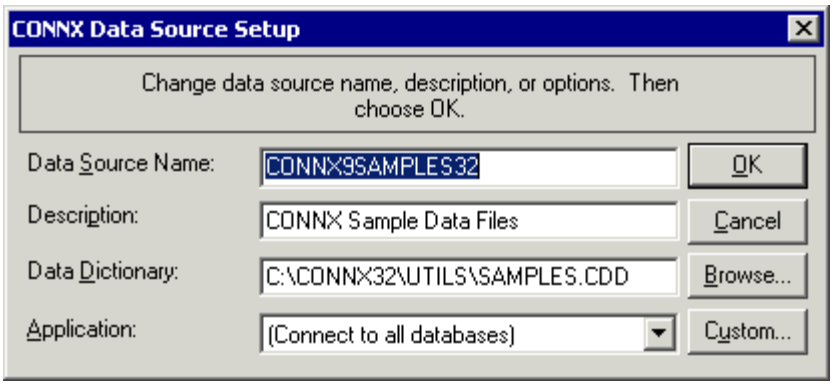


3. A list of installed drivers appears in the **Create New Data Source** dialog box. CONNX32 is the ODBC driver.



4. Double-click **CONNX32.**
5. The **CONNX Data Source Setup** dialog box appears. Type a data source name, a description, the name of the CONNX Data Dictionary where the data source or CDD is located, and the type of application to use. You may choose to complete the fields requiring the data source name and a description, and leave the other fields blank. Insert the required information when the application is run, or click the **Browse** button to confirm locations.
6. The **Data Source Name** is the name used to identify the data source to ODBC. This generally equates to a database. There are three types of data source names:

   User data source name: A data source name only relevant to the current user.
   Machine data source name: A data source name used only by the client computer.
   File data source name (recommended): A separate file placed on a network and available to multiple clients.
7. You must insert a comment in the **Description** field. This is a required field.
8. **Data Dictionary** is the full UNC (Universal Naming Convention) path to the CDD definition.
9. A CONNX **Application** name is defined to specify the databases required for this data source if using CONNX to access multiple databases. This is an optional field. An application name of OnDemand will delay the connection of a database contained within the CDD until a query is executed against that table within the database. OnDemand speeds up the initial connection and has the potential of reducing resource usage and connection times if a CDD contains multiple database connections.



10. Click the **OK** button. The driver writes the values to the ODBC.INI file, and they become the default values displayed when connecting to the data source. The data source is reconfigured to change these defaults.

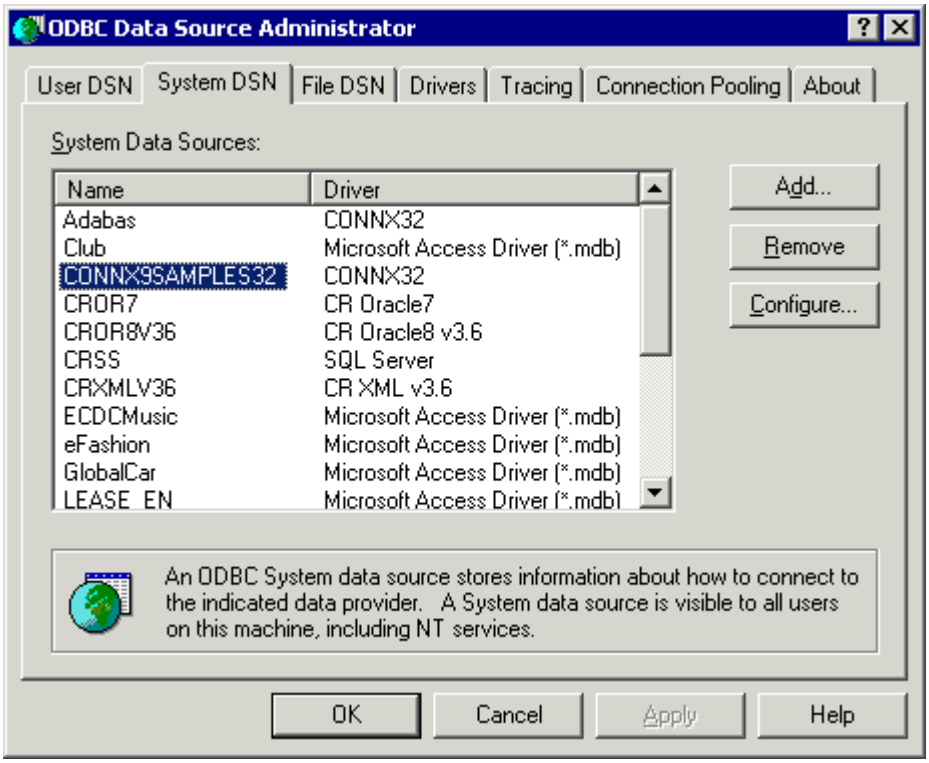
Note: *After exiting Control Panel, use the sample ODBC application shipped with this software or use other query tools to ensure the data source is functioning successfully.*

Related Topics

CONNX Security Overview

To configure an existing data source

1. Click the **Start** button, and then point to **Settings.** In **Control Panel,** click **Administrative Tools,** and then double-click the **Data Sources (ODBC)** icon.



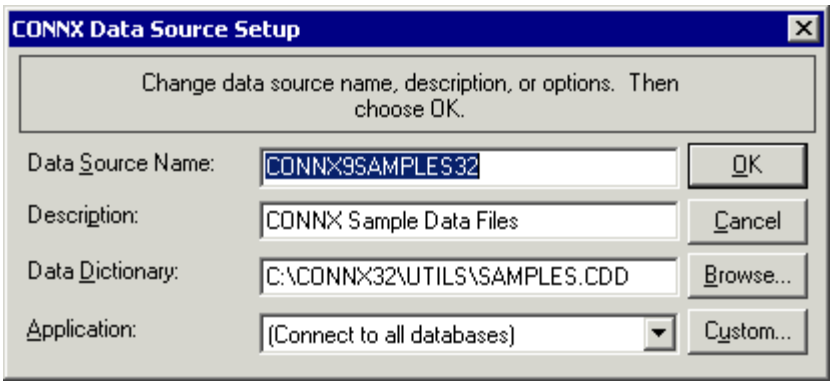
2. The **ODBC Data Source Administrator** dialog box appears. Select the data source to configure, and then click the **Configure** button.
3. The **CONNX Data Source Setup** dialog box appears. Type a data source name, a description, the name of the CONNX Data Dictionary where the data source or CDD is located, and the type of application to use. Complete the fields requiring the data source name and a description, and leave the other fields blank. Insert the required information when the application is run, or click the **Browse** button to confirm locations.

   The **Data Source Name** is the name used to identify the data source to ODBC. This generally equates to a database. There are three types of data source names:

   User data source name: A data source name only relevant to the current user.
   Machine data source name: A data source name used only by the client computer.
   File data source name (recommended): A separate file placed on a network and available to multiple clients.

   You must insert a comment in the **Description** field. This is a required field.

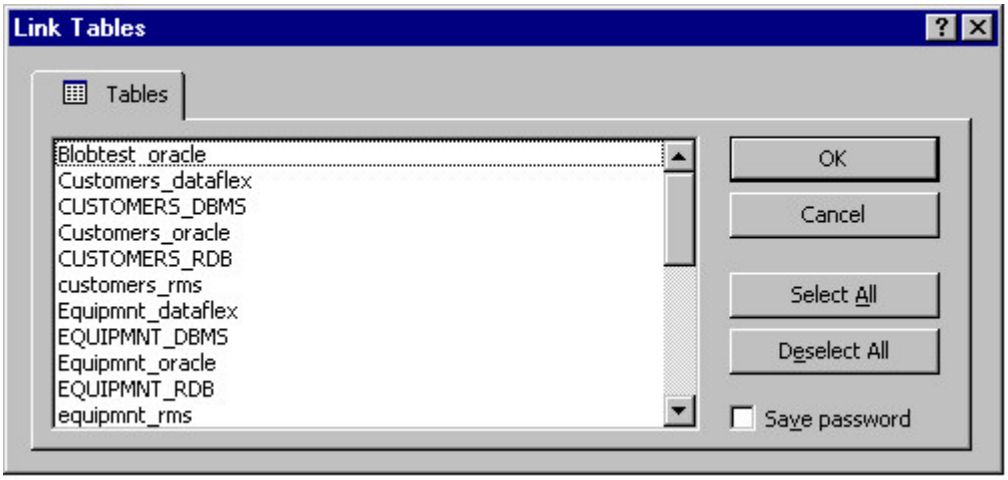
   Data Dictionary is the full UNC (Universal Naming Convention) path to the CDD definition.
4. A CONNX **Application** name is defined to specify the databases required for this data source if using CONNX to access multiple databases. This is an optional field.



5. Click the **OK** button. The driver writes the values to the ODBC.INI file, and they become the default values that are displayed when connecting to the data source. The data source must be reconfigured to change these defaults.

To link to a data source using Microsoft Access 2003

1. Start **Microsoft Access.**
2. Select the **Blank Database** option under **New,** and then click the **Create** button.
3. On the **File** menu, point to **Get External Data,** and then click **Link Tables.**
4. Select **ODBC Databases ()** from the **Files of Type** list box in the **Link** dialog box.
5. Select a CONNX data source in the **Select Data Source** dialog box, and then click the **New** button. The **Create New Data Source** dialog box appears. Enter the name of the new data source, and then click **OK.**
6. The **Link Tables** dialog box appears.



7. Select the tables or views to use with Microsoft Access, and then click the **OK** button.

To open a connection to the CONNX ODBC Driver

A connection string must be used with a connection object to open a connection to a server. A recordset requires a SQL query string and an open connection object to retrieve a recordset from a server. See CONNX ODBC Connection String Parameters for a complete list of the available CONNX connection parameters.

The ADO technology available in Microsoft Visual Basic can be used to open connections within CONNX. Use the following instructions to activate ADO in Visual Basic:

1. Select **Project** on the **File** menu.
2. Select the check boxes for **Microsoft ActiveX Data Objects Library** and **Microsoft ActiveX Data Objects Recordset Library,** and then click the **OK** button.
3. Enter the following code snippet:

```
Public Sub ConnectToDb()
Dim con as new ADODB.Connection
Dim rec as new ADODB.Recordset
Dim strConnection as string
Dim strSQLQuery as string

strConnection = "DSN=MYDSN;UID=MYUSER;PWD=MYPASSWORD"
strSQLQuery = "SELECT * FROM MYTABLE"


con.Open strConnection
rec.Open strSQLQuery, con

   End Sub
```

where MYDSN is a DSN created on your computer, MYUSER and MYPASSWORD are valid CONNX user names and passwords. MYTABLE is any table located in a CONNX Data Dictionary (CDD).

CONNX ODBC Connection String Parameters

This table lists the CONNX ODBC connection string parameters that can be used when opening a connection to the CONNX ODBC driver.

Connection Parameter Name	Description
Application	Specifies the CONNX application to use when connecting. The application can be a database within the CDD or within an entire collection of databases.
ClientID	This is a user-defined identifier sent from the CONNX Client to the CONNX Server. It can be used to run custom procedures on the server during a connection.
DBKey	This is an Rdb-only feature. The only valid option is ATTACH. If DBKEY=ATTACH is specified, then Rdb DBKEYS is not reused throughout the lifetime of a connection.
DBREADONLY	A list of databases in the CDD, separated by commas, which are treated as read only.
DBREADWRITE	A list of databases in the CDD, separate by commas, which are treated as read/write.
DSN	The User or System DSN name to use for connecting. The correct DD, FILEDSN, or DSN must be specified.
DD	A full path to the name of the CONNX Data Dictionary to use for connecting. This can be used as a replacement for DSN or FILEDSN. This option enables "DSN-less" connections. The correct DD, FILEDSN, or DSN must be specified.
Exclusive	Connect to all databases in "exclusive" mode.
FILEDSN	The name of the file data source to use during the connection. The correct DD, FILEDSN, or DSN must be specified.
Node	The name of the server to connect to. This option is only useful if connecting to a CDD that only has one data source in it.
OEM	Key used for OEM redistributors.
Prompt	Options are "YES" or "NO". Determines whether CONNX displays a logon dialog if sufficient connection information is not specified.
PWD	The CONNX password.
READONLY	This option connects to all databases in the CDD in read only mode.
RPC	The name of a CONNX RPC to execute right after a connection has been established.
UID	The CONNX UserID.
ADADBIDLIST	Allows dynamic rebinding of adabas DBIDs in the CONNX data dictionary. This is useful if you have multiple Adabas databases with identical structures, but different data. At connection time, you can redirect one or more databases in the CDD to different adabas DBIDs. It is the responsibility of the user to ensure that the structure is the same between the database, otherwise error will occur during query processing.

	ADADBIDLIST=(<logicalCDDDatabaseName>:<new DBID>,<logicalCDDDatabaseName>:<new DBID>,…) Example: ADADBIDLIST=(adatest:2,adalocal:1) Where adatest and adalocal are database cylinder names in the CDD.

OLE DB Provider Definition

The CONNX OLE DB Provider is a dynamic-link library that applications can call to access data located in remote systems. Coupled with the CONNX Data Dictionary, the driver works with OLE DB-compliant applications, which increase its flexibility when used by companies with a range of front-end applications.

The OLE DB interface enables applications to access data in database management systems using Structured Query Language (SQL) as a standard. (SQL is a widely accepted industry standard for defining data, manipulating data, data management, access protection, and transaction control. SQL uses tables, indexes, keys, rows, and columns to define storage locations.)

CONNX OLE DB Conformance

CONNX is a Level 2.5 provider. The CONNX driver supports all core functions, such as COMMIT and ROLLBACK. CONNX is also compatible with OLE DB/ADO 2.x applications.

Configuring the OLE DB Data Source

Before an application program can communicate with the data source, configuration information must be provided.

The data source configuration comprises accessible data and its associated operating system.

Use the following provider string to configure the OLE DB data source:

```
"Provider=CONNXOLEDB;Persist Security Info=True;prompt=NoPrompt;User
   ID=theuser;Password=thepass;Data
   Source=C:\connx32\utils\CONNX.cdd;Mode=ReadWrite"
```

Features of the OLE DB Provider

Because the CONNX OLE DB Provider uses the CONNX Data Dictionary (CDD), there are functions that can be used to enhance performance.

The CDD utilizes keys and key segments that enable the OLE DC Provider to perform SQL optimization automatically. During a query, the driver uses the key information to access the requested data. This feature dramatically reduces the need for sequential reads of files.

The advanced optimization feature driver also enables the use of ranges during a keyed lookup, promoting faster retrieval of information.

The CDD and the OLE DB Provider also support the use of segmented keys and perform rapid keyed retrievals, requiring only part of a key. Segmented keys are an important element in many business applications. CONNX automatically utilizes a segmented key if the leftmost field in the key is included.

The CONNX OLE DB Provider supports Unicode and ANSI data types.

To create an OLE DB Provider connection object

Use Visual Basic, C#, or another OLE DB-compliant resource to create a connection object.The following code exemplifies creating a connection in Visual Basic:

```
Dim cnn As New ADODB.Connection
Dim rs As New ADODB.Recordset

cnn.Open  "Provider=CONNXOLEDB;Persist Security
   Info=True;prompt=NoPrompt;User ID=theuser;Password=thepass;Data
   Source=C:\connx32\utils\CONNX.cdd;Mode=ReadWrite"

SQL = "SELECT orderid, customerid, productid, orderdate,
   productquantity FROM orders_rms"

rs.Open SQL, cnn
```

or in .NET:

```
Dim cnn As OleDbConnection
Dim cmd As OleDbCommand
Dim sqlda As OleDbDataAdapter
Dim sqlds As DataSet

cnn = New OleDbConnection("Provider=CONNXOLEDB;Persist Security
   Info=True;prompt=NoPrompt;User ID=theuser;Password=thepass;Data
   Source=C:\connx32\utils\CONNX.cdd;Mode=ReadWrite")

SQL = "SELECT orderid, customerid, productid, orderdate,
   productquantity FROM orders_rms"

cmd  = New OleDbCommand(SQL, cnn)
sqlda = New OleDbDataAdapter(cmd)
sqlds = New DataSet
sqlda.Fill(sqlds)
```

Chapter 6 - CONNX JDBC Driver

JDBC Driver Definition

JDBC is Sun Microsystem's JavaSoft application programming interface (API) standard for connecting to databases that support Structured Query Language (SQL). JDBC is patterned as a package of object-oriented objects that include Connect, ResultSet, and Statement. Each object contains various API methods, for example, Connect(), DisConnect(), and PrepareSQL().

The CONNX JDBC driver implements the JDBC specification developed for use within CONNX to enable connectivity to all types of databases. Used with the CONNX Data Dictionary (CDD), the JDBC driver provides a means of using many popular querying tools and application development tools. The CONNX JDBC driver works with JDBC-compliant software, which increases its flexibility when used by companies with a wide range of front-end applications and database types. The CONNX JDBC interface enables applications to access data in database management systems using the JavaSoft JDBC API to connect to the databases.

Related Topics

JDBC Driver Architecture

CONNX JDBC Driver Architecture

The CONNXJDBC.LOG file

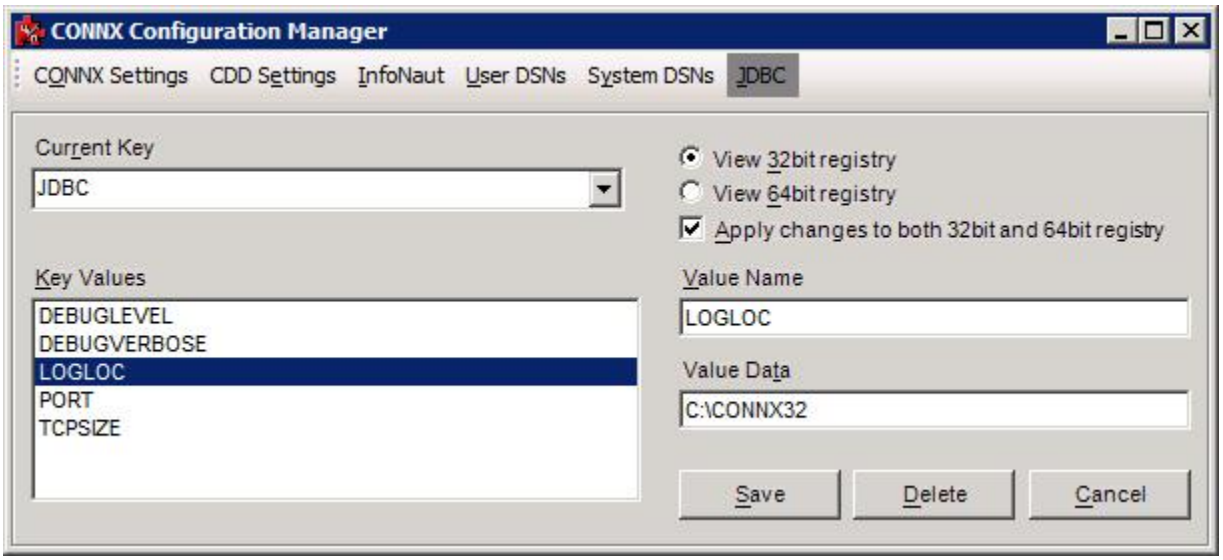
CONNX JDBC logging activity is turned on by default as some server functions are located in a file called CONNXJDBC.LOG. The file resides in C:\CONNX32\. It can be changed through the Windows registry settings.

The CONNXJDBC.LOG file records the following server functions:

- Client Logon
- Client Logoff
- Exception Error Messages

CONNX JDBC Configuration Settings

The Configuration Manager can adjust some of the CONNX JDBC configuration settings to increase the flexibility of the CONNX JDBC Server.



NOTE: The above image shows the 32bit setting. To view the 64bit JDBC configuration settings, select the "View 64bit registry". Since it is possible to run both a 32bit and 64bit JDBC server at the same time, it is recommended that you uncheck the "Apply changes to both 32bit and 64bit registry" checkbox when making any changes. Otherwise, a conflict will occur between the two JDBC servers.

NOTE2: On 32bit Windows operating systems the "View 32bit registry/View 64bit registry" options will not be available.

The following is a list of parameters that can be adjusted by using the Configuration Manager. The parameters are located at HKEY_LOCAL_MACHINE\SOFTWARE\CONNX\JDBC.

Registry Setting	Value	Note
DebugLevel	0, 1, 2, 3	Higher value brings additional debug messages. Default is 0.
DebugVerbose	0, 1	Verbose level for debug messages. Default is 0.
LogLoc	C:\CONNX32\ (default for 32bit JDBC server) C:\Program Files\CONNX (default for 64bit JDBC server)	Location of .log file.
Port	7500 (default for 32bit	Port used by the

	JDBC server) 7502 (default for 64bit JDBC server)	server.
TCPIPSize	8192	Size of TCP/IP packet.

CONNX JDBC Router

The CONNX JDBC Router component is only necessary if Web applications are served by a Web server located on a different machine than the JDBC Server. The router is necessary because applet security states that a socket connection must be made only to the Web server machine.

The router is a Java application placed on the Web server. It is stored in the connxRouter.jar file which is located in C:\CONNX32\CONNXJDBC\java\jar\connxRouter.jar, and in the redistributable Java jar file, C:\CONNX32\CONNXJDBC\java\ftp\CONNXjdbcftp.jar.

The parameters for the router are as follows:

IP# of CONNX JDBC Server

Port# of CONNX JDBC Server

Port# of CONNX JDBC Router

The router can be invoked as a Java application with the following command, typed within your Java program code:

```
Java -classpath <location and name of CONNXRouter.jar>
com.CONNX.Router.TCRouter <machine name of CONNX JDBC Server or
machine ip address or alias> <PORT# of CONNX JDBC Server> <Port# of
Router>
```

Example:

```
Java -classpath C:\CONNX32\CONNXJDBC\java\jar\connxRouter.jar
com.CONNX.Router.TCRouter MyServerMachine127.00.00.01 7500 7503
```

The Java client applications must connect to the Java Router, and address the fact that the Port number may differ from the default.

Note: It is recommended that the JDBC Server run on the same machine as the Web server. The JDBC Router is only used in environments where this is not possible.

Related Topics

CONNX JDBC Java Applet Architecture with Router

JDBC Data Type Conversions

The following tables display the data conversions that are acceptable for both retrieval and update in CONNX JDBC.

getxxx Methods to Update Data											
Available Data Type Conversions											
Methods	**Binary**	**Bit**	**Char**	**Double**	**Integer**	**Long Varbinary**	**Long Varchar**	**Numeric**	**Real**	**Smallint**	**Timestamp**
getAsciStream	X		X			X	X				
getBigDecimal		X	X	X	X		X	X	X	X	
getBinaryStream						X					
getBoolean		X	X	X	X		X	X	X	X	
getByte	X	X	X	X	X		X	X	X	X	
getBytes	X		X		X		X				
getDate											X
getDouble		X	X	X	X		X	X	X	X	
getFloat		X	X	X	X		X	X	X	X	
getInt		X	X	X	X		X	X	X	X	
getLong		X	X	X	X		X	X	X	X	
getObject	X	X	X	X	X	X	X	X	X	X	
getShort		X	X	X	X		X	X	X	X	
getString		X	X	X	X		X	X	X	x	X
getTime					X						X
getTimestamp					X						x
getUnicodeStream	x	x	X			X	X				

setxxx Methods to Update Data											
Available Data Type Conversions											
Methods	**Binary**	**Bit**	**Char**	**Double**	**Integer**	**Long Varbinary**	**Long Varchar**	**Numeric**	**Real**	**Smallint**	**Timestamp**
setAsciStream	X		X			X	X				
setBigDecimal		X	X	X	X		X	X	X	X	
setBinaryStream						X					
setBoolean		X	X	X	X		X	X	X	X	
setByte	X	X	X	X	X		X	X	X	X	
setBytes	X		X		X		X				
setDate											X
setDouble		X	X	X	X		X	X	X	X	
setFloat		X	X	X	X		X	X	X	X	
setInt		X	X	X	X		X	X	X	X	

setLong		X	X	X	X		X	X	X	X	
setObject	X	X	X	X	X	X	X	X	X	X	
setShort		X	X	X	X		X	X	X	X	
setString		X	X	X	X		X	X	X	x	X
setTime					X						X
setTimestamp					X						x
setUnicodeStream	x	x	X			X	X				

CONNX JDBC Architecture

JDBC Driver Architecture

The four types of JDBC drivers are as follows:

- **Type 1 - ODBC Bridge**
  Used with databases that are unable to directly support JDBC. ODBC is a Microsoft Windows interface to SQL. This solution works best on a Windows-based system, but may not work on other operating systems that do not support ODBC.
- **Type 2 - Native API, Native Code**
  The fastest JDBC driver, written partly in Java and partly in native code, for example, Microsoft C++ . Although it speaks the native protocol of the SQL database, it is limited to operating systems on which native code is preinstalled on client machines.
- **Type 3 - Net Protocol, Pure Java**
  Written entirely in Java. This type of driver can run on any platform or browser that supports Java. The driver converts requests into a database that uses vendor-neutral protocol. A server process receives the requests and carries out the specified action on the database. With this type of driver, you can access SQL databases on different client machines without loading additional JDBC drivers. This JDBC driver is small and loads quickly.
- **Type 4 - Native Protocol, Pure Java**
  The fastest way to use the Web from a single server. This type of JDBC driver is written entirely in Java, which means it can be safely loaded into any Java-powered Web browser. The driver speaks DBMS-vendor-specific protocol directly to the SQL server. It is efficient, but if you need to attach to various types of SQL databases, several JDBC drivers must be loaded onto the client machines.

CONNX JDBC is a Type 3 driver, which means it can be run on any platform that supports Java. It is a pure Java implementation, designed to take advantage of the CONNX architecture.

Related Topics

JDBC Driver Definition

CONNX JDBC Driver Architecture

CONNX JDBC Driver Architecture

The architecture of the CONNX JDBC Driver has five components.

- **CONNX JDBC (Thin Client) Driver**
  The CONNX JDBC Driver is located in a Java .jar file named connxjdbc.jar. The JDBC driver is a thin JDBC driver that communicates to the CONNX JDBC Server through the use of a socket. JDBC calls are translated to socket requests to which the CONNX JDBC Server responds. The .jar file must be copied to Java Virtual Machine (JVM) platforms other than Windows.

- **CONNX JDBC Server**
  The CONNX JDBC Server component communicates with the CONNX JDBC Driver. It resides on the machine on which CONNX is installed. It is a Windows executable that opens a socket and listens for new connections. When it accepts a new connection, it creates a new thread that is dedicated to communicating to that client. Installing the CONNX JDBC Server component on every machine is an optional task since only one server is required for communication with all CONNX JDBC client machines.

When the CONNX JDBC Server component is installed with the CONNX client component on a non-Windows platform (Linux, HP-UX, AIX, Solaris, etc.), run the CONNX JDBC Server as a daemon process.

- **CONNX JDBC Router**
  The CONNX JDBC Router is a Java application placed on the non-Windows Web server. It is stored in the ConnxRouter.jar file which is located in the redistributable Java jar file, C:\CONNX32\CONNXJDBC\JAVA\ftp\connxjdbcftp.jar

  The CONNX JDBC Router must be run on the machine that hosts a Web server but does not have CONNX installed. If JDBC applets are used, the following environments must include a CONNX JDBC Router:

  Non-Windows Web servers (Linux, Solaris, HP-UX)
  Windows Web servers running on a system on which CONNX is not installed.

- **CONNX DSN Registry Tool**
  The CONNX DSN Registry tool enables JDBC connections to locate and refer to the logical name of the data source. It must be installed on the same machine as the CONNX JDBC Server.

- **CONNX Engine**
  This component is the CONNX ODBC Driver, responsible for interacting with data sources.

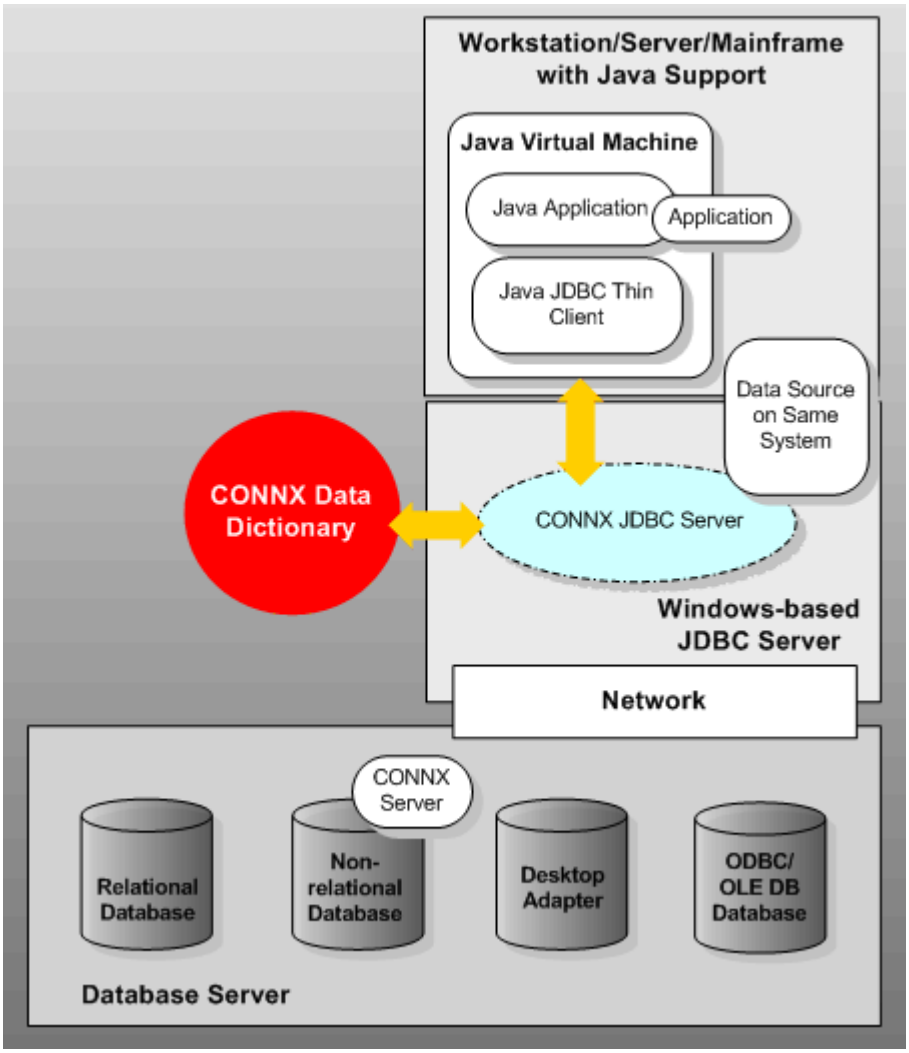
Related Topics

- Registering the Data Source Name
- Opening a connection to the CONNX JDBC Driver
- CONNX JDBC Server Definition
- CONNX JDBC Router

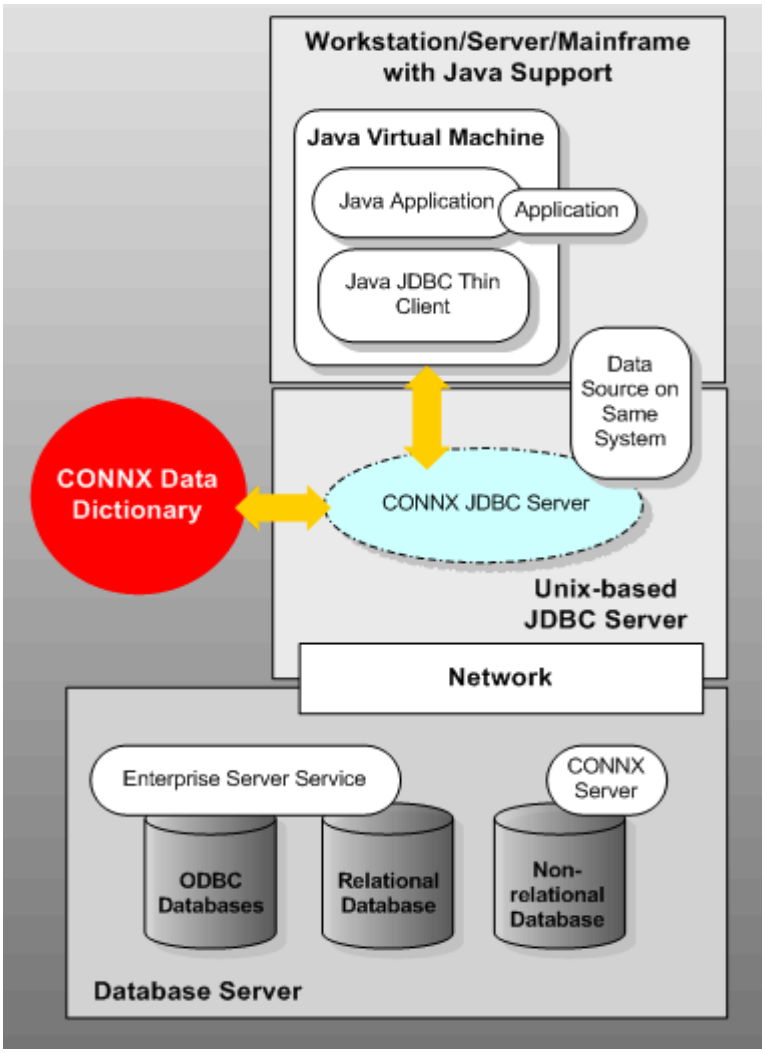
CONNX JDBC Java Application Architecture

Java applications, stand-alone programs that require the installation of a Java Virtual Machine (JVM), are usually invoked with a Java command. They also require that the CONNX JDBC .jar file (Thin Client) be stored on the Classpath for that machine.

JDBC Architecture - Windows version

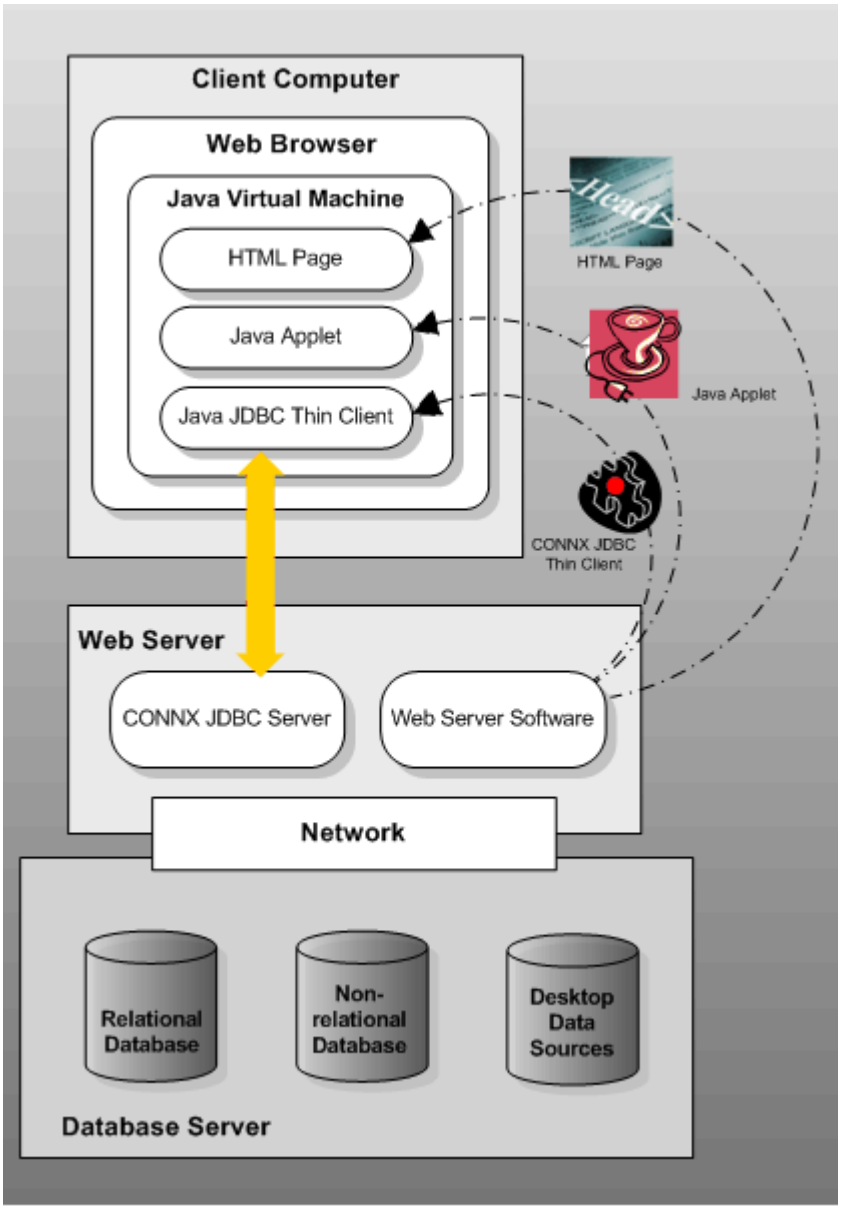


JDBC Architecture - Unix version



CONNX JDBC Java Applet Architecture

Java applets are usually invoked from within an HTML page. The HTML page is downloaded along with the Java applet, which includes all code necessary to process the applet, including the CONNX JDBC .jar file (Thin Client).

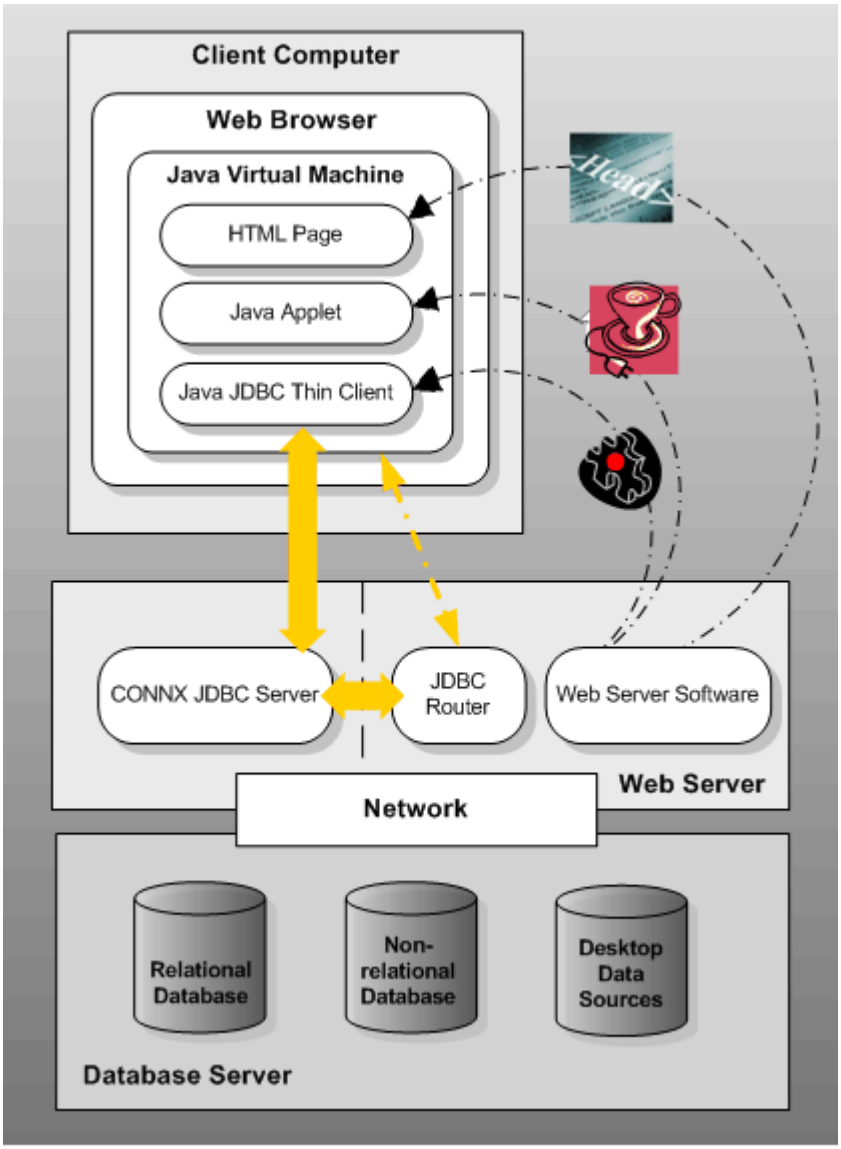


CONNX JDBC Java Applet Architecture with Router

Because the CONNX JDBC Server must be installed on Microsoft Windows, a CONNX JDBC Router is required in instances where the Web server is not in the same location as the CONNX JDBC Server.

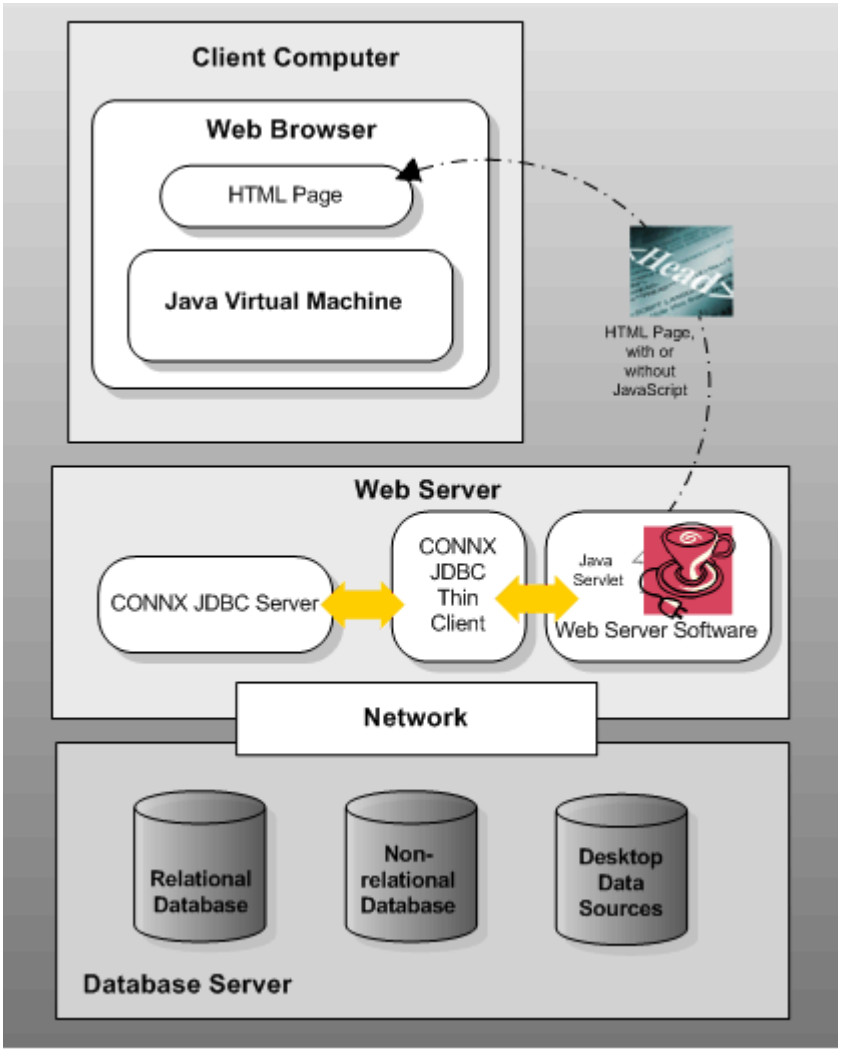
Example: Because the CONNX JDBC Server must be in a Windows environment, an Apache Web server running on Linux cannot be on the same machine as the CONNX JDBC Server. Install the CONNX JDBC Router, which routes calls going to the CONNX JDBC Server.

If the applet is trusted, this extra step is not necessary. If the applet cannot be trusted, the Java client does not create new code for the router, because the router appears as a CONNX JDBC Server to the client. Applet security measures force a proxy server to be used in order to ensure socket connections to a computer other than the Web server.

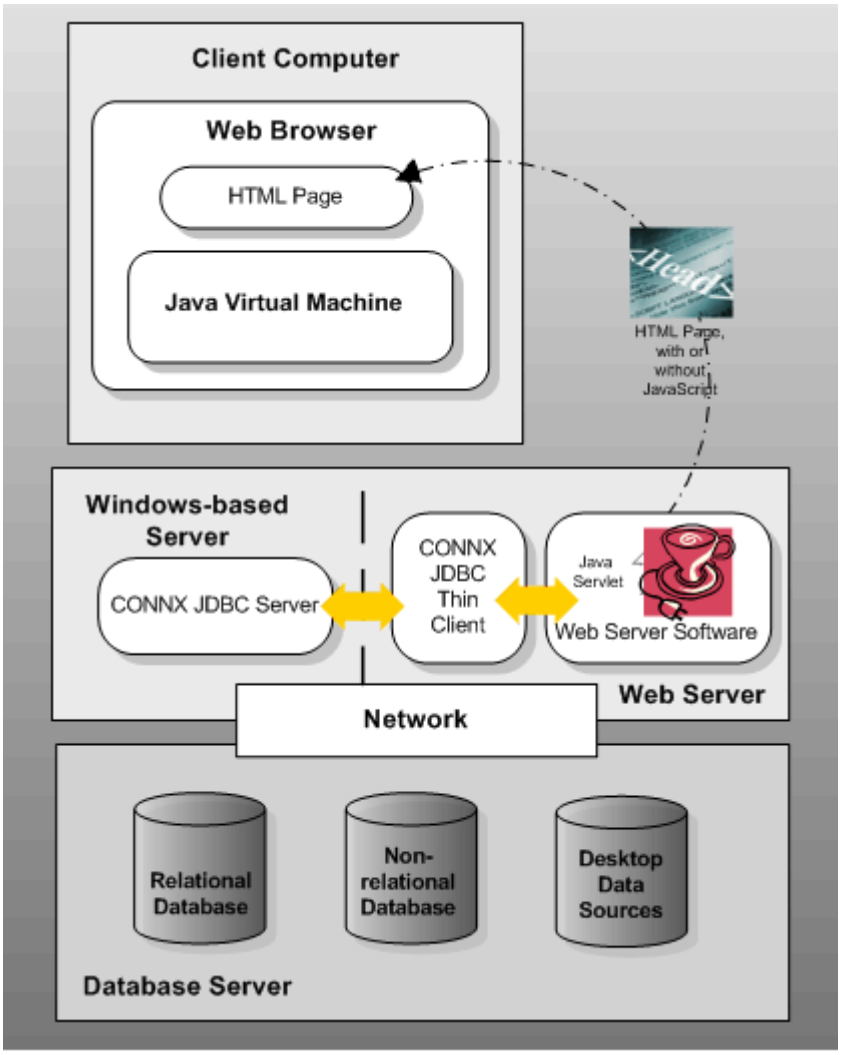


CONNX JDBC Java Servlet Architecture

A Java servlet is Java code executed by the host Web server. It is a non-interactive application that is referenced in a server script, for example, Java Server Pages (JSP). A CONNX JDBC Router is not needed in this scenario, as the router is not bound by browser security.



CONNX JDBC Java Servlet Architecture running on a non-Windows Web server connecting with Windows-based servers


Client Computer
Web Browser
HTML Page
Java Virtual Machine
HTML Page, with or without JavaScript
Windows-based Server
CONNX JDBC Server
CONNX JDBC Thin Client
Java Servlet
Web Server Software
Web Server
Network
Relational Database
Non-relational Database
Desktop Data Sources
Database Server

Connecting to a JDBC Data Source

Connecting to a Data Source

JDBC is a specification created by Sun Microsystems, and supported by many third-party tools. Its Object-Oriented Objects -- Connection, ResultSet, Statement, DatabaseMetaData -- represent connecting to a database, getting results back from a query, executing a statement, and getting metadata from a database.
There are generally four steps that a Java developer must follow in order to connect to CONNX JDBC:

1. Register the Data Source
2. Set ClassPath
3. Load the CONNX JDBC Driver
4. Open a connection to the CONNX JDBC Driver (Thin Client)

Before beginning any of the above procedures, you must have installed a JDK (Java Development Kit) that is compatible with your system. Refer to "CONNX JDBC" in the online CONNX Installation Guide for a list of Web sites that offer downloadable JDKs.

Related Topics

 Registering the Data Source Name

 Setting the Classpath

 To load and register the CONNX JDBC Driver

 Opening a connection to the CONNX JDBC Driver

Registering the Data Source Name

To use CONNX JDBC, you must first use the DSNRegistry tool to create a logical name, known as a Data Source Name (DSN) that points to the CDD.

The CDD DSN should be registered on the same machine on which the CONNX JDBC Server is installed. The CONNX JDBC Server checks the Windows registry on that machine to locate the DSNs.

Note: A JDBC DSN is not the same as an ODBC DSN. If you intend to use both ODBC and JDBC to access databases, you must create a DSN for both access methods.

Related Topics

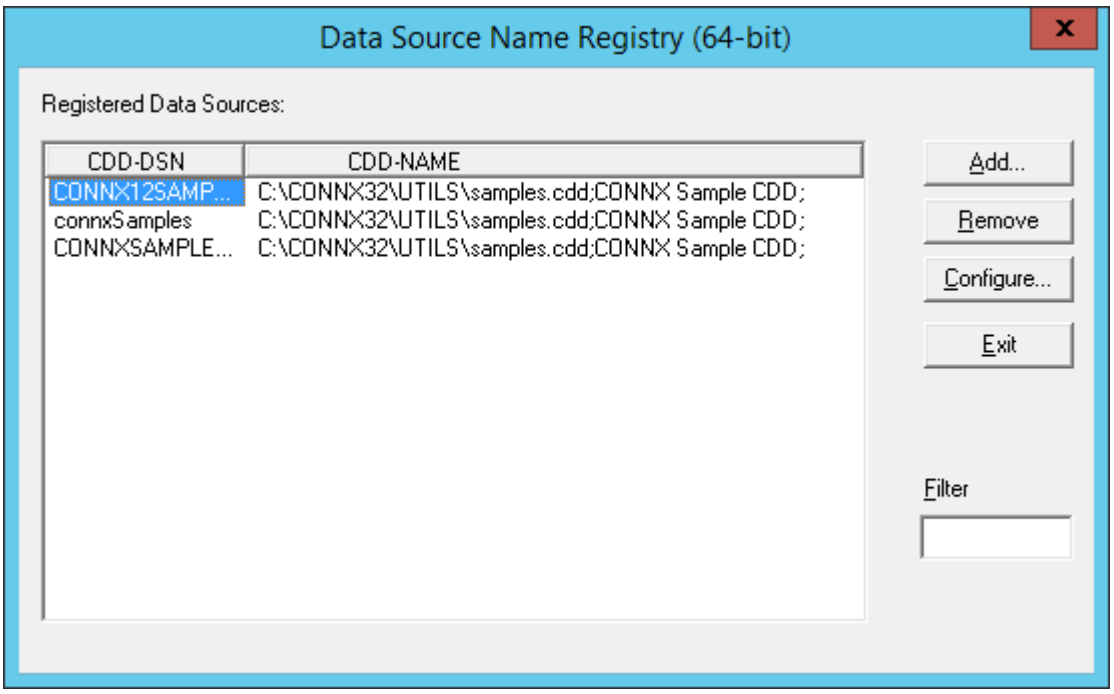
CONNX JDBC Server Definition

Connecting to a Data Source

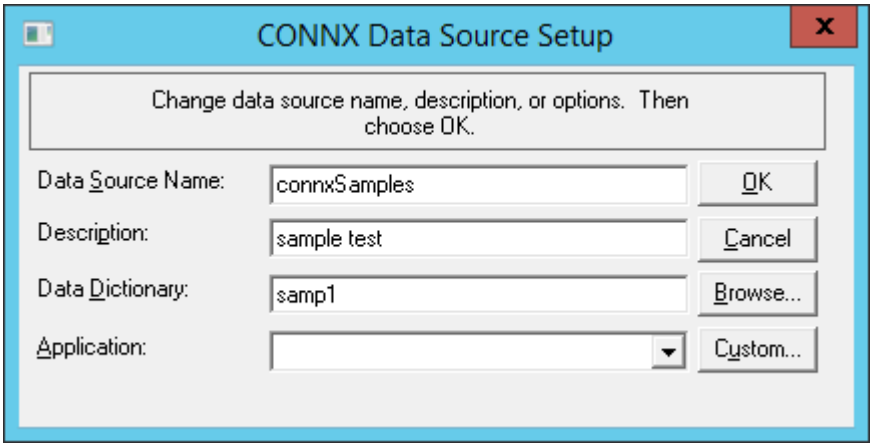
To add a new data source name for the JDBC Driver

To add a new data source name for the JDBC Driver

1. Click the **Start** button, and then point to **All Programs**. Point to **CONNX Driver** then click **DSNRegistry.** The Data Source Name Registry window appears.



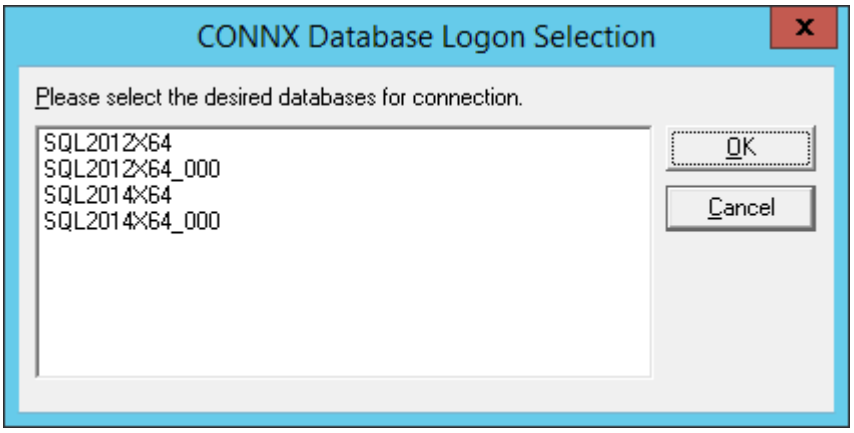
2. Click the **Add** button. The **CDD Data Source Setup** dialog box appears.



Note: *The Samples CDD DSN for ODBC is created automatically when the CONNX JDBC component is installed*.

3. Type a 4- to 32-character logical name for the CDD in the **CDD DSN** text box.
4. Type an optional description of the contents of the CDD in the **Description** text box.
5. Type the absolute path to the location of the CDD files in the **CDD Name** text box.
6. Enter the name of the application used in the CDD, for example, RMS, Oracle, or DBMS, from the **Application** list box.

7. The custom button becomes active when the (Connect to selected databases) is selected from the Application combo box. When you press the Custom button, a list of the databases in the CDD appears, allowing you to select one or all of the databases for connection.



3. Click the **OK** button to return to the **Data Source Name Registry** dialog box.
4. Click the **Exit** button.

Setting the Classpath

Java Virtual Machines rely on an environment variable called Classpath to point to the necessary Java components (classes) required for execution. When the CONNX JDBC Driver is used, the Classpath environment variable must be extended to enable the CONNX JDBC Driver (Thin Client). The CONNX JDBC Driver is referred to by its file name:

C:\CONNX32\CONNXJDBC\java\jar\connxjdbc.jar

An example of a modified Classpath is as follows:

```
CLASSPATH=C:\CONNX32\CONNXJDBC\java\jar\connxjdbc.jar
```

To set the system environment variable, use the following procedure:

1. Right-click on **My Computer,** and then select **Properties.**
2. Select the **Advanced** tab, and then click the **Environment Variables** button.
3. Under **System Variables,** click the **New** button.
4. Enter the new **Variable Name (Classpath),** and the **Variable Value (C:\CONNX32\CONNXJDBC\java\jar\connxjdbc.jar),** and then click the **OK** button.
5. Click the **OK** button to return to the **System Properties** window.
6. Click the **OK** button.

Related Topics

- Connecting to a Data Source
- Registering the Data Source Name
- To add a new Data Source Name for the JDBC Driver

Loading the JDBC Driver

To load and register the CONNX JDBC Driver

Enter the following JDBC call within the Java program:

```
Class.forName("com.Connx.jdbc.
TCJdbc.TCJdbcDriver").newInstance();
```

Note: *The variable newInstance is inserted to ensure that the Class is registered. Some browsers have had difficulty loading Class without newInstance included as part of the syntax.*

Related Topics

Opening a connection to the CONNX JDBC Driver

Setting the Classpath

Opening a connection to the CONNX JDBC Driver

The JDBC method getConnection(xxx) is used to make a connection to the JDBC data source. Several different getConnection() methods exist in JDBC. The URL remains constant in both of the methods, as this is where the driver- specific information is placed. The URL is concatenated with the following attributes:

- **Driver Identifier**
  For CONNX JDBC, the format is as follows:

  jdbc:connx:

- **Database Identifier**
  For CONNX JDBC, this represents the symbolic name for the CDD registered via the DSNRegistry tool. The format is as follows:

  DD=Symbolic Name for Data Source

- **TCP/IP Address of Server**
  This is the TCP/IP address or host name of the server (or the CONNX Router if a CONNX Router is required by your configuration). The format is as follows:

  GATEWAY=TCPIP address or hostname of Server/Router

- **PORT (optional argument)**
  The CONNX JDBC server is bound to a specific port. The default is 7500, but the server can be invoked with a different port. The client and server must be in sync; if the server has a different port number, then the URL must represent this by including the PORT option. The format is as follows :

  PORT=7510

- **SSL (optional argument)**
  If the JDBC server is configured for SSL connections, then the java client must specify "ssl=true" in the connection string. The format is as follows:

  SSL=true

  The example below demonstrates a getConnection call for CONNX based on the following criteria:

  - Server named Sparky
  - JDBC Data Source called Payroll

```
connectionObj = DriverManager.getConnection
("jdbc:connx:DD=Payroll;Gateway=Sparky",
username, password);
```

- **ADADBIDLIST (optional argument)**
  Allows dynamic rebinding of adabas DBIDs in the CONNX data dictionary. This is useful if you have multiple Adabas databases with identical structures, but different data. At connection time, you can redirect one or more databases in the CDD to different adabas DBIDs. It is the responsibility of the user to ensure that the structure is the same between the database, otherwise error will occur during query processing.

  ADADBIDLIST=(<logicalCDDDatabaseName>:<new DBID>,<logicalCDDDatabaseName>:<new DBID>,…)

  Example: String connString = "jdbc:connx:DSN=demo4adabas;GATEWAY=localhost;port=7502;USER=xxx;PASSWORD=x

xx;ADADBIDLIST=(adatest:2,adalocal:1)";

Where adatest and adalocal are database cylinder names in the CDD for the DSN demo4adabas.

Additionally - support for multiple JDBC Servers and/or Data servers has been added to the CDD server name fields and the JDBC connection string.

<servername>[:<port>][!<servername>[:<port>]][!<servername>[:<port>]]...

Note: Each instance of <servername>[:<port>] is spearated by an exclaimation (!) mark.

For example - if you have two JDBC servers set up for redundancy - one on a server called PRIMARY, listening on port 7323
and another server called SECONDARY, listening on port 8754, the gateway server name would be:

PRIMARY:7323!SECONDARY:8754

When a list of servers is provided - CONNX will initially pick one at random. If the connection fails for that server, it will then serially loop through
all the servers from that point forward, wrapping back around to the beginning if necessary, until all options are exhausted, or a successful connection is established.

Type the above criteria on the command line in your system or within the Java program code.

Related Topics

To load and register the CONNX JDBC Driver

CONNX JDBC Connection String Parameters

This table lists the CONNX JDBC connection string parameters that can be used when opening a connection to the CONNX JDBC Driver.

Connection Parameter Name	Description
APPLICATION	Specifies the CONNX application to use when connecting. The application can be a database within the CDD or within an entire collection of databases.
CLIENTID	This is a user-defined identifier sent from the CONNX Client to the CONNX Server. It can be used to run custom procedures on the server during a connection. Use of this setting will affection connection pooling, as any identifier used here is part of the unique signature of a connection.
CNXSESSIONID	This is a user-defined identifier sent from the CONNX Client to the CONNX Server. It can be used to run custom procedures on the server during a connection. Use of this setting will not affect connection pooling. A special CNXSessionID of "BYPASSCAP" will enable connections that exceed the JDBC Maximum parallel connection cap.
DBKEY	This is an Rdb-only feature. The only valid option is ATTACH. If DBKEY=ATTACH is specified, then Rdb DBKEYS is not reused throughout the lifetime of a connection.
DBREADONLY	A list of databases in the CDD, separated by commas, which are treated as read only.
DBREADWRITE	A list of databases in the CDD, separated by commas, which are treated as read/write.
DD	The name of the JDBC data source. Required. (deprecated)
DSN	The name of the JDBC data source. Required.
EXCLUSIVE	Connect to all databases in "exclusive" mode.
GATEWAY	The name or ID address of the JDBC Server. Required.
NODE	The name of the server to connect to. This option is only useful when connecting to a CDD that only has one data source in it.
OEM	Key used for OEM redistributors.
PORT	The port that the JDBC Server is configured to listen to. The default is 7500.
PASSWORD	The CONNX password.
PWD	The CONNX password (deprecated)
READONLY	This option connects to all databases in the CDD in read only mode.
RPC	The name of a CONNX RPC to execute right after a connection has been established.
RPCBUFFERSIZE	The size of the buffer used to send data between the JDBC driver and the JDBC server. The default value is 100000 bytes.
SSL	Enable TLS/SSL encryption for this JDBC Connection. Valid values are "true" and "false"
UID	The CONNX UserID (deprecated)
USER	The CONNX UserID.

VIEWCHECK	A logical connection is made to the CDD without physically connecting to any of the data sources. All queries will return 0 rows. The defaul value is false.

.

Starting the CONNX JDBC Server

CONNX JDBC Server Definition

The CONNX JDBC Server component is a Microsoft Windows server component that communicates with the CONNX JDBC Driver. It is a Windows executable that opens a socket and listens for new connections. When it accepts a new connection, it creates a new thread that is dedicated towards communicating to that client. Installing the CONNX JDBC Server component on every machine is an optional task since only one server is required for communication with all CONNX JDBC client machines.

During installation, CONNX JDBC verifies that the target system is running on Windows. The user is prompted at the end of installation procedures to place CONNX JDBC in the Windows Startup folder.

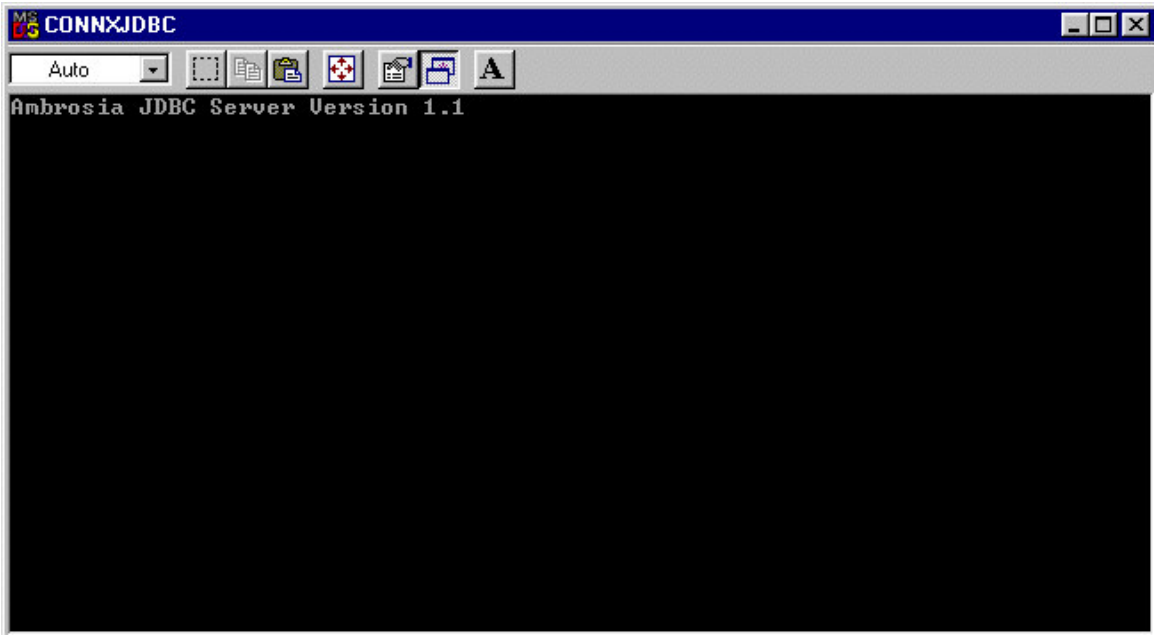
Related Topics

To start the standalone server for Windows

To start the standalone server for Windows

Note: This procedure is to be used if you have decided not to place CONNX JDBC in the Windows Startup folder during installation.

1. Click the **Start** button, and then point to **All Programs**. Point to **CONNX Driver,** then click **CONNX JDBC Server (Command Line).** The JDBC server startup begins, and the CONNXJDBC window appears and is minimized to the taskbar.



2. The server's main task is to listen for new requests and then process requests from the CONNX JDBC Java clients. While most CONNX JDBC clients need not install the server, the CONNX JDBC Server component must be installed on at least one machine.

   The CONNXJDBC.LOG file, located in C:\CONNX32, records all activity processed by the CONNX JDBC Server.

Important: *Do not close the CONNXJDBC window if there are any active client processes.*

Related Topics

CONNX JDBC Server Definition

CONNX JDBC Conformance

Conformance

CONNX JDBC is a JDBC 1.1.6 driver and fully supports that specification. CONNX JDBC is compatible with JDBC 1.2 and above, and also is supported by the major Web browsers, including Firefox, Chrome, Opera, and Internet Explorer.

Related Topics

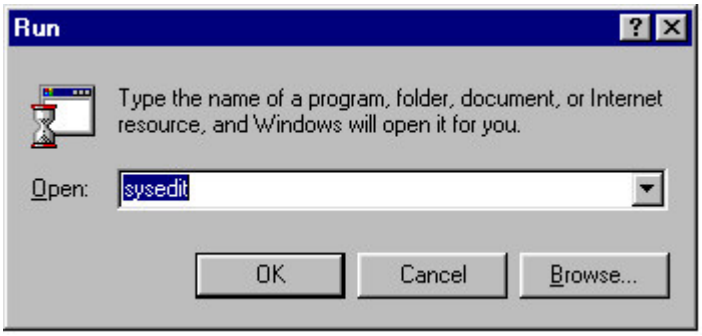
CONNX JDBC Error Messages

CONNX JDBC Error Messages

The addition of the environment variables needed to run Java and CONNX JDBC may cause you to produce an out-of-environment error message. To resolve the problem, follow this procedure:

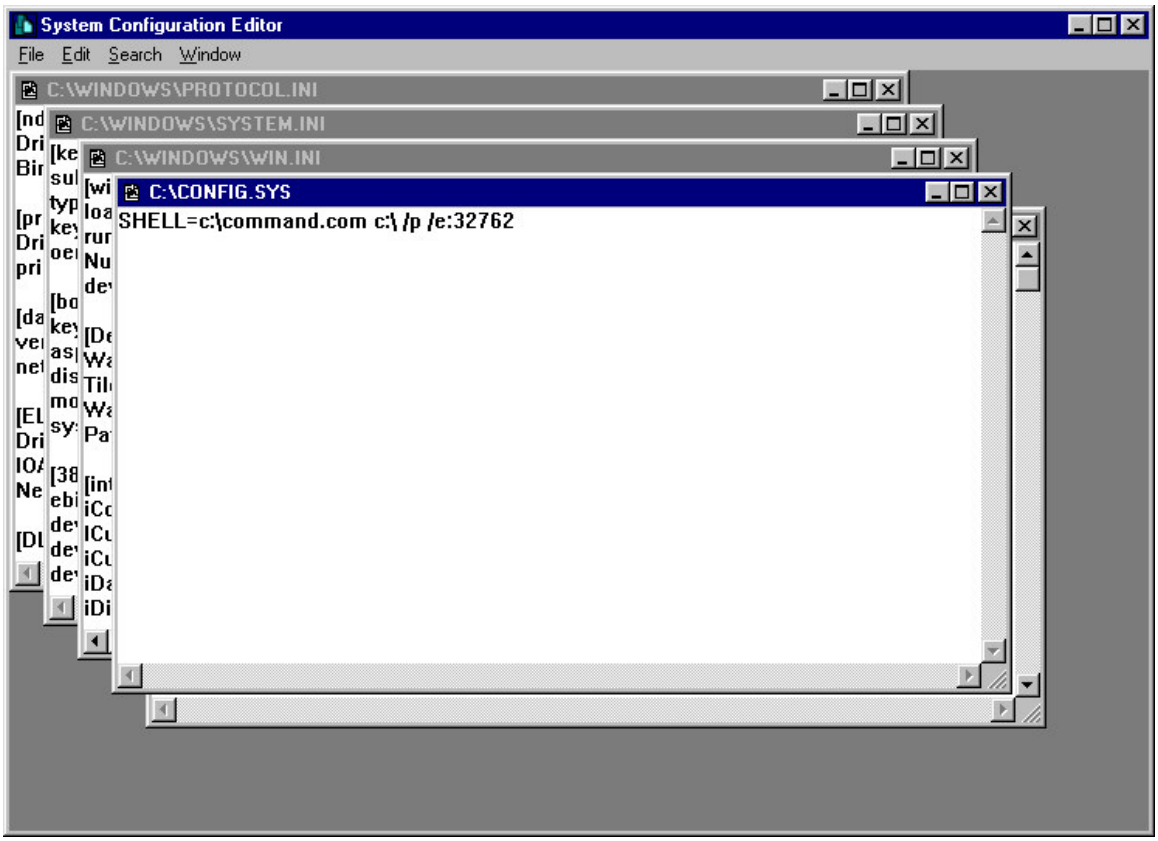
To eliminate an out-of-environment error message

1. Click **Start,** click **Run,** and then type **`sysedit`** in the Open text box.



2. Click the **OK** button.
3. Select **C:\config.sys** in the System Configuration Editor window, and then type the following statement:

```
shell=c:\command.com c:\ /p /e:32762
```



4. This command in the config.sys file increases the size of your environment to approximately 32K. If you continue to experience problems with environment space, increase the number following /e:<number> in 1K (1024 bytes) increments until your environment is sufficiently expanded.

JDBC Interfaces Supported by CONNX

JDBC Interfaces Supported by CONNX

CONNX for JDBC complies with JDBC 1.2 and has some JDBC 2.0 functionality. See the table below for a list of the JDBC interfaces that are currently supported by CONNX.

Interface Type	JDBC Interface	Supported	Not Supported
Array	getArray		x
	getArray (long arg0, int arg1)		x
	getArray (long arg0, int arg1, Map arg2)		x
	getArray (Map arg0)		x
	getBaseType()		x
	getBaseTypeName()		x
	getResultSet()		x
	getResultSet(long arg0,int arg1)		x
	getResultSet(long arg0,int art1, Map arg2)		x
	getResultSet(Map arg0)		x
Blob	getBinaryStream()		x
	getBytes(long arg0, int arg1)		x
	length()		x
	position(Blob arg0, long arg1)		x
	position(byte[] arg0, long arg1)		x
CallableStatement	getArray(int arg0)		x
	getBigDecimal(int parameterIndex()		x
	getBigDecimal(int parameterIndex, int scale)	x	
	getBlob(int arg0)		x
	getBoolean(int parameterIndex)	x	
	getByte(int pnt parameterIndex)	x	

	getBytes(int parameterIndex)	x	
	getClob(int arg0)		x
	getDate(int parameterIndex)	x	
	getDouble(int parameterIndex)	x	
	getFloat(int parameterIndex)	x	
	getInt(int parameterIndex)	x	
	getLong(int parameterIndex)	x	
	getObject(int arg0, Map arg1)		x
	getObject(int parameterIndex)	x	
	getRef(int arg0)		x
	getShort(int parameterIndex)	x	
	getString(int parameterIndex)	x	
	getTime(int arg0, Calendar calendarObj)		x
	getTime(int parameterIndex)	x	
	getTimestamp(int arg0, Calendar calendarObj)		x
	getTimestamp(int parameterIndex)	x	
	registerOutParameter(int parameterIndex, int sqlType)		x
	registerOutParameter(int parameterIndex, int sqlType, int scale)		x
	RegisterOutParameter(int parameterIndex, int sqlType, String parmString)		x
	wasNull()	x	
Clob	getAsciiStream		x
	getCharacterSTream		x
	getSubString(long arg0,		x

	int arg1)		
	length()		x
	position(Clob arg0, long arg1)		x
	position(String arg0, long arg1)		x
Connection	clearWarnings()	x	
	close()	x	
	commit()	x	
	createStatement()	x	
	createStatement(int arg0, int arg1)	x	
	getAutoCommit()	x	
	getCatalog()	x	
	getMetaData()	x	
	getTransactionIsolation()	x	
	getTypeMap()	x	
	getWarnings()	x	
	isClosed()	x	
	isReadOnly()	x	
	nativeSQL(String sql)	x	
	prepareCall(String arg0, int arg1, int arg2)	x	
	prepareCall(String sql)	x	
	prepareStatement(String arg0, int arg1, int arg2)	x	
	prepareStatement(String sql)	x	
	rollback()	x	
	setAutoCommit(boolean autoCommit)	x	
	setCatalog(String catalog)	x	
	setReadOnly(boolean readOnly)	x	
	setTransactionIsolation(int level)	x	
	setTypeMap(Map arg0)	x	

Array

Interface Type	JDBC Interface	Supported	Not Supported
Array	getArray		x
	getArray (long arg0, int arg1)		x
	getArray (long arg0, int arg1, Map arg2)		x
	getArray (Map arg0)		x
	getBaseType()		x
	getBaseTypeName()		x
	getResultSet()		x
	getResultSet(long arg0,int arg1)		x
	getResultSet(long arg0,int art1, Map arg2)		x
	getResultSet(Map arg0)		x

Blob

Interface Type	JDBC Interface	Supported	Not Supported
Blob	getBinaryStream()	x	
	getBytes(long arg0, int arg1)	x	
	length()	x	
	position(Blob arg0, long arg1)		x
	position(byte[] arg0, long arg1)		x

CallableStatement

Interface Type	JDBC Interface	Supported	Not Supported
CallableStatement	getArray(int arg0)		x
	getBigDecimal(int parameterIndex()	x	
	getBigDecimal(int parameterIndex, int scale)	x	
	getBlob(int arg0)	x	
	getBoolean(int parameterIndex)	x	
	getByte(int pnt parameterIndex)	x	
	getBytes(int parameterIndex)	x	
	getClob(int arg0)	x	
	getDate(int parameterIndex)	x	
	getDouble(int parameterIndex)	x	
	getFloat(int parameterIndex)	x	
	getInt(int parameterIndex)	x	
	getLong(int parameterIndex)	x	
	getObject(int arg0, Map arg1)		x
	getObject(int parameterIndex)	x	
	getRef(int arg0)		x
	getShort(int parameterIndex)	x	
	getString(int parameterIndex)	x	
	getTime(int arg0, Calendar calendarObj)		x
	getTime(int parameterIndex)	x	
	getTimestamp(int arg0, Calendar calendarObj)		x
	getTimestamp(int parameterIndex)	x	
	registerOutParameter(int parameterIndex, int sqlType)		x
	registerOutParameter(int parameterIndex, int sqlType, int scale)		x
	RegisterOutParameter(int parameterIndex, int sqlType, String parmString)		x
	wasNull()	x	

Clob

Interface Type	JDBC Interface	Supported	Not Supported
Clob	getAsciiStream	x	
	getCharacterStream	x	
	getSubString(long arg0, int arg1)	x	
	length()	x	
	position(Clob arg0, long arg1)		x
	position(String arg0, long arg1)		x

Connection

Interface Type	JDBC Interface	Supported	Not Supported
Connection	clearWarnings()	x	
	close()	x	
	commit()	x	
	createStatement()	x	
	createStatement(int arg0, int arg1)	x	
	getAutoCommit()	x	
	getCatalog()	x	
	getMetaData()	x	
	getTransactionIsolation()	x	
	getTypeMap()	x	
	getWarnings()	x	
	isClosed()	x	
	isReadOnly()	x	
	nativeSQL(String sql)	x	
	prepareCall(String arg0, int arg1, int arg2)	x	
	prepareCall(String sql)	x	
	prepareStatement(String arg0, int arg1, int arg2)	x	
	prepareStatement(String sql)	x	
	rollback()	x	
	setAutoCommit(boolean autoCommit)	x	
	setCatalog(String catalog)	x	
	setReadOnly(boolean readOnly)	x	
	setTransactionIsolation(int level)	x	
	setTypeMap(Map arg0)	x	

DatabaseMetaData

Interface Type	JDBC Interface	Supported	Not Supported
DatabaseMetaData	allProceduresAreCallable()	x	
	allTablesAreSelectable()	x	
	dataDefinitionCausesTransactionCommit	x	
	dataDefinitionIgnoredinTransactions()	x	
	deletesAreDetected(int arg0)		x
	doesMaxRowSizeIncludeBlobs()	x	
	getBestRowIdentifier(String catalog, String schema, String table, int scope, boolean nullable)	x	
	getCatalogs()	x	
	getCatalogSeperator()	x	
	getCatalogTerm()	x	
	getColumnPrivileges(String catalog, String schema, String table, String columnNamePattern())	x	
	getColumns(STring catalog, STring schemaPatter, STring tableNamePattern, String columnNamePattern)	x	
	getConnection()	x	
	getCrossReference(String PrimaryCatalog, String primarySchema, String primaryTable, String foreignCatalog, String foreignSchema, String foreignTable)	x	
	getDatabaseProductName()	x	
	getDatabaseProductVersion()	x	
	getDefaultTransactionIsolation()	x	
	getDriverMajorVersion()	x	
	getDriverMinorVersion()	x	
	getDriverName()	x	
	getDriverVersion()	x	
	getExportedKeys(String catalog, String schema, String table)	x	
	getExtraNameCharacters()	x	
	getIdentifierQuoteString()	x	
	getImportedKeys(String catalog, String schema, String table)	x	
	getIndexInfo(String catalog, STring schema, String table)	x	

	getMaxBinaryLiteralLength()	x	
	getMaxCatalogNameLength()	x	
	getMaxCharLiteralLength()	x	
	getMaxColumnNameLength()	x	
	getMaxColumnsInGroupBy()	x	
	getMaxColumnsInIndex()	x	
	getMaxColumninOrderBy()	x	
	getMaxColumnsInSelect()	x	
	getMaxColumnsInTable()	x	
	getMaxConnections()	x	
	getMaxCursorNameLength()	x	
	getMaxIndexLength()	x	
	getMaxProcedureNameLength()	x	
	getMaxRowSize()	x	
	getMaxSchemaNameLength()	x	
	getMaxStatementLength()	x	
	getMaxStatements()	x	
	getMaxTableNameLength()	x	
	getMaxTablesInSelect	x	
	getMaxUserNameLength()	x	
	getNumericFunctions()	x	
	getPrimaryKeys(String catalog, String schema, String table)	x	
	getProcedureColumns(Strintg cataslog, String schemaPattern, String procedureNamePattern, String columnNamePattern)	x	
	getProcedures(String catalog, String schemaPattern, String procedureNamePattern)	x	
	getProcedureTerm()	x	
	getSchemas()	x	
	getSchemaTerm()	x	
	getSearchStringEscape()	x	
	getSQLKeywords()	x	
	getStringFunctions()	x	
	getSystemFunctions()	x	
	getTablePrivileges(String catalog, String schemaPattern, String table)		x
	getTables(STring catalog, STring schemaPattern, STring tablePattern, String Types[])	x	
	getTableTypes()	x	

	getTimeDataFunctions()	x	
	getTypeInfo()	x	
	getUDTS(String arg0, String arg1, String arg2, int[]arg3)		x
	getURL()	x	
	getUserName()	x	
	getVersionColumns(String catalog, String schema, String table)	x	
	insertsAreDetected()		x
	isCatalogAtStart()	x	
	isReadOnly()	x	
	nullPlusNonNullsIsNull()	x	
	nullsAreSortedAtEnd()	x	
	nullsAreSortedAtStart	x	
	nullsAreSortedHigh()	x	
	nullsAreSortedLow()	x	
	othersDeletesAreVisible(int arg0)		x
	othersInsertsAReVisible(int arg0)		x
	othersUpdatesAreVisible(int arg0)		x
	ownDeletesAreVisible(int arg0)		x
	ownInsertsAreVisible(int arg0)		x
	ownUpdatesAreVisible(int arg0)		x
	storesLowerCaseIdentifiers(	x	
	storesLowerCaseQuotedIdentifiers()	x	
	storesMixedCaseIdentifiers()	x	
	storesMixedCaseQuotedIdentifiers()	x	
	storesUpperCaseIdentifiers()	x	
	storesUpperCaseQuotedIdentifiers()	x	
	supportsAlterTableWithAddColumn()	x	
	supportsAlterTableWithDropColumn()	x	
	supportsANSI92EntryLevelSQL()	x	
	supportsANSI92FullSQL()	x	
	supportsANSI92IntermediateSQL()	x	
	supportsBatchUpdates()		x
	supportsCatalogsInDataManipulations	x	
	supportsCatalogsInIndexDefinitions	x	
	supportsCatalogsInPrivilegeDefinitions()	x	
	supportsCatalogsInProcedureCalls()	x	

	supportsCatalogsInTableDefinitions()	x	
	supportsColumnAliasing()	x	
	supportsConvert()	x	
	supportsconver(int fromType, int to Type)	x	
	supportsCoreSQLGrammar()	x	
	supportsCorrelatedSubQueries()	x	
	supportsDataDefinitionsAndDataManipuilationTransactions()	x	
	supportsDataManipulationTransactionsOnly()	x	
	supportsDifferentTAbleCorrelationNames()	x	
	supportsExpressionsInOrderBy()	x	
	supportsExtendedSQLGrammar()	x	
	supportsFullOuterJoins()	x	
	supportsGroupBy()	x	
	supportsGroupByBeyondSelect()	x	
	supportsGroupByUnrelated()	x	
	supportsIntegrityEnhancementFacility()	x	
	supportsLikeEscapeClause()	x	
	supportsLimitedOuterJoins	x	
	supportsMinimumSQLGrammar()	x	
	supportsMixedCaseIdentifiers()	x	
	supportsMixedCaseQuotedIdentifiers()	x	
	supportsMultipleResultSets()	x	
	supportsMultipleTransactions()	x	
	supportsNonNullableColumns()	x	
	supportsOpenCursorsAcrossCommit()	x	
	supportsOpenCursorsAcrossRollback()	x	
	supportsOrderByUnrelated()	x	
	supportsOuterJoins()	x	
	supportsPositionedDelete()	x	
	supportsPositionedUpdate()	x	
	supportsResultSetConcurrency(int arg0, int arg1)		x
	supportsResultSetType(int arg0)	x	
	supportsSchemasInDataManipulation()	x	
	supportsSchemasInIndexDefinitions()	x	
	supportsSchemasInPrivilegeDefinitions()	x	
	supportsSchemasInProcedureCalls()	x	
	supportsInTableDefinitions()	x	

	supportsSelectForUpdate()	x	
	supportsStoredProcedures()	x	
	supportsSubQueriesInComparisons()	x	
	supportsSubqueriesInExists()	x	
	supportsSubqueriesInIns()	x	
	supportsSubqueriesInQualified()	x	
	supportsTAbleCorrelationNames()	x	
	supportsTransactionIsolationLevel(int level)	x	
	supportsTransactions()	x	
	supportsUnions()	x	
	supportsUnionAll()	x	
	updatesAreDetected(int arg0)		x
	useLocalFilePerTable()	x	
	useLocalFiles()	x	

Driver

Interface Type	JDBC Interface	Supported	Not Supported
Driver	acceptsURL(STring url)	x	
	connect(STring url, PropertiesInfo)	x	
	getMajorVersion()	x	
	getMinorVersion()	x	
	getPropertyInfo(String url, Properties info)	x	
	jdbcCompliant()	x	

InputStream

Interface Type	JDBC Interface	Supported	Not Supported
InputStream	read()	x	
	setBlobBuffer()	x	
	generateError(int status)	x	
	getBlobRaw()	x	
	getBlobString()	x	

PreparedStatement

Interface Type	JDBC Interface	Supported	Not Supported
PreparedStatement	addBatch	x	
	clearParameters	x	
	execute()	x	
	executeQuery()	x	
	getMetaData()	x	
	setArray(int arg0, Array arg1)		x
	setASCIIStream(int parameterIndex, InputStream x, int length)	x	
	setBigDecimal(int pindex, BigDecimal x)	x	
	setBinaryStream(int pIndex, InputStream x, int length)	x	
	setBlob(int arg0, Blob arg1)		x
	setBoolean(int pindex, boolean x)	x	
	setByte(int pindex, byte x)	x	
	setBytes(int pindex, byte[]x)	x	
	setCharacterStream(int arg0, Reader arg1, int arg2)		x
	setClob(int arg0, Clob arg1)		x
	setDate(int arg0, Date arg1, Calendar arg2)		x
	setDate(Int pindex, Datex)	x	
	setDouble(int pindex, doublex)	x	
	setFloat(int pindex, float x)	x	
	setInt(int pindex, int x)	x	
	setLong(int pinde, long x)	x	
	setNull(int arg0, int arg2, String arg2)		x
	setNull(int pindex, int sqlType)	x	
	setObject(int pindex, Object x)	x	
	setObject(int pindex, Object x, int targetSqlType)	x	
	setObject(jint pindex, Object x, int targetSqlType, int scale)	x	
	setRef(int arg0, Ref arg1)	x	
	setShort(int pindex, short x)	x	
	setString(int pindex, String x)	x	
	setTime(int arg0, Time arg1, Calendar arg2)		x
	setTime(int pindex, Time x)	x	

	setTimestamp(int arg0, Timestamp arg1, Calendar arg2)		x
	setTimestamp(int pindex, Timestamp x)	x	
	setUnicodeStream(int pindex, InputStream x, int length)	x	

Ref

Interface Type	JDBC Interface	Supported	Not Supported
Ref	getBaseTypeNormal		x

ResultSet

Interface Type	JDBC Interface	Supported	Not Supported
ResultSet	absolute()	x	
	afterLast()	x	
	beforeFirst()	x	
	cancelRowUpdates()		x
	clearWarnings()	x	
	close()	x	
	deleteRow()	x	
	findColumnString(String columnName)	x	
	first()	x	
	getArray(int arg0)		x
	getArray(String arg0)		x
	getASCIIStream(int columnIndex)	x	
	getASCIIStream(String columnName)	x	
	getBigDecimal(int arg0)	x	
	getBigDecimal(int cIndex, int scale)	x	
	getBigDecimal(String arg0)	x	
	getBigDecimal(STring cName,int scale)	x	
	getBinaryStream(int cIndex)	x	
	getBinaryStream(String Name)	x	
	getBlob(int arg0)	x	
	getBlob(String arg0)	x	
	getBoolean(int cIndex)	x	
	getBoolean(String Name)	x	
	getByte(int cIndex)	x	
	getByte(String cName)	x	
	getBytes(int cIndex)	x	
	getCharacterStream(int arg0)	x	
	getCharacterStream(String arg0)	x	
	getClob(int arg0)	x	
	getClob(String arg0)	x	
	getConcurrency()	x	
	getCursorName()	x	
	getDate(int arg0, Calendar arg1)		x
	getDate(int cIndex)	x	
	getDate(String arg0, Calendar arg1)		x

	getDate(String cName)	x	
	getDouble(int cIndex)	x	
	getDouble(String cName)	x	
	getFetchDirection()	x	
	getFetchSize()	x	
	getFloat(Int cIndex)	x	
	getFloat(String cName)	x	
	getInt(int cIndex)	x	
	getInt(String cName)	x	
	getLong(int cIndex)	x	
	getLong(String cName)	x	
	getMetaData()	x	
	getObject(int arg0, Map arg1)	x	
	getObject(int cIndex)	x	
	getObject(String arg0, Map arg1)		x
	getObject(String cName)	x	
	getRef(int arg0)		x
	getRef(String arg0)		x
	getRow()	x	
	getShort(int cIndex)	x	
	getShort(String cName)	x	
	getStatement()	x	
	getString(int cIndex)	x	
	getString(String cName)	x	
	getTime(int arg0, Calendar arg1)		x
	getTime(String cName)	x	
	getTimestamp(int arg0, Calendar arg1)		x
	getTimestamp(int cIndex)	x	
	getTimestamp(String arg0, Calendar arg1)		x
	getTimestamp(String cName)	x	
	getType()	x	
	getUnicodeStream(int cIndex)		x
	getUnicodeStream(String columnName)	x	
	getWarnings()	x	
	insertRow()		
	isAfterLast()	x	
	isBeforeFirst()	x	
	isFirst()	x	

	isLast()	x	
	last()	x	
	moveToCurrentRow()		x
	moveToInsertRow()		x
	next()	x	
	previous()	x	
	refreshRow()		x
	relative(int arg0)	x	
	rowDeleted()		x
	rowInserted()		x
	rowUpdated()		x
	setFetchDirection(int arg0)	x	
	setFetchSize(int arg0)	x	
	updateAsciiStream(int arg0, InputStream arg1, int arg2)		x
	updateAsciiStream(String arg0, InputStream arg1, int arg2)		x
	updateBigDecimal(int arg0, BigDecimal arg1)	x	
	updateBigDecimal(String arg0, Big Decimal arg1)	x	
	updateBinaryStream(int arg0, InputStream arg1, int arg2)		x
	updateBinaryStream(String arg0, InputStream arg1, int arg2)		x
	updateBoolean(int arg0, boolean arg1)	x	
	updateBoolean(String arg0, boolean arg1)	x	
	updateByte(int arg0, byte arg1)	x	
	updateByte(String arg0, byte arg1)	x	
	updateBytes(int arg0, byte[] arg1)	x	
	updateBytes(String arg0, byte[]Arg1)	x	
	updateCharacterStream(int arg0, Reader arg1, int arg2)		x
	updateCharacterStream(String arg0, Reader arg1, int arg2)		x
	updateDate(int arg0, Date arg1)	x	
	updateDate(String arg0, Date arg1)	x	
	updateDouble(int arg0, double arg1)	x	
	updateFloat(int arg0, float arg1)	x	
	updateFloat(String arg0, float arg1)	x	
	updateInt(int arg0, int arg2)	x	
	updateInt(String arg0, int arg1)	x	

	updateLong(int arg0, long arg1)	x	
	updateLong(String arg0, long arg1)	x	
	updateNull(int arg0)	x	
	updateNull(String arg0)	x	
	updateObject(int arg0, Object arg1)	x	
	updateObject(String arg0, Object arg1)	x	
	updateObject(String arg0, Object arg1, int arg2)		x
	updateRow()	x	
	updateShort(int arg0, short arg1)	x	
	updateShort(String arg0, short arg1)	x	
	updateString(int arg0, String arg1)	x	
	updateString(String arg0, String arg1)	x	
	updateTime(int arg0, Time arg1)	x	
	updateTime(String arg0, Time arg1)	x	
	updateTimestamp(int arg0, Timestamp arg1)	x	
	updateTimestamp(String arg0, Timestamp arg1)	x	
	wasNull()	x	

ResultSetMetaData

Interface Type	JDBC Interface	Supported	Not Supported
ResultSetMetaData	getCatalogName(int column)	x	
	getColumnClassName(int column)		x
	getColumnCount()	x	
	getColumnDisplaySize(int column)	x	
	getColumnLabel(int column)	x	
	getColumnName(int column)	x	
	getColumnType(int column)	x	
	getMetaData()	x	
	getPrecision(int column)	x	
	getScale(int column)	x	
	getSchemaName(int column)	x	
	getTableName(int column)	x	
	isAutoIncrement(int column)	x	
	isCaseSensitive(int column)	x	
	isCurrency(int column)	x	
	isDefinitelyWritable(int column)	x	
	isNullable(int column)	x	
	isReadOnly(int column)	x	
	isSearchable(int column)	x	
	isSigned(int column)	x	
	isWritable(int column)	x	

SQLData

Interface Type	JDBC Interface	Supported	Not Supported
SQLData	getSQLTypeName		x
	readSQL(SQLInput arg0, String arg1)		x
	writeSQL(SQLOutput arg0)		x

SQLInput

Interface Type	JDBC Interface	Supported	Not Supported
SQLInput	readArray()		x
	readAsciiStream()		x
	readBigDecimal()		x
	readBinaryStream()		x
	readBlob()		x
	readBoolean()		x
	readByte()		x
	readBytes()		x
	readCharacterStream()		x
	readClob()		x
	readDate()		x
	readDouble()		x
	readFloat()		x
	readInt()		x
	readLong()		x
	readObject()		x
	readRef()		x
	readShort()		x
	readString()		x
	readTime()		x
	readTimestamp()		x
	wasNull()		x

SQLOutput

Interface Type	JDBC Interface	Supported	Not Supported
SQLOutput	writeArray(Array arg0)		x
	writeAsciiStream(InputStream arg0)		x
	writeBigDecimal(BigDecimal arg0)		x
	writeBinaryStream(InputStream arg0)		x
	writeBlob(Blob arg0)		x
	writeBoolean(boolean arg0)		x
	writeByte(byte arg0)		x
	writeBytes(byte[] arg0)		x
	writeCharacterStream(Reader arg0)		x
	writeClob(Clob arg0)		x
	writeDate(Date arg0)		x
	writeDouble(double arg0)		x
	writeFloat(float arg0)		x
	writeInt(int arg0)		x
	writeLong(long arg0)		x
	writeObject(SQLData rg0)		x
	writeRef(Ref arg0)		x
	writeShort(short arg0)		x
	writeString(String arg0)		x
	writeStruct(Struct arg0)		x
	writeTime(Time arg0)		x
	wasTimestamp(Timestamp arg0)		x

Struct

Interface Type	JDBC Interface	Supported	Not Supported
Struct	getAttributes()		x
	getAttributes(Map arg0)		x
	getSQLTypeName()		x

Statement

Interface Type	JDBC Interface	Supported	Not Supported
Statement	cancel()	x	
	clearBatch()	x	
	clearWarnings()	x	
	close()	x	
	execute(String sql)	x	
	executeBatch()	x	
	executeQuery(String sql)	x	
	executeUpdate(String sql)	x	
	getConnection()	x	
	getFetchDirection()	x	
	getFetchSize()	x	
	getMarketString()	x	
	getMaxFieldSize()	x	
	getMaxRows()	x	
	getMoreResults()	x	
	getQueryTimeout()	x	
	getResultSet()	x	
	getResultSetConcurrency()	x	
	getResultSetType()	x	
	getUpdateCount()	x	
	getWarnings()	x	
	setCursorName(String name)	x	
	setEscapeProcessing(boolean enable)	x	
	setFetchDirection(int arg0)	x	
	setFetchSize(int arg0)	x	
	setMaxFieldSize(int max)	x	
	setMaxRows(int max)	x	
	setQueryTimeout(int seconds)	x	

Chapter 7 - CONNX .NET Data Provider

General Information

The CONNX .NET Data Provider requires the Microsoft .NET Framework. The .NET Framework can be found on the CONNX CD-ROM, or can be downloaded from the Microsoft Web site at http://msdn.microsoft.com/netframework/downloads/

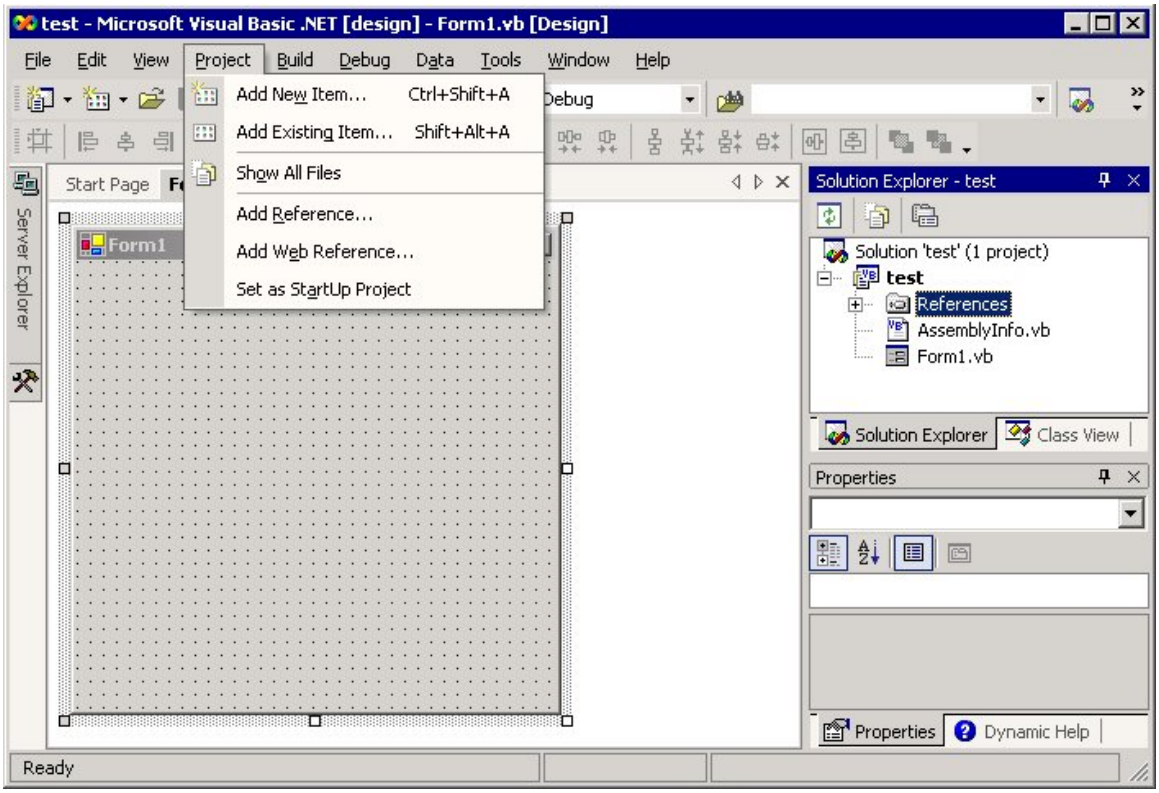
Once the framework is installed, Microsoft Visual Studio .NET or Visual Studio .NET 2003 can be used to quickly develop applications that take advantage of the CONNX .NET Data Provider, a feature of CONNX that can be used for accessing data from .NET applications.

The .NET Data Provider supports Unicode and ANSI data types.

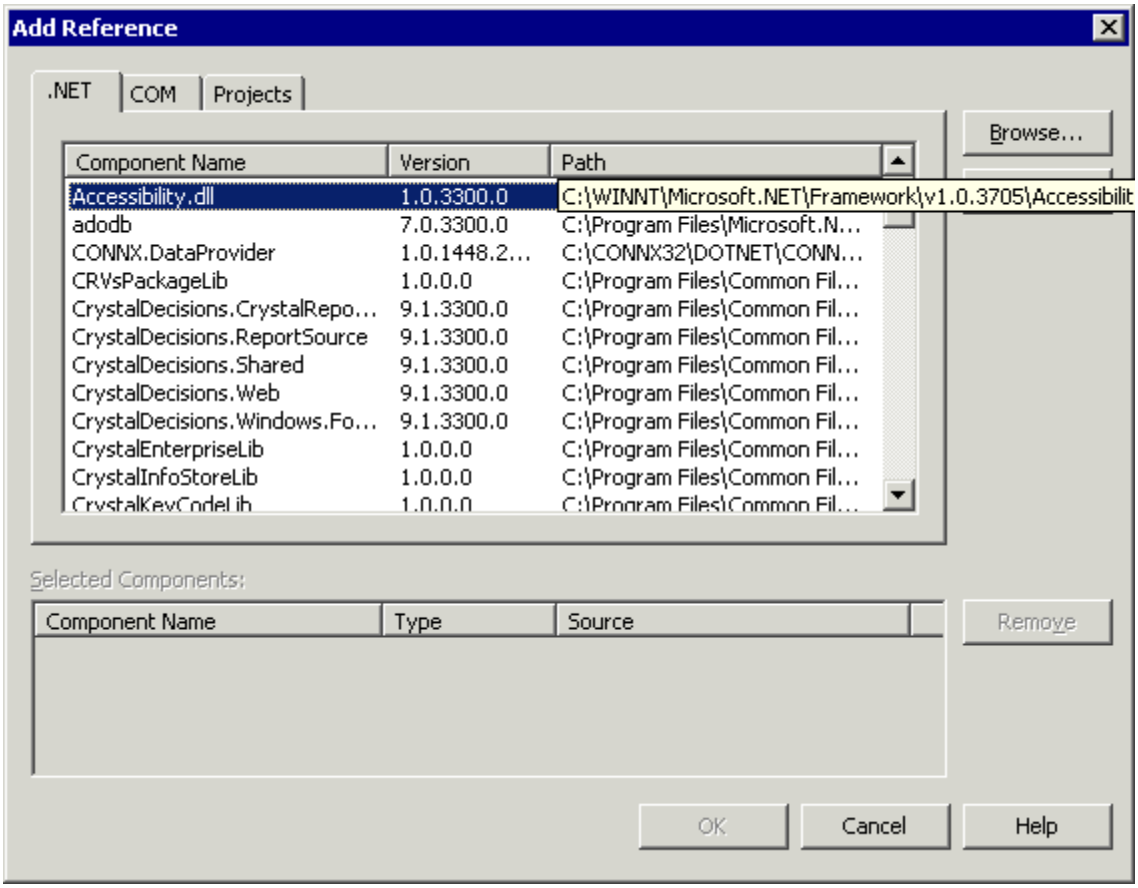
To add a reference to the CONNX .NET Data Provider

To use the CONNX .NET Data Provider within Microsoft Visual Studio .NET or Visual Studio .NET 2003, you must first add a reference to the CONNX.Provider.dll file containing the CONNX .NET Data Provider objects.

1. In Microsoft Visual Studio .NET or Visual Studio .NET 2003, open a new project or use a current project, and then select the **Project** menu.



2. Select **Add Reference.** You may also choose to right-click the Project name as it is listed in the Microsoft Visual Studio .NET or Visual Studio .NET 2003 pane in the Solution Explorer and select **Add Reference.** A third option is to right-click the **References** folder in the Solution Explorer and select **Add Reference.**
3. The **Add Reference** dialog box appears.



4. On the **.NET** tab in the **Add Reference** dialog box, click the **Browse** button.
5. The **Select Component** dialog box appears. Navigate to the **Windows\System32** folder and select the **CONNX.DataProvider.dll** file.
6. Click the **OK** button to return to the **Add Reference** dialog box.
7. Select the **CONNX.DataProvider.dll** in the **Add Reference** dialog box, and then click the **OK** button. The objects in the CONNX .NET Data Provider can now be used within your code. This enables you to add the CONNX .NET Data Provider object to your application.

To specify a name space for the added reference

The .NET NameSpace for the CONNX .NET Data Provider is System.Data.CONNX. To reference the CONNX .NET Data Provider objects by the class name instead of the fully qualified name, use the following lines at the top of your code:

```
VB.NET
Imports System.Data.CONNX
```

or

```
C#
using System.Data.CONNX;
```

Reference the CONNX .NET Data Provider objects by either of the following methods:

```
VB NET
Dim cnxCommand1 as System.Data.CONNX.CNXCommand
Dim cnxCommand2 as CNXCommand
```

or

```
C#
System.Data.CONNX.CNXCommand cnxCommand1;
CNXCommand cnxCommand2;
```

CONNX .NET Data Provider Object Model

The following tables contain descriptions of the CONNX .NET Data Provider object model. All of the classes have a brief description of the overall functionality provided by that class. All Properties are documented with the data type and a description of the property. All Methods are documented with the arguments and the argument types, the return value and return value type (if applicable), and a description of the method. All Events are documented with the event arguments and a description of the event.

Each Class listed in the tables on the following pages is divided into Constructors, Properties, and Methods or Attributes. It should be noted that possession of any one category does not necessitate the existence of any other.

Class Tables

Class : CNXCommand

Class : CNXCommand		
Namespace	System.Data.CONNX	
Description	The CNXCommand class is used as the main interface for retrieving data from a CONNX data source. This class accepts an SQL statement to process zero or more CNXParameters objects, and optional CNXTransaction object, and a CNXConnection object, all of which are used to retrieve data and present it in various ways, depending upon the Execute method used.	
Constructors	**CNXCommand**	
	Description:	Implement the default constructor here. The default constructor is used to create an empty CNXCommand object, which has an empty SQL statement, an empty CNXConnection object, and an empty CNXTransaction object.
	Parameters:	None
	CNXCommand	
	Description:	This constructor creates a new CNXCommand object with the specified SQL statement.
	Parameters:	**cmdText**
		Type: String
		Description: SQL statement assigned to the CommandText property.
	CNXCommand	
	Description:	This constructor creates a new CNXCommand object with the specified SQL statement and the specified connection object.
	Parameters:	**cmdText**
		Type: String
		Description: SQL statement assigned to the CommandText property.
		connection

		Type:	CNXConnection
		Description:	CNXConnection object assigned to the Connection property.
	CNXCommand		
	Description:	This constructor creates a new CNXCommand object with the specified SQL statement, the specified CNXConnection object, and the specified CNXTransaction object.	
	Parameters:	**cmdText**	
		Type:	String
		Description:	SQL statement assigned to the CommandText property.
		connection	
		Type:	CNXConnection
		Description:	CNXConnection object assigned to the Connection property.
		txn	
		Type:	CNXTransaction
		Description:	CNXTransaction object assigned to the Transaction property.
Properties	**CommandText**		
	Return Type:	String	
	Description:	The CommandText property gets/sets the SQL statement processed when an Execute method is called.	
	CommandTimeout		
	Return Type:	Int	
	Description:	The CommandTimeout property is not supported at this time. Setting to non-zero value throws an exception.	
	CommandType		

	Return Type:	CommandType
	Description:	The CommandType property gets/sets the CommandType.
	Connection	
	Return Type:	CNXConnection
	Description:	The Connection property gets/sets the CNXConnection object.
	Parameters	
	Return Type:	CNXParameterCollection
	Description:	The Parameters property gets the CNXParameterCollection that contains the CNXParameter objects.
	Transaction	
	Return Type:	IDbTransaction
	Description:	The Transaction property gets/sets the CNXTransaction object.
	UpdatedRowSource	
	Return Type:	UpdateRowSource
	Description:	The UpdatedRowSource property specifies how results are applied to the row that is being updated. This property is not implemented and is provided for interface compatibility.
Methods	**Cancel**	
	Return Type:	Void
	Description:	The Cancel method is not supported at this time and throws an exception. New CNXParameter object
	Parameters:	None
	CreateParameter	
	Return Type:	IDbDataParameter
	Description:	The CreateParameter method returns a newly created CNXParameter object. New CNXParameter object.
	Parameters:	None
	Dispose	

Return Type:	Void	
Description:	The Dispose method releases any resources used by the object.	
Parameters	disposing	
	Type:	Bool
	Description:	Specified if managed resources should also be disposed (true).
ExecuteNonQuery		
Return Type:	int	
Description:	The ExecuteReader method executes the Action query and returns the number of records affected.	
Parameters:	None	
ExecuteReader		
Return Type:	IDataReader	
Description:	The ExecuteReader method executes the SQL statement contained in the CommandText property and returns a CNXDataReader object that can be used to iterate through the results.	
Parameters:	None	
ExecuteReader		
Return Type:	IDataReader	
Description:	The ExecuteReader method executes the SQL statement contained in the CommandText property and returns a CNXDataReader object that can be used to iterate through the results.	
Parameters:	**Behavior**	
	Type:	CommandBehavior
	Description:	The CommandBehavior specifies if the connection should be closed after the query is executed.
ExecuteScalar		

	Return Type:	Object
	Description:	The ExecuteScalar method executes the SQL statement contained in the CommandText property and returns the first column of the first row of the resulset.
	Parameters:	None
	Prepare	
	Return Type:	Void
	Description:	The Prepare method is not implemented at this time and will not throw an exception.
	Parameters:	None

Class : CNXConnection

Class : CNXConnection		
Namespace:	System.Data.CONNX	
Description:	The CNXConnection object is used to handle the physical connection to the database. Connection Pooling is available to conserve server resources, while still providing exceptional performance. The CNXConnection object also contains details about the data source that it is connected to.	
Constructors:	**CNXConnection**	
	Description:	This constructor creates a new CNXConnection object. The ConnectionState property is closed, and the ConnectionString property is blank.
	Parameters:	None
	CNXConnection	
	Description:	This constructor accepts a String value assigned to the ConnectionString property.
	Parameters:	**sConnString**
		Type: String
		Description: Yes
Properties:	**ConnectionString**	
	Type:	String
	Description:	The ConnectionString property gets/sets the connection String for the current database connection.
	ConnectionPoolTimeout	
	Type:	Int
	Description:	The ConnectionPoolTimeout specifies how many seconds the connection remains in the pool before it is closed.
	ConnectionTimeout	
	Type:	N/A
	Description:	N/A

	Database		
	Type:	String	
	Description:	The Database property is not implemented at this time.	
	PoolConnection		
	Type:	Boolean	
	Description:	Determines whether a connection will be added to the connection pool.	
	State		
	Type:	ConnectionState	
	Description:	The State property returns the ConnectState object that has the current state of the connection.	
	TransactionCapabilities		
	Type:	IDbTransaction	
	Description:	The TransactionCapabilities property gets the CNXTransactionCapabilities object that contains the transaction capabilities for the current database connection.	
Methods:	**BeginTransaction**		
	Return Type:	IDbTransaction	
	Description:	The BeginTransaction method attempts to start a transaction on the current connection.	
	Parameters:	**None**	
	BeginTransaction		
	Return Type:	IDbTransaction	
	Description:	The BeginTransaction method attempts to start a transaction on the current connection. The IsolationLevel object is not used at this time.	
	Parameters:	dbName	
		Type:	IsolationLevel
		Description:	Transaction isolation level
	ChangeDatabase		
	Return Type:	Void	

Description:	The ChangeDatabase method is not implemented at this time. String value consisting of database name to which you may want to change.	
Parameters:	dbName	
	Type:	string
	Description:	New database name
Close		
Return Type:	Void	
Description:	The Close method either closes or releases the current connection to the connection pool, depending on the ConnectionPooling property.	
Parameters:	None	
CreateCommand		
Return Type:	IDbCommand	
Description:	The CreateCommand method creates a new CNXCommand object with the Connection property set to the current connection. A CNXCommand object with the Connection property set to the current connection.	
Parameters:	None	
Dispose		
Return Type:	void	
Description:	The Dispose method releases any resources held by the object.	
Parameters:	disposing	
	Type:	Bool
	Description:	Specifies if managed resources should also be disposed (true).
Open		
Return Type:	Void	
Description:	The Dispose method releases any resources held by the object.	
Parameters:	None	

Class : CNXDataAdapter

Class : CNXDataAdapter			
Namespace:	System.Data.CONNX		
Description:	The CNXDataAdapter class handles much of the work related to presenting the native resultset in a .NET-friendly format.		
Constructors:	**CNXDataAdapter**		
	Description:	A default constructor that simply creates a new CNXDataAdapter.	
	Parameters:	None	
	CNXDataAdapter		
	Description:	This constructor creates a new CNXDataAdapter with the provided CNXCommand object set to the SelectCommand property.	
	Parameters:	**selectCommand**	
		Type:	CNXCommand
		Description:	CNXCommand object which is set to the SelectCommand property.
	CNXDataAdapter		
	Description:	This constructor creates a new CNXDataAdapter with a new CNXCommand object and an existing CNXConnection object.	
	Parameters:	**selectCommandText**	
		Type:	String
		Description:	SQL statement to be used in the creation of a new CNXCommand object which will be used for the SelectCommand property.
		selectConnectionString	
		Type:	String
		Description:	Connection string to be used in the creaton of a new CNXConnection

			object.
	CNXDataAdapter		
	Description:	This constructor creates a new CNXData Adapter with a new CNXCommand object and an existing CNXconnection object.	
	Parameters:	**selectCommandText**	
		Type:	String
		Description:	SQL statement to be used in the creation of a new CNXCommand object which will be used for the SelectCommand property.
		selectConnection	
		Type:	CNXConnection
		Description:	An existing CNXConnection object.
Properties:	**DeleteCommand**		
	Return Type:	CNXCommand	
	Description:	The DeleteCommand property gets/sets the CNXCommand object used for Delete statements.	
	SelectCommand		
	Return Type:	CNXCommand	
	Description:	The SelectCommand property gets/sets the CNXCommand object used for Select statements.	
	InsertCommand		
	Return Type:	CNXCommand	
	Description:	The InsertCommand property gets/sets the CNXCommand object used for Insert statements.	
	UpdateCommand		
	Return Type:	CNXCommand	
	Description:	The UpdateCommand property gets/sets the CNXCommand object used for Update statements.	

Methods	Fill		
	Description:	Adds or refreshes rows in the DataSet to match those in the data source.	
	Parameters:	DataSet	
		Type:	DataSet
		Description:	DataSet to fill with data (and schema) from data source.
	Fill		
	Description:	Adds or refreshes rows in the DataSet to match those in the data source.	
	Parameters:	DataTable	
		Type:	DataTable
		Description:	DataTable to fill with data (and schema) from data source.
	Fill		
	Description:	Adds or refreshes rows in the DataSet to match those in the data source.	
	Parameters:	DataSet	
		Type:	DataSet
		Description:	DataSet to fill with data (and schema) from data source.
		srcTable	
		Type:	String
		Description:	Table name for newly created table.

Fill		
Description:	Adds or refreshes rows in the DataSet to match those in the data source.	
Parameters:	DataSet	
	Type:	DataSet
	Description:	DataSet to fill with data (and schema) from data source.
	StartRecord	
	Type:	Int
	Description:	0-based record from which to start retrieval.
	MaxRecords	
	Type:	Int
	Description:	Maximum number of records to retrieve.
	SrcTable	
	Type:	String
	Description:	Table name for newly created data table.

Class : CNXDataReader

Class : CNXDataReader			
Namespace:	System.Data.CONNX		
Description:	The CNXDataAdapter class handles the actual reading from the database and the data conversion from the native data source to .NET-managed types.		
Properties:	**FieldCount**		
	Return Type:	Int	
	Description:	The FieldCount property returns an int value containing the number of fields in the resultset.	
	IsClosed		
	Return Type:	Bool	
	Description:	This IsClosed property returns a Boolean value that is true if the CNXDataReader is open.	
	RecordsAffected		
	Return Type:	Int	
	Description:	The RecordsAffected property returns an Int value that is the number of records affected.	
Methods:	**Close**		
	Return Type:	Void	
	Description:	The Close method closes the CNXDataReader.	
	Parameters:	None	
	Dispose		
	Return Type:	Void	
	Description:	The Dispose method releases any resources held by the object.	
	Parameters:	Disposing	
		Type:	Bool
		Description:	Specifies if managed resources should also be disposed (true).
	GetBoolean		
	Return Type:	Bool	

Description:	The GetBoolean method retrieves data from the specified field index and casts it to a Boolean value.	
Parameters:	i	
	Type:	Int
	Description:	Value that corresponds to the position of the requested field.
GetByte		
Return Type:	Byte	
Description:	The GetByte method retrieves data at the specified index and casts it to a byte.	
Parameters:	I	
	Type:	Int
	Description:	Value that corresponds to the position of the requested field.
GetBytes		
Return Type:	Long	
Description:	The GetBytes method reads a number of bytes from the target field and fills the specified byte array with the data. The return value is the actual number of bytes read.	
Parameters:	**I**	
	Type:	Int
	Description:	Value that corresponds to the position of the requested field.
	FieldOffset	
	Type:	Long
	Description:	Position in the field to begin reading.
	Buffer	
	Type:	Byte[]
	Description:	Destination array for the data.
	BufferOffset	
	Type:	Int

		Description:	Starting position in the buffer to begin writing.
		Length	
		Type:	Int
		Description:	Number of characters to read.
	GetChar		
	Return Type:	Char	
	Description:	The GetChar method retrieves data from the specified index and casts it to a Char.	
	Parameters:	**I**	
		Type:	Int
		Description:	Value that corresponds to the position of the requested field.
	GetChars		
	Return Type:	Long	
	Description:	The GetChars method reads a number of characters from the target field and fills the specified character array with the data. The return value is the actual number of characters read. The GetChars method is not supported at this time. It is provided for interface compatibility.	
	Parameters:	I	
		Type:	Int
		Description:	Value that corresponds to the position of the requested field.
		FieldOffset	
		Type:	Long
		Description:	Position in the field to begin reading.
		Buffer	
		Type:	Char[]
		Description:	Destination array for the data.

		BufferOffset	
		Type:	Int
		Description:	Starting position in the buffer to begin writing.
		Length	
		Type:	Int
		Description:	Number of characters to read.
	GetData		
	Return Type:	IDataReader	
	Description:	The GetData method is not supported at this time.	
	Parameters:	I	
		Type:	Int
		Description:	Value that corresponds to the position of the requested field.
	GetDataTypeName		
	Return Type:	String	
	Description:	The GetDataTypeName method returns the datatype name for the requested field index.	
	Parameters:	I	
		Type:	Int
		Description:	Value that corresponds to the position of the requested field.
	GetDateTime		
	Return Type:	DateTime	
	Description:	The GetTime method retrieves the data at the specified index and casts it to a DateTime.	
	Parameters:	I	
		Type:	Int]
		Description:	Value that corresponds to the position of the requested field.
	GetDecimal		
	Return Type:	Decimal	

Description:	The GetDecimal method retrieves the data at the specified index and casts it to a Decimal.	
Parameters:	I	
	Type:	Int
	Description:	I Value that corresponds to the position of the requested field.
GetDouble		
Return Type:	Double	
Description:	The GetDouble method retrieves the data at the specified position and casts it to a Double.	
Parameters:	I	
	Type:	Int
	Visible:	Value that corresponds to the position of the requested field.
GetFieldType		
Return Type:	Type	
Description:	The GetFieldType method returns the Type for the requested field index.	
Parameters:	I	
	Type:	Int
	Description:	Value that corresponds to the position of the requested field.
GetFloat		
Return Type:	Float	
Description:	The GetFloat method retrieves the data at the specified position and casts it to a Float.	
Parameters:	I	
	Type:	Int
	Description:	Value that corresponds to the position of the requested field.
GetGuid		
Return Type:	Guid	

	Description:	The GetGuid method retrieves the data at the specified index and casts it to a GUID value.	
	Parameters	I	
		Type:	Int
		Description:	Value that corresponds to the position of the requested field.
	GetInt16		
	Return Type:	Int16	
	Description:	The GetInt16 method retrieves the data at the specified index and casts it to an INT16 value.	
	Parameters	I	
		Type:	Int
		Description:	Value that corresponds to the position of the requested field.
	GetInt32		
	Return Type:	Int32	
	Description:	The Get32 method retrieves the data at the specified field and casts it to an Int32.	
	Parameters:	I	
		Type:	Int
		Description:	Value that corresponds to the position of the requested field.
	GetInt64		
	Return Type:	Int64	
	Description:	The GetInt64 method retrieves the data at the specified field and casts it to an Int64.	
	Parameters:	I	
		Type:	Int
		Description:	Value that corresponds to the position of the requested field.
	GetName		
	ReturnType:	String	

	Description:	The GetName method returns the name of the requested field index.	
	Parameters:	I	
		Type:	Int
		Description:	Value that corresponds to the position of the requested field.
	GetOrdinal		
	Return Type:	Int	
	Description:	The GetOrdinal method returns the column position for the field with the specified name.	
	Parameters:	Name	
		Type:	String
		Description:	Value that corresponds to the name for which the ordinal value will be returned.
	GetSchemaTable		
	Return Type:	DataTable	
	Description:	The GetSchemaTable method requests the schema and returns the schema as a data table. This feature is not imlemented, but is provided for interface compatibility.	
	Parameters:	None	
	GetString		
	Return Type:	String	
	Description:	The GetString method retrieves the data contained in the specified field and casts it to a string.	
	Parameters:	I	
		Type:	int
		Description:	Value that corresponds to the position of the requested field.
	GetValue		
	Return Type:	Object	
	Description:	The GetValue method returns an object consisting of the data from the requested field.	
	Parameters:	I	

	Type:	Int
	Description:	Value that corresponds to the position of the requested field.
GetValues		
Return Type:	Int	
Description:	The GetValues method populates an object array with the data contained in the entire row. The return value is the number of fields populated.	
Parameters:	Values	
	Type:	Object[]
	Description:	Object array that is populated by the GetValues method.
IsDBNull		
Return Type:	Bool	
Description:	The IsDBNull method returns a Boolean value that is Trus if the value at the specified field is null.	
Parameters:	I	
	Type:	Int
	Description:	Value that corresponds to the position of the requested field.
NextResult		
Return Type:	Bool	
Description:	The NextResult method advances to the next Resultset when there is more than one resultset returned. This features is not supported and is provided for interface compatibility.	
Parameters:	None	
Read		
Return Type:	Bool	
Description:	The Read method advances the CNXDataReader to the next record in the resultset.	
Parameters:	None.	

Class : CNXDbType

Class : CNXDataAdapter	
Stereotype:	Enum
Namespace:	System.Data.CONNX
Description:	The CNXDbType enumeration contains all of the valid CONNX datatypes that are recognized by the CONNX .NET Data Provider.
Attributes:	BigInt
	Binary
	Bit
	Char
	Date
	Decimal
	Double
	Float
	GUID
	Integer
	IntervalDay
	IntervalDayToHour
	IntervalDayToMinute
	IntervalDatToSecond
	IntervalHour
	IntervalHourToMinute
	IntervalHourToSecond
	IntervalMinute
	IntervalMinuteToSecond
	IntervalMonth
	IntervalSecond
	IntervalYear
	IntervalYearToMonth
	LongVarBinary
	LongVarChar
	Numeric

	Real
	SmallInt
	Time
	Timestamp
	TinyInt
	Unknown
	UnsignedBigIng
	UnsignedInteger
	UnsignedSmallInt
	UnsignedTinyInt
	VarBinary
	VarChar

Class : CNXException

Class : CNXException		
Namespace:	System.Data	
Description:	The CNXException class is used for handling internal CONNX .NET Data Provider Exceptions.	
Properties:	**Code**	
	Return Type:	Int
	Description:	The Code property returns an Int value that corresponds to the error number.
	Errors	
	Return Type:	Array List
	Description:	The Errors property contains a collection of the CNXError objects.
	Message	
	Return Type:	String
	Description:	The Message property returns a string value that corresponds to the error message.

Class : CNXParameter

Class : CNXParameter			
Namespace:	System.Data.CONNX		
Description:	The CNXParameter class is used to provide dynamic parameters to an SQL statement.		
Constructors	**CNXParameter**		
	Description:	This default constructor creates an empty CNXParameter object.	
	Parameters:	None	
	CNXParameter		
	Description:	This constructor accepts a string value containing the parameter name and a CNXDbType value that contains the datatype for the parameter.	
	Parameters:	ParameterName	
		Type:	String
		Description:	Name of the parameter.
		Type	
		Type:	CNXDbType
		Description:	CONNX data type for the parameter.
	CNXParameter		
	Description:	This constructor accepts a string value containing the parameter name, a CNXDbTpe value that contains the data type for the parameter, and an Int value representing the length of the parameter data.	
	Parameters:	ParameterName	
		Type:	String
		Description:	Name of the parameter.
		Type	
		Type:	CNXDbType
		Description:	CONNX data type for the parameter.
		Length	

	Type:	Int
	Description:	Size of the parameter.
CNXParameter		
Description:	This constructor accepts a string value containing the parameter name, a CNXDbType value that contains the data type for the parameter, an Int value representing the length of the parameter data, and a string value that contains the name of the column to which the parameter is mapped.	
Parameters:	ParameterName	
	Type:	String
	Description:	Name of parameter.
	Type	
	Type:	CNXDbType
	Description:	CONNX data type for the parameter.
	Length	
	Type:	Int
	Description:	Size of the parameter.
	SourceColumn	
	Type:	String
	Description:	Name of the column to which the parameter is mapped.
CNXParameter		
Description:	This constructor accepts a string containing the parameter name and an object that contains the data for the parameter string value that represents the name of the parameter object that contains the data for the parameter.	
Parameters:	ParameterName	
	Type:	String
	Description:	Name of the parameter.
	Value	
	Type:	Object

		Description:	Object that contains the data for the parameter.
	CNXParameter		
	Description:	This constructor accepts a string value containing the parameter name, a CNXDbType value that contains the data type for the parameter, and Int value representing the length of the parameter data, and a string value that contains the name of the column to which the parameter is mapped.	
	Parameters:	ParameterName	
		Type:	String
		Description:	Name of the parameter.
		Type	
		Type:	CNXDbType
		Description:	CONNX data type for the parameter.
		SourceColumn	
		Type:	String
		Description:	Name of the column to which the parameter is mapped.
	CNXParameter		
	Description:	The constructor accepts a string value containing the parameter name, a CNXDbType value that contains the data type for the parameter, and a data stream object containing the data to be streamed to the data source.	
	Parameters:	ParameterName	
		Type:	String
		Description:	Name of the parameter.
		Type	
		Type:	CNXDbType
		Description:	CONNX data type for the parameter.
		DataStream	
		Type:	System.IO.Stream
		Description:	Stream object that contains the parameter data.
Properties:	**CNXType**		

Return Type:	CNXDbType
Description:	The CNXType property specifies the native CONNX data type for the parameter.
CType	
Return Type:	Int16
Description:	The CType property contains the native C data type for the parameter.
DbType	
Return Type:	DbType
Description:	The DbType property contains the data type for the parameter.
Direction	
Return Type:	ParameterDirection
Description:	The Direction property specifies the type of the parameter, whether it is input only, a return value, etc.
IsNullable	
Return Type:	Boolean
Description:	The IsNullable property returns a Boolean value if null values are acceptable in the parameter.
ODBCType	
Return Type:	Int16
Description:	The ODBCType property contains the native ODBC data type for the parameter.
ParameterName	
Return Type:	String
Description:	The Parameter name property is used to specify the name of the parameter.
Precision	
Return Type:	Byte
Description:	The Precision property is used to specify the number of decimal places to retain for the given parameter.
Scale	
Return Type:	Byte
Description:	The Scale property specifies the default number of decimal places for the parameter date.
Size	
Return Type:	Int
Description:	The Size property specifies the length of the parameter data.

	SourceColumn		
	Return Type:	String	
	Description:	The SourceColumn property is used for the name of the source column that is mapped to the data set. This feature is currently not implemented, but is provided for interface compatibility.	
	SourceVersion		
	Return Type:	DataRowVersion	
	Description:	The SourceVersion property is used to determine if the original or current data is used in the parameter. This feature is currently not implemented, but is provided for interface compatibility.	
	Stream		
	Return Type:	System.IO.Stream	
	Description:	The Stream property accepts a Stream object that can be used to stream Blob and Clob parameter data to the data source.	
	Value		
	Return Type:	Object	
	Description:	The Value property is used to store the data for the parameter.	
Methods	**GetDBType**		
	Return Type:	DBType	
	Description:	The GetDBType method returns the DBType of the specified object.	
	Parameters:	Value	
		Type:	Object
		Description:	Object for which the type will be retrieved.

Class : CNXParameterCollection

Class : CNXParameterCollection			
Namespace:	System.Data.CONNX		
Description:	The CNXParameterCollection is used for holding the parameters for a given CNXCommand object.		
Properties:	**Count**		
	Return Type:	Int	
	Description:	The Count property returns the number of CNXParameter objects in the collection.	
Methods:	**Add**		
	Return Type:	Int	
	Description:	The Add method adds the specified object to the collection. The return value is an Int value representing the index of the item in the collection.	
	Parameters:	Value	
		Type:	Object
		Description:	Object to be added to the collection.
	Add		
	Return Type:	Int	
	Description:	The Add method adds the specified CNXParameter object to the collection. The return value is an Int value representing the index of the item in the collection.	
	Parameters:	Value	
		Type:	CNXParameter
		Description:	CNXParameter object to be added to the collection.
	Add		
	Return Type:	Int	
	Description:	The Add methods adds a new CNXParameter object to the collection with the specified parameterName and value. The return value is an Int value representing the index of the new CNXParameter object in the collection.	
	Parameters:	ParameterName	

		Type:	String
		Description:	Name of the parameter.
		Value	
		Type:	Object
		Description:	Object containing the data for the new parameter.
	Add		
	Return Type:	Int	
	Description:	The Add method adds a new CNXParameter object to the collection with the specified ParameterName and value. The return value is an Int value representing the index of the new CNXParameter object in the collection.	
	Parameters:	ParameterName	
		Type:	String
		Description:	Name of parameter.
		dbType	
		Type:	CNXDbType
		Description:	CONNX data type for the parameter.
		SourceColumn	
		Type:	String
		Description:	Name of the target column for the parameter.
	Add		
	Return Type:	Int	
	Description:	The Add method adds a new CNXParameter object to the collection with the specified ParameterName and value. The return value is an Int value representing the index of the new CNXParameter object in the collection.	
	Parameters:	ParameterName	
		Type:	String
		Description:	Name of the parameter.
		DbType	

		Type:	CNXDbType
		Description:	CONNX data type for the parameter.
		Length	
		Type:	Int
		Description:	Object that contains the data for the parameter.
Add			
Return Type:	Int		
Description:	The Add method adds a new CNXParameter object to the collection with the specified ParameterName and value. the return value is an Int value representing the index of the new CNXParameter object in the collection.		
Parameters:	ParameterName		
		Type:	String
		Description:	Name of the parameter.
		DbType	
		Type:	CNXDbType
		Description:	CONNX data type for the parameter.
		Length	
		Type:	Int
		Description:	Length of the parameter.
		SourceColumn	
		Type:	String
		Description:	Name of the target column for the parameter.
CNXParameter			
Return Type:	Void		
Description:	The Clear method removes all of the items from the collection.		
Parameters:	None		
Contains			
Return Type:	Bool		

Description:	The Contains method returns a Boolean value based on whether the specified parameter name exists in the collection.	
Parameters:	ParameterName	
	Type:	String
	Description:	Name of parameters to find.
Contains		
Return Type:	Bool	
Description:	The Contains method returns a Boolean value denoting whether the specified object is contained in the collection.	
Parameters:	Value	
	Type:	Object
	Description:	Object to search for.
CopyTo		
Return Type:	Void	
Description:	The CopyTo method enables users to copy elements of the collection to a specified array.	
Parameters:	Array	
	Type:	Array
	Description:	Destination for the copied elements.
	Index	
	Type:	Int
	Description:	Starting position in the array.
GetEnumerator		
Return Type:	IEnumerator	
Description:	The GetEnumerator method returns an IEnumerator interface that is used to iterate through the items in the collection.	
Parameters:	None	
IndexOf		
Return Type:	Int	
Description:	The IndexOf method returns an Int value representing the index of the parameter specified by the parameter name.	
Parameters:	ParameterName	

	Type:	String
	Description:	Name of the parameter for which to retrieve an index.
IndexOf		
Return Type:	Int	
Description:	The IndexOf method is used to retrieve the index of the specified object.	
Parameters:	Value	
	Type:	Object
	Description:	Object for which the index is retrieved.
Insert		
Return Type:	Void	
Description:	The Insert method enables users to insert an object into the collection at the specified index.	
Parameters:	Index	
	Type:	Int
	Description:	Index at which to insert the object.
	Value	
	Type:	Object
	Description:	Object to be inserted.
Remove		
Return Type:	Byte	
Description:	The Precision property is used to specify the number of decimal places to retain for the given parameter.	
Parameters:	Value	
	Type:	Object
	Description:	Object to be removed from the collection.
RemoveAt		
Return Type:	Byte	
Description:	The RemoveAt method removes the parameter specified by the parameter name.	
Parameters:	ParameterName	

		Type:	String
		Description:	Name of the parameter to be removed.
	RemoveAt		
	Return Type:	Int	
	Description:	The RemoveAt method removes the parameter at the specified index.	
	Parameters:	I	
		Type:	Int
		Description:	Index of parameter to be removed.

Class : CNXTransaction

Class : CNXTransaction			
Namespace:	System.Data.CONNX		
Description:	The CNXTransaction class encapsulates a data transaction, enabling unsuccessful updates and inserts to be undone.		
Properties:	**IsolationLevel**		
	Return Type:	IsolationLevel	
	Description:	The IsolationLevel property gets/sets the isolation level of the transaction. Currently, this method does not have any effect, but is provided for interface compatibility.	
Methods:	**Commit**		
	Return Type:	Void	
	Description:	The Commit method causes all changes to be committed to the data source. Once committed, changes cannot be undone.	
	Parameters:	None	
	Connection		
	Return Type:	IDbConnection	
	Description:	The Connection property gets/sets the CNXConnection for this transaction.	
	Dispose		
	Return Type:	Void	
	Description:	The Dispose method releases any resources held by the object.	
	Parameters:	None.	
	Dispose		
	Return Type:	Void	
	Description:	The Dispose method releases any resources held by the object.	
	Parameters:	Disposing	
		Type:	Bool
		Description:	Specifies whether manages resources should also be

			disposed of (true).
	Rollback		
	Return Type:	Void	
	Description:	The Rollback method causes all changes to be discarded, reverting the data source to its previous state.	
	Parameters:	None.	

Class : CNXTransactionCapabilities

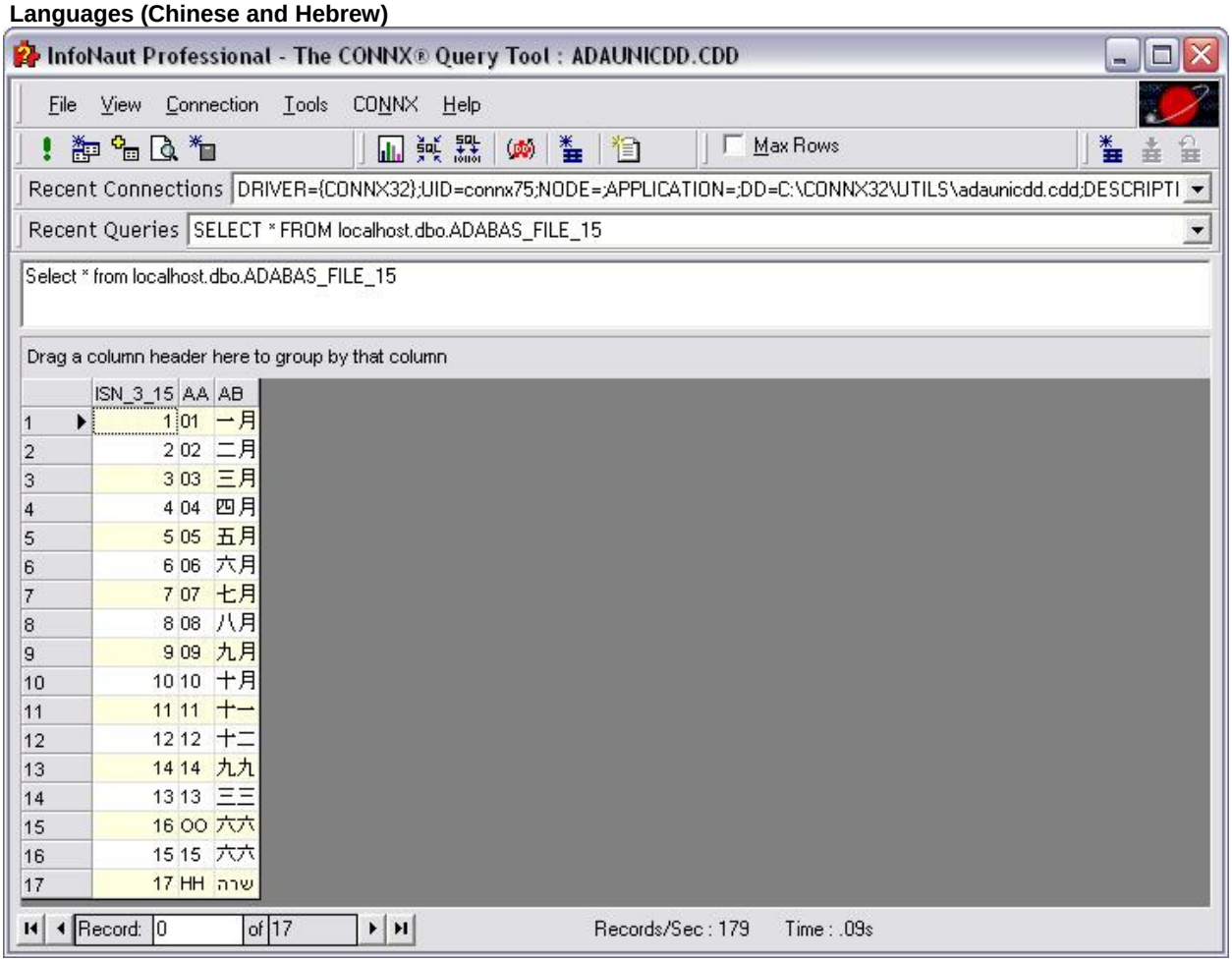
Class : CNXTransactionCapabilities		
Namespace:	System.Data.CONNX	
Description:	The CNXTransactionCapabilities enumeration contains the possible Transaction Capabilities that will be reported.	
Attributes:	**None**	
	Default Value:	0
	Description:	No registered attributes means that the connection has no transaction capabilities.
	DML	
	Default Value:	1
	Description:	DML means the connection can handle standard DML calls (Select, Update, Insert, Delete), but DDL statements generate errors.
	All	
	Default Value:	2
	Description:	All means the connection can handle all DML and DDL statements inside a transaction.
	DDL_Commit	
	Default Value:	3
	Description:	DDL_Commit is the same as DML, except that any DDL statements cause the transaction to be committed.
	DDL_Ignore	
	Default Value:	4
	Description:	DDL_Ignore is the same as DML, except that any DDL statements are ignored.

Unicode support

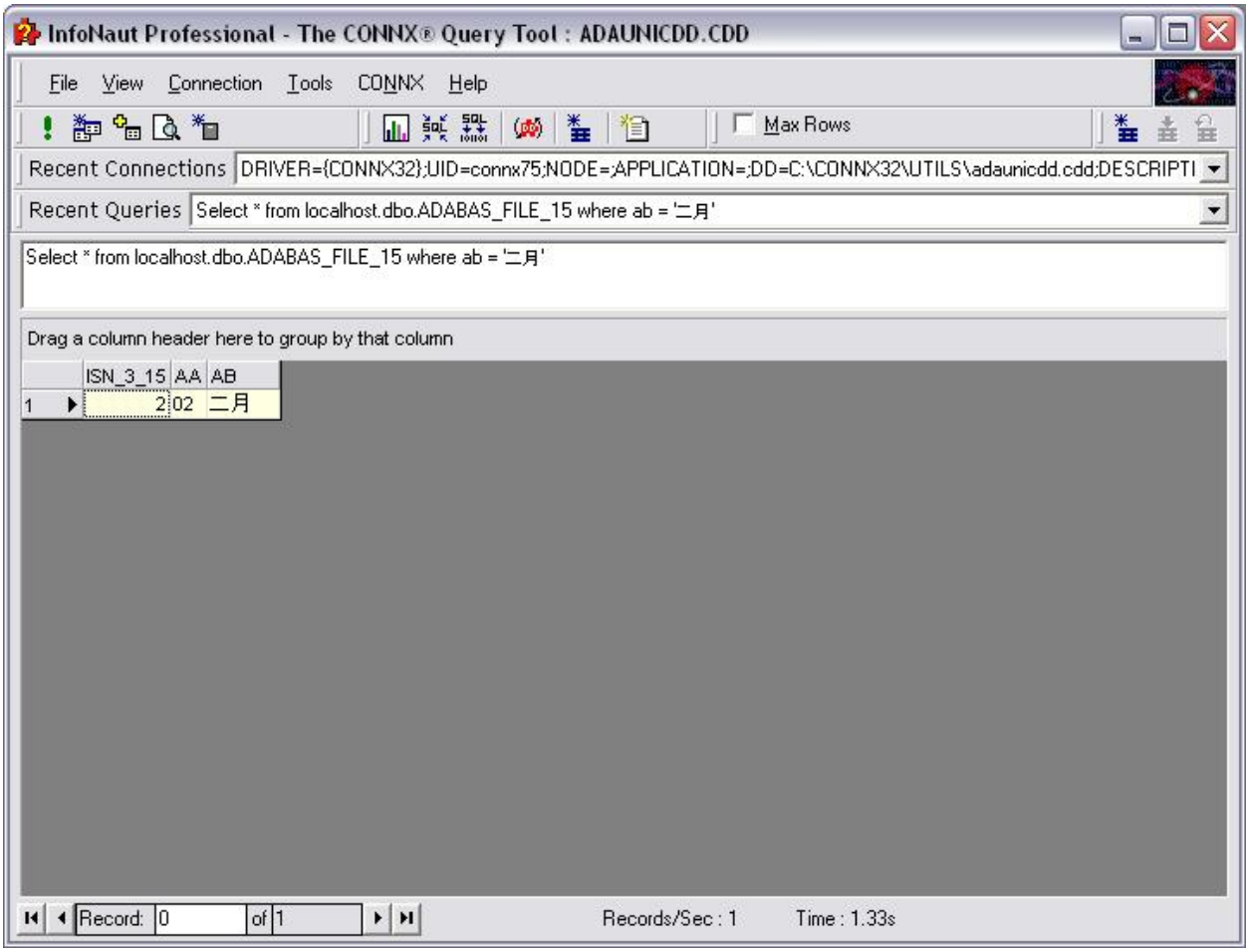
CONNX supports Unicode and ANSI data types, which will enable users to import tables other than those containing only English information into the CONNX Data Dictionary Manager.

The following example shows data imported from an Adabas database containing Unicode data. Unicode support is transparent and does not require any special manipulation or translation of the native data. The data will import in its native format.

CONNX Unicode Support with SELECT SQL Statement Displaying Data Containing Rows with Two Different Languages (Chinese and Hebrew)



CONNX Unicode Support with SELECT SQL Statement Displaying a Dynamic SQL Statement Containing Unicode Characters and the Resultant Output


InfoNaut Professional - The CONNX® Query Tool : ADAUNICDD.CDD
File View Connection Tools CONNX Help
Max Rows
Recent Connections
DRIVER={CONNX32};UID=connx75;NODE=;APPLICATION=;DD=C:\CONNX32\UTILS\adaunicdd.cdd;DESCRIPTI
Recent Queries
Select * from localhost.dbo.ADABAS_FILE_15 where ab = '二月'
Select * from localhost.dbo.ADABAS_FILE_15 where ab = '二月'
Drag a column header here to group by that column
ISN_3_15 AA AB
1 2 02 二月
Record: 0 of 1
Records/Sec : 1
Time : 1.33s

CONNX OLE RPC Server

The CONNX OLE RPC server enables RPC (remote procedure calls) from any programming language that supports OLE 2.0, including Microsoft Word, Microsoft Excel, Microsoft Visual Basic, Microsoft Access, and Paradox for Windows. The server is supported by VSAM, RMS, Oracle Rdb, and DBMS databases. The CONNX OLE server object is called CONNX.Connect.

In Visual Basic, the following code can be used to connect to this server object:

```
Dim gOleConnect as Object
Set gOleConnect = CreateObject("CONNX.Connect")
```

Related Topics

CONNX Remote Procedures
To build a remote procedure
Execution of OS/400 Remote Commands via CONNX and DB2
To execute OS/400 remote commands via CONNX

CONNX.Connect

The following table is a detailed description of the properties and methods used in the CONNX OLE server object.

Properties and Methods of CONNX OLE Server Object

Property	Property Type	Description
Node	String Read/Write	The node to which CONNX attempts to connect. Node name must be set before using the Connect method.
UserName	String Read/Write	VAX account name used to connect to the VAX. UserName must be set before using the Connect method.
Password	String Read/Write	Accompanying password for the UserName property. Password must be set before using the Connect method.
LastResultCode	String Read only	Contains the result code for the last method called.
LastErrorMessage	String Read only	Contains the text of the translated error code for the last method called.
InputValue	Long Write only	Contains the input string for the RPC method.
InputValueLength	String Write only	Contains the length of the input string for the RPC method.
OutputValue	String Read only	Contains the output string for the RPC method.
OutputValueLength	Long Read/Write only	Contains the maximum length of the output string for the RPC method.

The following table includes descriptions of the connect methods used with the OLE RPC server.

Connection Methods for the OLE RPC Server

Method	Description
Connect	Uses the stored properties UserName, Password, and Node, to connect to the VAX. The result code of the connection attempt is stored in LastResultCode.
Disconnect	Closes the RPC connection to the VMS server. This is the last method called to the CONNX OLE object.
RPC (RPCName as a string)	The RPC method uses the stored properties InputValue, InputValueLength, and OutputValueLength to execute the RPC specified in the parameter RPCName. The result code of the connection attempt is stored in LastResultCode. The result buffer is returned in OutputValue. To execute a DCL command under VMS, the RPC name must begin with ""$."" RPCs can consist of more than one command. Each command must be separated by a newline character, each line must begin with ""$."" Example: The following example performs a directory and then shows the current VMS time.

	gOleObject.RPC ""DIR/nSHOW TIME""

CONNX Remote Procedures

A CONNX Remote Procedure is a VMS program or function with which the client computer communicates. The CONNX server, running on the VMS system, passes messages, known as remote procedure calls (RPC), back and forth between the computer application and the VMS-based remote procedure. Since applications can vary widely in functionality, the CONNX server is able to pass the messages without regard to content or format. It is up to the application programmer to ensure both the computer and VMS programs are working within the same message format and structure.

CONNX comes with two sample RPC stubs: RPC_BAS, written in VAX BASIC, and RPC_CPP.RPC, written in DEC C++. They can be found in the directory into which CONNX is installed on the VMS system and are described in the following table:

C syntax example:

```
long rpc (char * lpszRPCName, int nSendLength, char * lpszSendBuff, int
 nReceiveLength, char *  lpszReceiveBuff, int nLog)
```

C Syntax Example Parameters and Description

Parameter	Description
lpszRCPName	Name of the VAX procedure with which to communicate. Maximum length is 16 characters.
NSendLength	Length of lpszSendBuff.
lpszSendBuff	Message to be transmitted to the VAX>
NReceiveLength	Length of lpszReceiveBuff.
lpszReceiveBuff	Message received from the VAX. This butter must be one byte longer than nReceiveLength.
NLog	Flag that determines if detailed logging message should be sent to standard output (printf).

RPC is a function that returns a long integer representing the status of the function as its value. Unless otherwise altered within the RPC, a success status of 1 is assumed. This status is passed back to the computer as the value of the computer's RPC function. While any value may be assigned, note that only actual VMS status codes may be translated via the vmsstatus function. Consequently, it is recommended that VMS status codes be assigned as the value of the RPC function.

A remote procedure is invoked by the computer via the RPC function available in the CONNX OLE Server.

Related Topic

To build a remote procedure

To build a remote procedure

The following steps are used to install an RPC on the VMS system. (If you are using RMS on Itanium, the options file is the same as that for Alpha users: RMSALPH.OPT.)

1. Determine the appropriate object library

   For RMS, the name is **CNXRMSC0.OLB**

   For DBMS, the name is **CNXDBM0.OLB**

   For Rdb, the name is **CNXRDBC0.OLB**

   Note: The '0' at the end of the name is a zero and not the letter O

2. Make a backup copy of the original object library, for example:

$ COPY CNXRDBC0.OLB CNXRDB_ORIG.OLB

3. Use the appropriate compiler to create an object file, for example:

$ BASIC RPC_

4. Create a new executable:

For RMS, the link command is as follows:

```
$ LINK/EXE=CNXRMSC0.EXE RPC_.obj CNXRMSC0.OLB/LIB/INCLUDE=(FACTORY),
RMSVAX.OPT/OPT
```

(For Alpha systems, the options file is called **RMSALPH.OPT**)

For Rdb, the link command is as follows:

```
$ LINK/EXE=CNXRDBC0.EXE RPC_.obj CNXRDBC0.OLB/LIB /INCLUDE= (FACTORY) ,
RDBVAX.OPT/OPT
```

(For Alpha systems, the options file is called **RDBALPH.OPT**)

For DBMS, the link command is as follows:

```
$ LINK/EXE=CNXDBMC0.EXE RPC_.obj CNXDBMC0.OLB/LIB /INCLUDE= (FACTORY) ,
DBMVAX.OPT/OPT
```

(For Alpha systems, the options file is called **DBMALPH.OPT**)

If the CONNX server was installed as a shared image, replace the image with the new version of the executable. Refer to the appropriate VMS manual for instructions on using the INSTALL utility.

If you do not have a license to use the VAX BASIC compiler, RPC.BAS can be rewritten in any language available. Descriptor passes the string parameters, while the integer parameters are passed by reference.

Execution of OS/400 Remote Commands via CONNX and DB2

CONNX enables submission of OS/400 commands from a client computer via the ODBC SQLExecDirect API or the SQLPrepare/SQLExecute API. The CONNX parsing logic scans the text input string and detects whether it is an OS/400 remote command or an SQL statement, and forwards the string to the host.

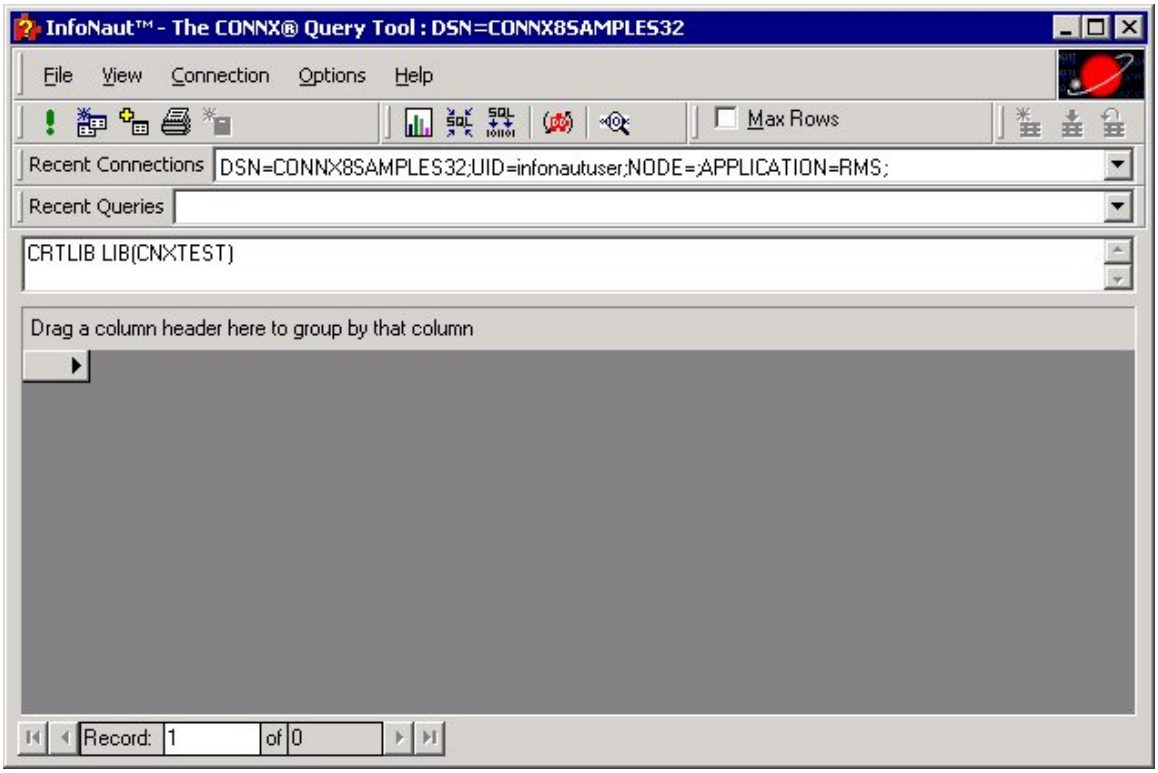
Although CONNX recognizes OS/400 remote commands based on standard three-character prefixes (WRK, DSP, CRT, DLT, etc.) and forwards these command strings to the host, the host remote command processor does not permit all OS/400 CL commands to be executed from a remote client.

For example, commands such as WRKSPLF can be submitted via CONNX, but the host will reject them because they require additional interaction via a 5250-based emulator product.

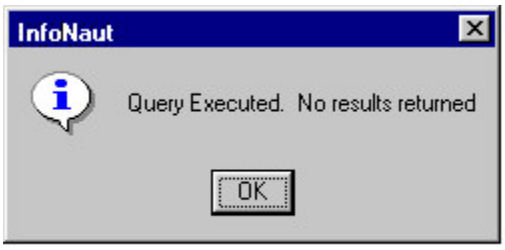
To execute OS/400 remote commands via CONNX

You can use any ODBC-compliant application to execute OS/400 remote commands via CONNX, provided that the application supports the SQLExecDirect or SQLPrepare/SQLExecute API call sequence. The application used in the following examples is InfoNaut™, but any ODBC-compliant application may be used.

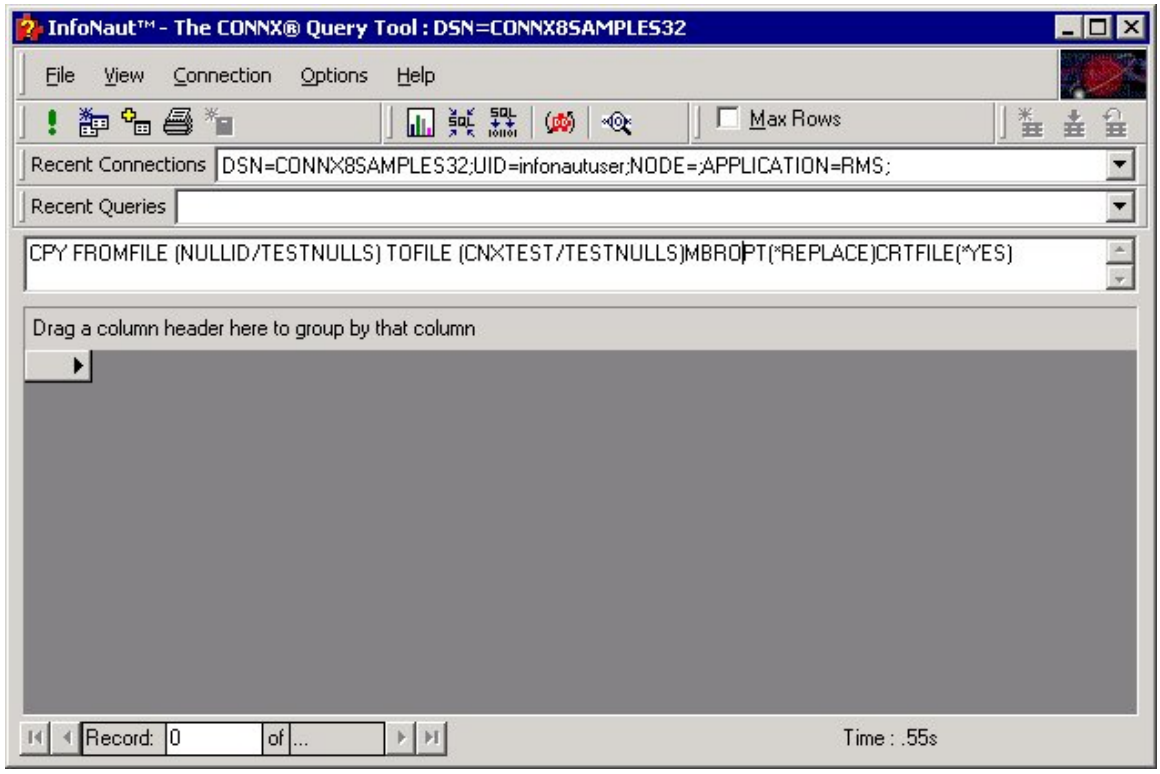
1. Create a library = **CNXTEST.**



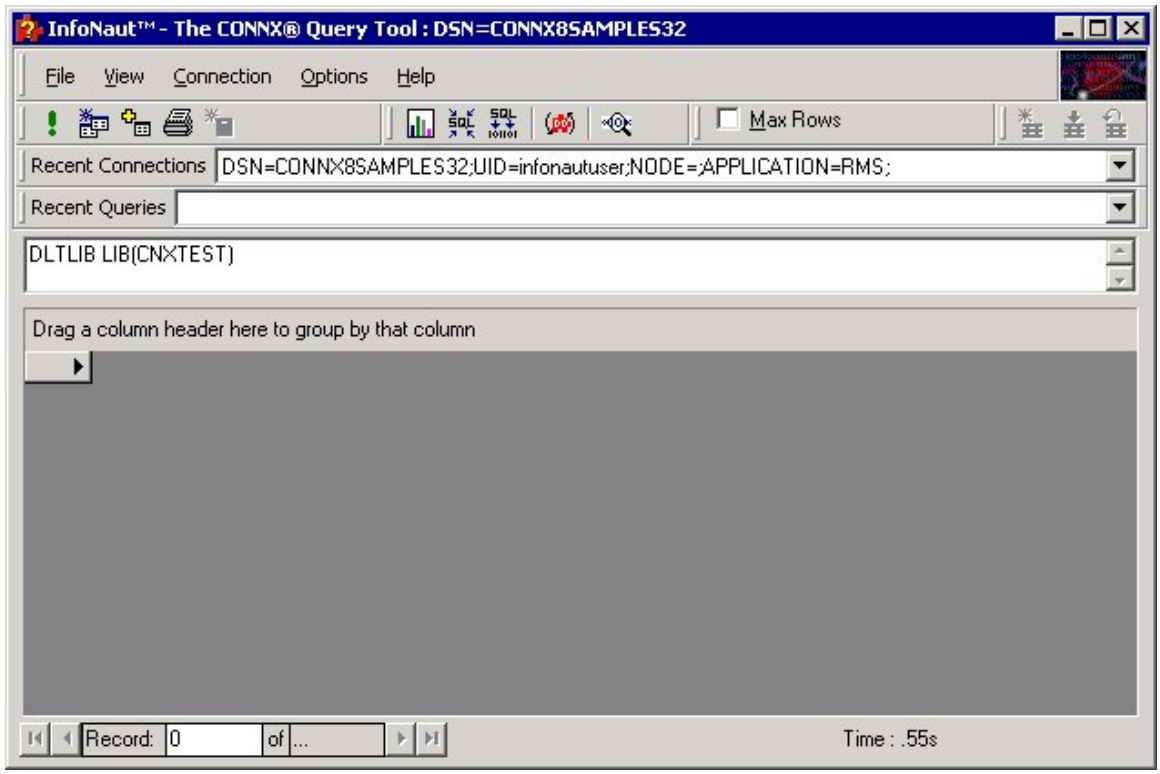
If the remote command succeeds, the InfoNaut application displays the following dialog box:



2. Copy the file **NULLID/TESTNULLS** to **CNXTEST/TESTNULLS.**

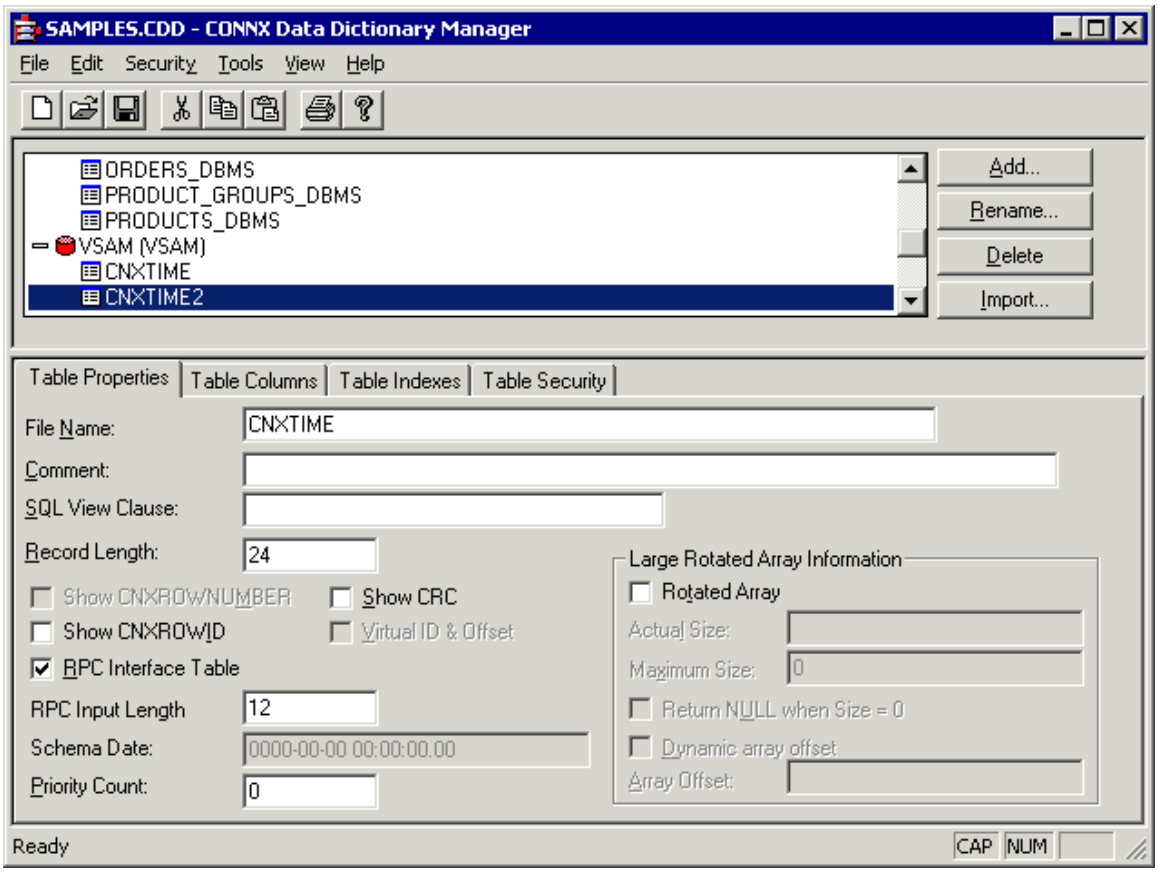


3. Delete the library **CNXTEST.**



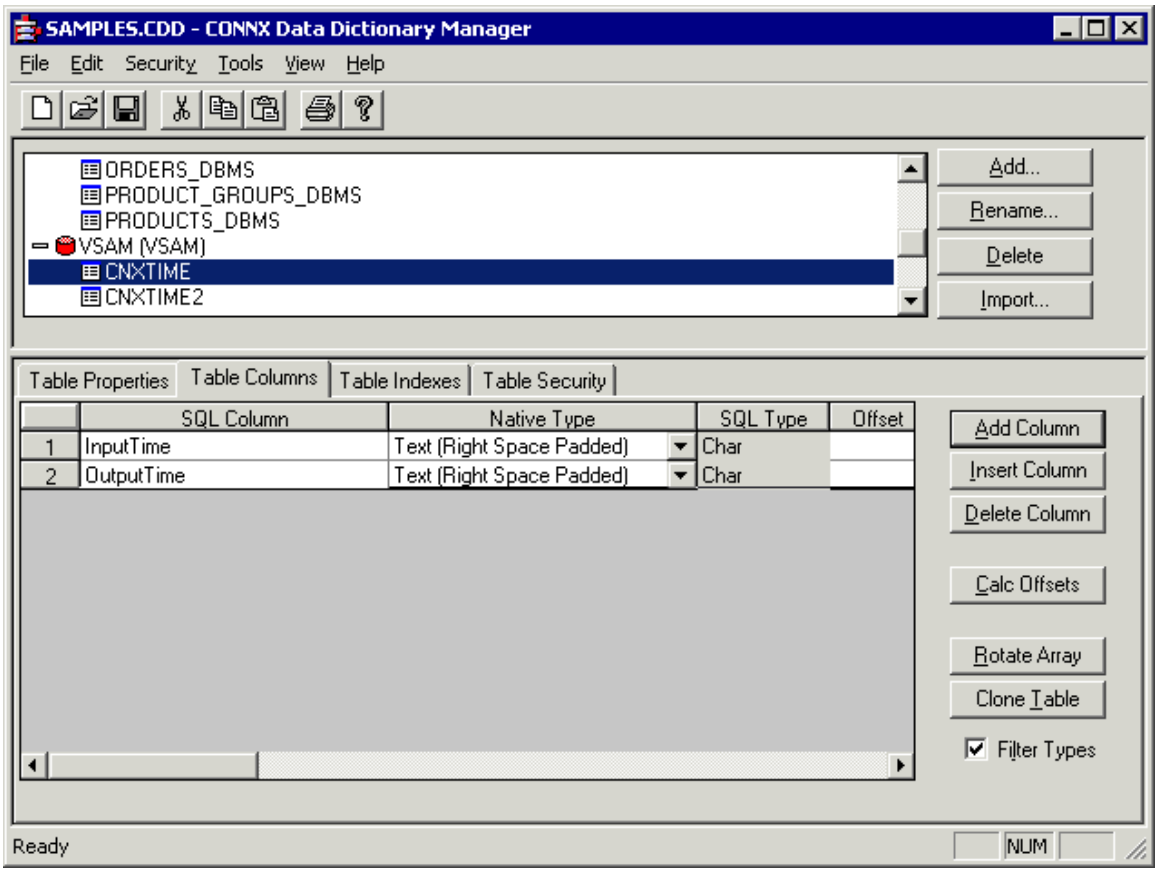
Executing a Remote Procedure Call via SQL

1. Select the **RPC Interface Table** check box.
2. Specify the name of the program to execute in the **File Name** field.
1. For VSAM running under CICS, specify the name of the CICS program to execute
2. For VSAM running as started task, specify the name of the TSO program to execute.
3. For VMS, specify the name of the DCL program to execute (prefixed by a $), or the name of the VMS RPC to execute (see the VMS RPC documentation)



3. Under **Table Columns** create a set of columns to map over the input and output buffers for the RPC.

1. For VSAM running under CICS, specify the a set of columns to map over the COMMAREA on input, and another set of columns to map over the COMMAREA on output.



4. Under **Table Properties,** specify the total number of bytes for RPC input.

5. Finally, specify the total number of bytes for the input and output as the Record Length.

6. To execute the RPC, issue a select statement against the CONNX SQL Table that holds the RPC information. For example, the following statement will execute the RPC, and return the output columns.

```
Select * from CNXTime
```

7. To specify data for the input columns, supply where criteria for them in the SQL Statement. For example, the following statement will execute the RPC, and return the output columns.

```
Select * from CNXTime where inputtime = '12:33:01'
```

Chapter 11 - SQL Grammar

SQL Grammar

SQL Queries are built using SQL language elements made up of SQL Grammar. This chapter covers the SQL Grammar supported by CONNX.

Common SQL Grammar Elements

What are Common SQL Grammar Elements?

Common elements in SQL Grammar include digits, characters and data types. SQL is composed of a series of lexical units. Each lexical unit consists of a series of characters and/or digits. The precise syntax of these digits, lexical units and tokens are described in the following sections.

Character

A character is one unit of a set of symbols, letters and numbers, collectively called a character set. Common character sets are ASCII (typically used on PCs), and EBCDIC (typically used on mainframes). Characters can either be single byte, or Unicode (two bytes). See discussion of "CHAR vs. NCHAR" under "Data Types".

Examples of characters:

Character	Type
f	single byte ASCII character
5	single byte ASCII character
Q	single byte ASCII character
#	single byte ASCII character
말	multi-byte Unicode character

Comments

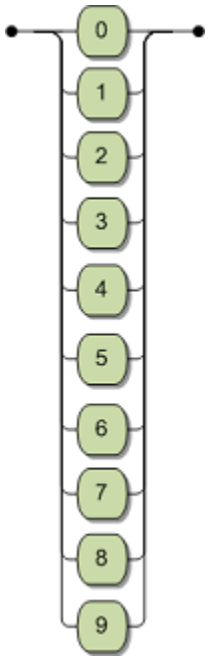
Comments can be inserted within the SQL. Precede the comment with a backslash (/) followed by an asterisk (*). Type the comment. When finished, follow the comment with an asterisk, followed by a backslash. The comment does not execute when the SQL statement is run, but remains within the statement.

Example:

```
UPDATE customers set customername = "After Merger" WHERE customerid = "ALCAO"
/* Per customer request. */
```

Digit

A digit is a single numeral between zero and nine. This is an example of the digit's syntax:



Data Types

The smallest unit of data is a value. A value can result from several different origins: a column, a constant, a function, an expression, a host variable, or a sub-query. CONNX interprets data based on its data type.

CONNX supports the following SQL data types:

Data Type	Maximum Precision or Length	Maximum Scale
Char	32000(*1)	N/A
Varchar	32000(*1)	N/A
Longvarchar	2 gigabytes	N/A
Nchar	32000(*1)	N/A
Varnchar	32000(*1)	N/A
Longvarnchar	2 gigabytes	N/A
Binary	32000(*1)	
Varbinary	32000(*1)	
Longvarbinary	2 gigabytes	
Decimal	100(*1)	100(*1)(*2)
Numeric	100(*1)	100(*1)(*2)
Bit	1	N/A
Tinyint	3	N/A
Smallint	5	N/A
Integer	10	N/A
Bigint	20	N/A
Date	N/A	N/A
Time	N/A	N/A
Timestamp	N/A	10(*3)
Real	7	N/A
Double	15	N/A
Float	7 or 15 (*5)	N/A
Qfloat(*4)	100	N/A

(*1) The maximum precision or scale of the data source may be smaller that the maximum allowed by CONNX.

(*2) Scale must be less than or equal to precision.

(*3) Scale for timestamp refers to the number of digits allowed in the fractional component of the "seconds field". For example, a scale for 3 would permit a fractional component from .000 to .999 in the timestamp.

(*4) CONNX also supports Qfloat, an internal-only numeric data type. Qfloat is high precision (352bits) and used to perform many internal calculations. Some constants or mathematical expressions may be stored and processed in the Qfloat format, and then downgraded to the requested SQL data type at retrieval or execution time.

(*5) For Adabas, the Float data type will get remapped to either Double, or Real, depending on the precision supplied during create. If no precision is supplied, or the precision is greater than or equal to 21, then Double is used, otherwise, Real is used.

CONNX maps these SQL types to the most appropriate Native type for each data adapter.

All data types can be set to SQL NULL, which is not the same as a string of zero length or a numeric 0.

Variable-Length Character-String Data Types

SQL Data types VARCHAR, LONGVARCHAR, VARNCHAR, and LONGVARNCHAR are variable-length character-strings. These data types contain character sequences; the character-string length is determined by either the data type definition or from the value itself. The maximum length of the data type is derived from the definition of the data type; if the value originates from a column which has been defined as a variable-length character-strings with length 15, the maximum length is always 15, but the length of a particular value may be between zero and 15.

Fixed-Length Character String Data Types

SQL Data types CHAR and NCHAR are fixed-length character strings. The length of a value with the data type fixed length character-string is determined by the definition of the origin of the value; if the value originates from a column which has been defined as a fixed length character-string with length 15, a value originating from this column will always have a length of 15.

CHAR vs. NCHAR

CONNX supports two categories of character data types: CHAR (single byte), and NCHAR (multibyte/UNICODE).

CHAR: Single byte character types use a single "byte" (a value ranging from 0 to 255) to represent each character in the string. With Latin based alphabets for example, the CHAR data type is sufficiently large enough to represent all the characters in the alphabet.

NCHAR: For non-Latin languages such as Chinese and Japanese, there are more than 256 characters in the alphabet. The NCHAR data type, where every character is stored in two bytes (a value ranging from 0 to 65535) is used to represent all the characters in these alphabets. This is also referred to as utf-16, or UNICODE.

Numeric Data Types

Numeric data types are used to specify the representation form of numeric values. CONNX supports four different representation forms of numeric values:

Each numeric value's form of representation has a precision and a sign.

In addition, the decimal forms of representation have a scale. The scale of a numeric value is defined as the number of digits in the fractional part of the number. The scale cannot be larger than the precision nor can it be negative.

Data Type	Description
Bit	Specifies a binary representation of a numeric value with a precision of one bit. The value range of a bit number is 0 to 1.

Tiny Integer	Specifies a binary representation of a numeric value with a precision of seven bits. The value range of a tiny integer number is -128 to +127.
Small Integer	Specifies a binary representation of a numeric value with a precision of 15 bits. The value range of a small integer number is -32768 to +32767.
Integer	Specifies a binary representation of a numeric value with a precision of 31 bits. The value range of an integer number is -2147483648 to +2147483647.
Bigint	Specifies a binary representation of a numeric value with a precision of 63 bits. The value range of an integer number is -9223372036854775808 to +9223372036854775807.
Single Precision Floating Point	Specifies a floating point representation with single precision. The value range of a single precision floating point number depends on the hardware platform.
Double Precision Floating Point	Specifies a floating point representation with double precision. The value range of a double precision floating point number depends on the hardware platform.
Numeric	Specifies an unpacked decimal representation with a user- specified scale and precision. The range of the precision is between 1 - 100. The range of the scale is zero to the value of the precision.
Decimal	Specifies a packed decimal representation with a user- specified scale and precision. The range of the precision is between 1 - 100. The range of the scale is zero to the value of the precision.

The maximum precision/scale of a particular datasource may be much smaller than the CONNX maximum.

Binary Data Type

CONNX supports BINARY, VARBINARY and LONGVARBINARY binary data types. The data stored within the binary data type is not byte swapped nor interpreted in any way.

For Adabas, the binary data type behavior deviates from the ANSI standard. The Adabas binary data type is an unsigned integer and is subject to byte swapping where appropriate. Application programs can use multiple interpretations when accessing the Adabas binary data type; in most cases the Adabas binary data type will be interpreted as a bit pattern. Adabas Binary values have a maximum length of 126 bytes. The maximum possible number of bits is 1008.

Data Type Conversion

CONNX is capable of converting a value of a certain data type to another data type.

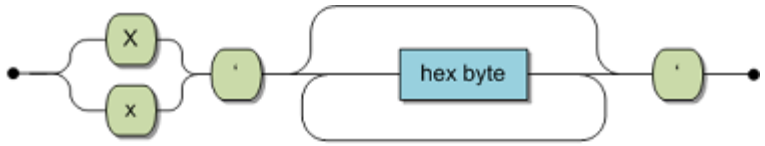
Data Conversion Matrix	char	varchar	longvarchar	nchar	varnchar	longvarnchar	binary	varbinary	longvarbinary	date	time	timestamp	bit	tinyint	smallint	integer	bigint	decimal	numeric	real	float	double
char	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
varchar	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
longvarchar	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
nchar	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
varnchar	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
longvarnchar	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
binary	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
varbinary	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
longvarbinary	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
date	x	x	x	x	x	x	x	x	x	x	x	x										
time	x	x	x	x	x	x	x	x	x	x	x	x										
timestamp	x	x	x	x	x	x	x	x	x	x	x	x										
bit	x	x	x	x	x	x	x	x	x				x	x	x	x	x	x	x	x	x	x
tinyint	x	x	x	x	x	x	x	x	x				x	x	x	x	x	x	x	x	x	x
smallint	x	x	x	x	x	x	x	x	x				x	x	x	x	x	x	x	x	x	x
integer	x	x	x	x	x	x	x	x	x				x	x	x	x	x	x	x	x	x	x
bigint	x	x	x	x	x	x	x	x	x				x	x	x	x	x	x	x	x	x	x
decimal	x	x	x	x	x	x	x	x	x				x	x	x	x	x	x	x	x	x	x
numeric	x	x	x	x	x	x	x	x	x				x	x	x	x	x	x	x	x	x	x
real	x	x	x	x	x	x	x	x	x				x	x	x	x	x	x	x	x	x	x
float	x	x	x	x	x	x	x	x	x				x	x	x	x	x	x	x	x	x	x
double	x	x	x	x	x	x	x	x	x				x	x	x	x	x	x	x	x	x	x

Precision or length may be lost if the data type is converted into a type with smaller precision/length.

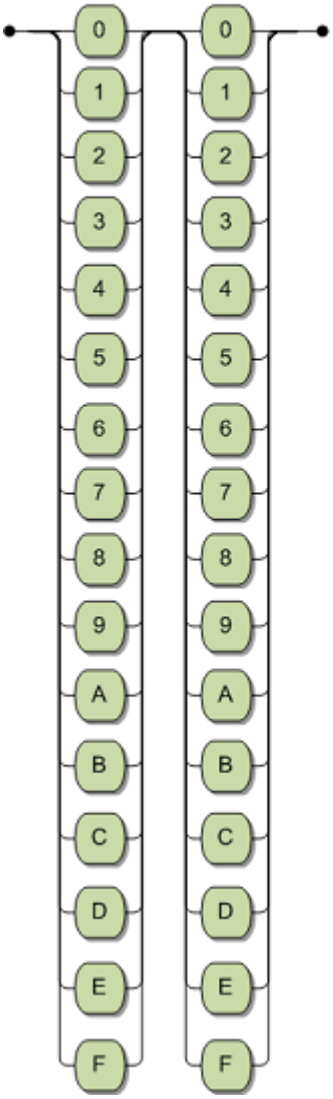
SQL Literals

Binary Literals

Syntax

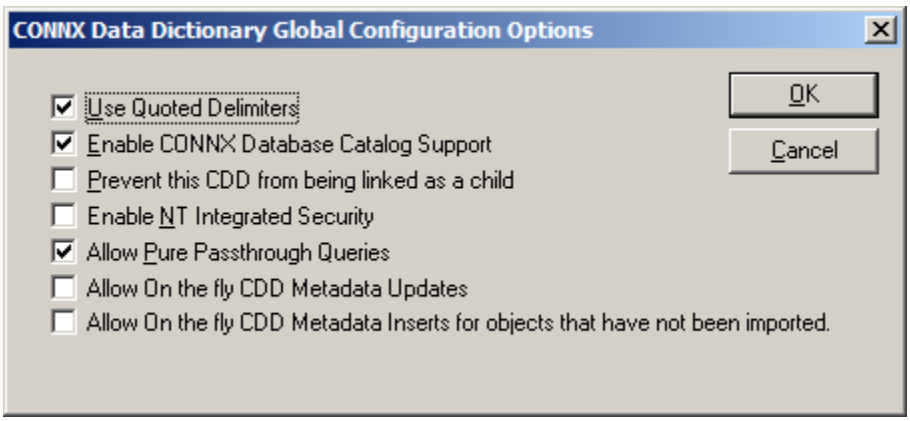


Hex Byte:



- Binary literals use hexadecimal bytes to represent the binary data.
- Binary literals must be prefixed by a capital or lower case X followed by a single quote character, and suffixed by a single quote character.

If the "Use Quoted Delimiters" option is not selected in the data dictionary, then character literals can also be prefixed and suffixed by a single double quote character.



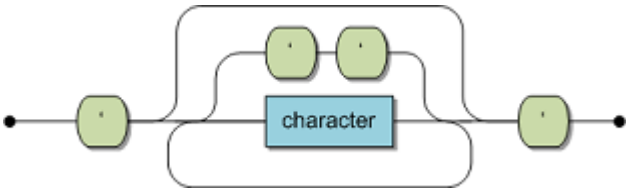
- The precision of the literal will be the count of all characters inside the quotes divided by two.

Examples of Binary Literals:

Literal	Data Type
x'AF034879F0CC'	binary(6)
x'' (empty quotes)	binary(0)

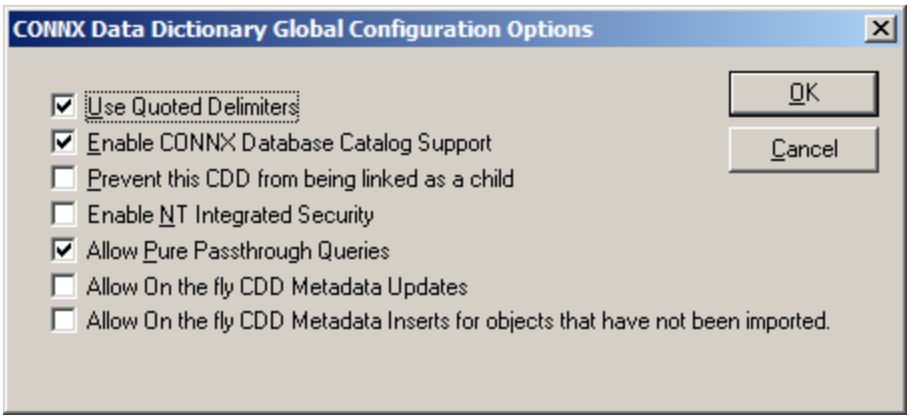
Character Literals

Syntax



- Character literals must be prefixed and suffixed by a single quote character.

  If the "Use Quoted Delimiters" option is not selected in the data dictionary, then character literals can also be prefixed and suffixed by a single double quote character.



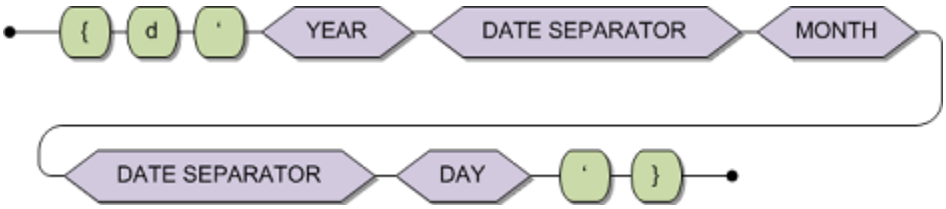
- The data type of a character literal is SQL Char, unless the literal contains a character outside of the Latin character set. If the literal contains a character with a code point greater than 255, its data type is SQL Unicode.
- The precision of the literal will be the count of all characters inside the quotes.
- An empty string will be treated a varchar(1) field, with a space.

Examples of Character Literals:

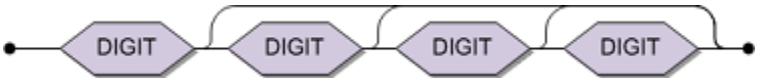
Literal	Data Type
'abcdef'	Char(6)
'abcdefg'	Char(7) only if "Use Quoted Delimiters" is disabled; otherwise this is treated as an identifier
'한국어/조선말'	Unicode(7)
'' (empty quotes)	Varchar(1)

Date Literals

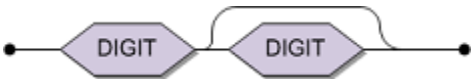
Syntax



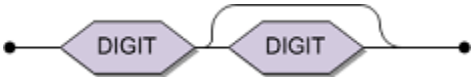
YEAR:



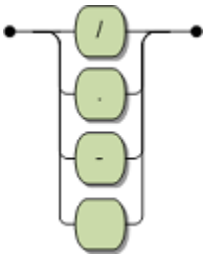
MONTH:



DAY:



DATE SEPARATOR:



A date literal has the format:

{d 'YYYY[- | / | .]MM[- | / | .]DD' }

where **YYYY** is the four digit year, **MM** is the one or two digit month of the year (between 1 and 12), and **DD** is the one or two digit day of the month (between 1 and 31).

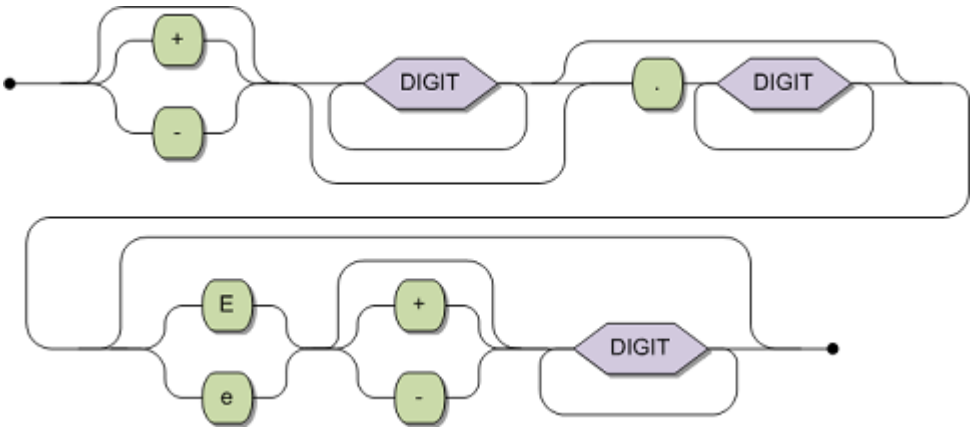
- CONNX does not assume the date is in the current century if **YYYY** has two digits. For example, the literal {d '02-01-14'} refers to the year 2 C.E., not 2002 C.E. or 1902 C.E.
- Date literals have a Date SQL type.
- When a character literal is used in an expression that expects a date, CONNX will assume the supplied character string is a date literal, even without the date literal prefix '{d' and suffix '}'.

Examples of valid date literals:

Literal	Validity
{d '2014-03-01' }	valid
{d '1920-12-20' }	valid
{d '1920.12.20' }	valid
{d '1950/1/20' }	valid
{d '1920-12-20' }	valid
{d '2014-13-01' }	invalid month
{d '1920-12-99' }	invalid day

Numeric Literals

Syntax



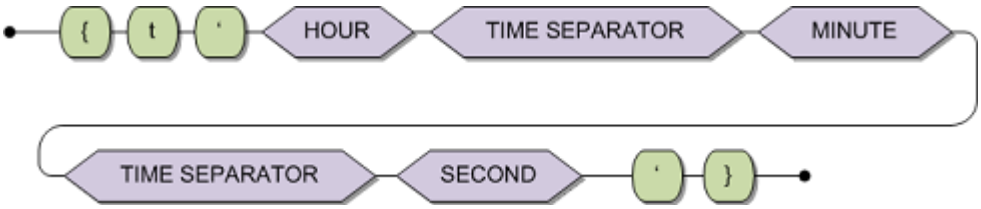
- If a numeric literal contains only digits, an optional sign, and an optional decimal point, it will have a data type of SQL Numeric.
  - The precision of the literal will be the count of all digits before and after the decimal point.
  - The scale of the field will be zero if no decimal point is present.
  - If a decimal point is present, the scale will be the total number of digits following the decimal point.
- If the "e" symbol is present in the numeric literal, the number is in scientific notation, and the data type will be Qfloat (see Data Types). At data retrieval time the value will be downgraded to SQL double.

Examples of Numeric Literals:

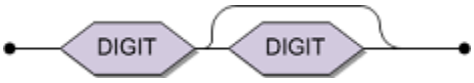
Literal	Data Type
1234	Numeric(4,0)
-123456	Numeric(6,0)
12345.56	Numeric(5,2)
123e0	Double

Time Literal

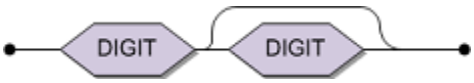
Syntax



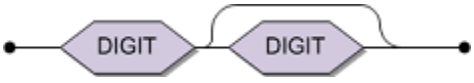
HOUR:



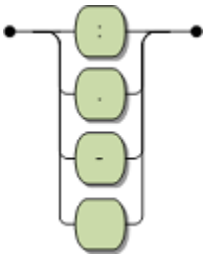
MINUTE:



SECOND:



TIME SEPARATOR:



A time literal has the following format.

{t 'HH[: | - | .]MM[: | - | .]SS' }

where **HH** is the one or two digit hour, in 24 hour format (0 to 23), **MM** is the one or two digit minute (0-59) and **SS** is the one or two digit seconds (0-59)

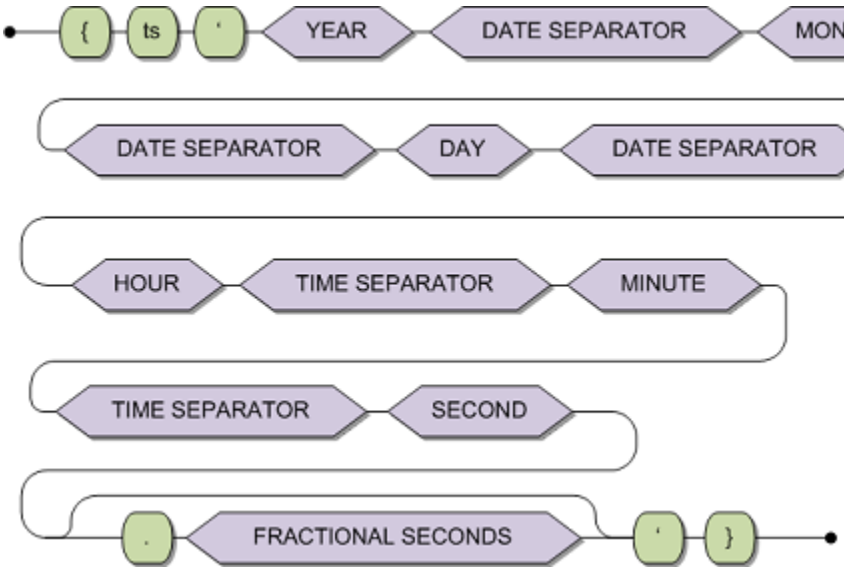
- Time literals cannot specify a seconds value with a fractional component. Only the timestamp literal has a seconds value with a fractional component.
- Time literals have a Time SQL type.
- When a character literal is used in an expression that expects a time, CONNX will assume the supplied character string is a time literal, even without the time literal prefix '{t' and suffix '}'.

Examples of valid time literals:

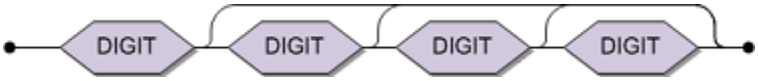
Literal	Validity
{t '12:34:56' }	valid
{t '12-34-56' }	valid
{t '12.34.56' }	valid
{t '99:03:33' }	invalid hour
{t '00:99:00' }	invalid minute

Timestamp Literal

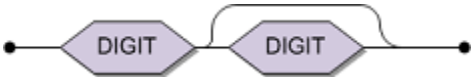
Syntax



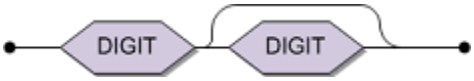
YEAR:



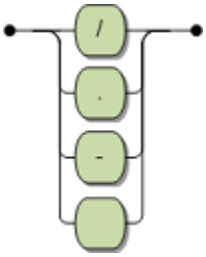
MONTH:



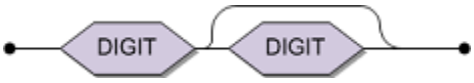
DAY:



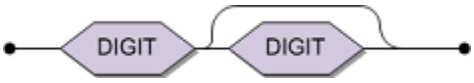
DATE SEPARATOR:



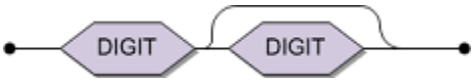
HOUR:



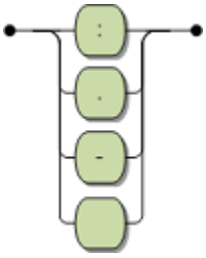
MINUTE:



SECOND:



TIME SEPARATOR:



A timestamp literal has the format:

{ts 'YYYY[- | / | .]MM[- | / | .]DD[| - | . | /]HH[: | - | .]MM[: | - | .]SS[- | / | .FFFFFFFFF]' }

where :

- **YYYY** is the four digit year
- **MM** is the one or two digit month of the year (between 1 and 12)
- **DD** is the one or two digit day of the month (between 1 and 31)
- **HH** is the one or two digit hour in 24 hour format (0 to 23)

  - **MM** is the one or two digit minute (0-59), **SS** is the one or two digit seconds (0-59)
  - **FFFFFFFFF** is the zero to 10 digit fractional component of the seconds
- Timestamp literals have a Timestamp SQL type.
- When a character literal is used in an expression that expects a timestamp, CONNX will assume the supplied character string is a timestamp literal, even without the timestamp literal prefix '{ts' and suffix '}'.

Examples of valid timestamp literals:

Literal	Validity
{ts '2014-03-01 12:34:56' }	valid
{ts '2014-03-01 12:34:56.12345678' }	valid
{ts '2014/03/01 12.34.56' }	valid
{ts '2014.03.01-12.34.56.12345678' }	valid
{ts '2014.33.01-12.34.56.12345678' }	invalid month

SQL Tokens

SQL consists of lexical units called tokens. Tokens consist of identifiers, constants and keywords. Delimiters are used to separate tokens.

Identifiers

Identifiers are used to identify or name objects like tables, columns, schemas, and indexes.

There are two kinds of identifiers, regular and delimited.

Regular identifiers are:

- Not delimited by double quotes
- Case insensitive
- Comprised of only letters, digits and the underscore character

Delimited identifiers are:

- Delimited by double quotes
- Case sensitive
- Comprised of letters, digits, and any of the following characters: % & ' ( ) * + , - . / : ; < = > ? [ ]

Use a delimited identifier if your object:

- Contain blanks
- Does not start with a letter
- Is identical to a keyword
- Is case-sensitive
- Contains any of the following characters: % & ' ( ) * + , - . / : ; < = > ? [ ]

Examples

Incorrect Indetifier	Correct Identifier	Explanation
create view table as select TABLE_NAME,TABLE_TYPE from information_schema.tables where table_schema=USER;	create view "table" as select TABLE_NAME,TABLE_TYPE from information_schema.tables where table_schema=USER;	Table is a keyword and can not be used as a regular identifier. If it is used as a delimited identifier, it can specify a view name.
select col1-1 from tab1;	select "col1-1" from tab1;	col1-1 is a numeric operation in select list and can not be used as a regular identifier. If it is used as a delimited identifier, it can specify a column name.

CONNX can handle identifiers of up to 128 characters.

Delimiters

A delimiter is used to separate the lexical units in the language for compiler processing. Delimiters are either spaces, control characters, comments or special tokens. A comment must be preceded by double hyphens (--).

The following symbols are used as special tokens in SQL:

,	()	<	>	.
:	=	+	-	*
<>	<=	>=	/	;
?	~>	~<	~=	"
||				

Correlation Identifiers

Correlation identifiers assign a new identifier to a table.

Correlation identifiers can only be defined in:

- the FROM clauses of either a query specification
- a DELETE statement
- an UPDATE statement.

Important: Correlation identifiers can only be used in the query specification, DELETE or UPDATE statement where they were defined.

Correlation identifiers affect the query specification or statement where they have been defined and all the sub-queries present within that query specification or statement.

If a correlation identifier has been defined for a table and a column of the table needs to be qualified, only the correlation name can be used to do so. The original table name or synonym can not be used.

Whenever you need to distinguish between two separate occurrences of the same table use a correlation identifier. In this example,

```
SELECT * FROM cruise a, cruise b;
```

the table CRUISE now logically exists twice and can be referenced as either A or B.

You can also use a correlation identifier if you want to use a shorter qualifier within the statement.

Table Specification

A table specification is used to identify a base table or viewed table (view) in a user session.

If **catalog_identifier** is omitted, the default catalog for the current session is used.

If a default catalog hasn't been set via the SET CATALOG statement, CONNX attempts to match the table identifier against all available catalogs in the data dictionary. If more than one match is found across multiple catalogs, an error is returned.

If **schema_identifier** is omitted, the default schema for the current session or user is used.

If a default schema hasn't been set for either the current session via the SET SCHEMA statement or for the current user during the CREATE USER statement, CONNX attempts to match the table identifier against all available schemas in the data dictionary. If more than one match is found across multiple schemas, an error is returned.

Column Specification

A column specification is used to identify a column (which must contain a column identifier) in an SQL statement.

Column specification are used:

- to define a table column in a CREATE TABLE, CREATE TABLE DESCRIPTION or CREATE VIEW statement
- to represent the column value in a SELECT clause expression or in a search.
- in WHERE, HAVING, GROUP BY or ORDER BY clauses, to represent all the result values after the clause is applied.
- to represent all the row values resulting from the grouping operation as a function argument.

A column specification can be either a qualified column specification or an unqualified column identifier.

- a qualified column specification explicitly specifies the related table
- an unqualified column identifier does not explicitly specify the related table.

Here are the rules used to relate a column specification to one and only one table specification:

1. the current query specification occurs in the column specification.
2. successive query specifications are analyzed starting with the current query.
3. the candidate table specification is the first table specification containing the column specification definition.
4. only one candidate table specification per query specification.

Note: *If the candidate table is contained in a higher query specification than the current one, the column is an outer reference.*

Unqualified Column Specification

An unqualified column specification can be used when a column can relate unambiguously to one table (only one table within the same query specification contains the column identifier).

Example:

```
SELECT cruise_id,contract_id
    FROM cruise,contract;
```

Qualified Column Specification

A qualified column specification consists of a table specification followed by a column identifier separated by a period. Use a qualified column specification when a column cannot be unambiguously related to a table (multiple tables in the same query specification contain columns with the same column identifier).

Example:

This example shows how the column specification distinguishes between two columns of the same name in different tables. Specify both tables in the FROM clause.

```
SELECT contract.id_cruise, sailor.id_cruise
  FROM contract,sailor;
```

Use a qualified column specification when the same table may need to be referenced more than once in the same query specification. You cannot qualify the column identifier with the table specification. Instead, use a correlation identifier to distinguish between the different references of the same table (see Table Specification for more information).

Example:

Use the following syntax to find the least expensive cruise for each destination. Correlate the first instance of the table cruise with the letter X because the sub-query needs to distinguish between two identical column references on two different 'instances' of the same table.

```
SELECT cruise_id,start_harbor,cruise_price
   FROM cruise X
   WHERE cruise_price = ( SELECT MIN(cruise_price)
      FROM cruise
      WHERE destination_harbor = X.destination_harbor );
```

Outer Reference

Outer references are a special type of qualified column specification.

An outer reference is a reference to a table column specified in a higher-level query specification. They are required if a column identifier cannot be unambiguously related to a single table.

Using qualified column specifications for outer references increases SQL statement readability.

Example:

This example identifies all contracts that cost more than double the cruise price of the cruises that the contracts identify. The id_cruise column is an outer reference as the table it references is contained in the higher query specification.

```
SELECT contract_id FROM contract
   WHERE (price*2) > (SELECT cruise_price FROM cruise
      WHERE cruise_id = contract.id_cruise );
```

Naming Result Table Columns

Naming result table columns is a part of the SQL-2 standard entry level. This function can be used to change the displayed column name of a result table. For example:

```
SELECT col1 as lastname FROM tab1 ;
or
SELECT col1 lastname FROM tab1 ;
```

The displayed column name is then lastname instead of col1.

Query Specification

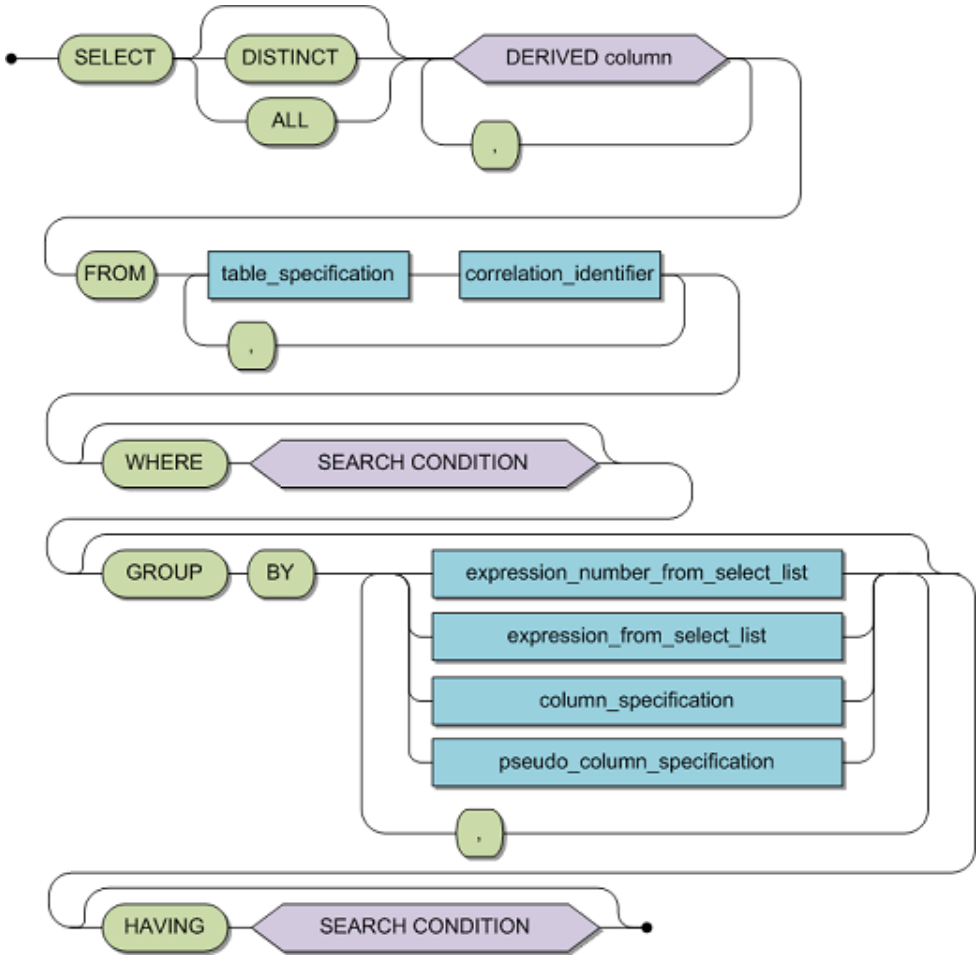
Function:

A query specification is used to define a resultant table.

Invocation:

A query specification can appear in one of the following contexts:

- as the operand of a query expression (in a DECLARE CURSOR statement),
- in a subquery within, for instance, a COMPARISON predicate,



Syntax:

DISTINCT	An optional directive which forces all rows of the resultant table to be unique, that is, duplicate rows will not be returned.

ALL	The default setting. Duplicate rows will be returned.
derived-column	The specification of the corresponding columns in the final resultant table derived by the query. Derived columns are separated by commas and all of them together are referred to as the derived column list (see separate diagram below).
table-specification	The specification of tables or views from which the resultant table is to be defined. Table and view names are separated by commas and all of them together are referred to as the table list.
correlation-identifier	Used to give an alternative name to a particular table for use within the query and subqueries which are in scope.
WHERE clause	The specification of a search condition which candidate rows must fulfill in order to become part of the resultant table.
GROUP BY clause	The specification of the desired grouping columns. A grouping column is the column by which the resultant table will be grouped.
HAVING clause	Specifies a search condition which candidate groups must fulfill in order to become part of the resultant table.

Syntax: Derived Column

correlation-identifier	An alternative name to a particular table for use within the query and subqueries which are in scope.
table-specification	The specification of a table or view. The correlation identifier and table specification must be specified in the table list of the FROM clause.
*	Abbreviated form of listing all columns of the table identified by the correlation identifier or the table specification. If this is specified, all columns of all tables specified in the table list of the FROM clause are selected. . In ANSI compatibility mode, the qualification of the asterisk in the form of the correlation identifier or the table specification is not permitted.
expression	A valid expression as described in the section Expressions.
column_title	Identifies the derived column in the resultant table.

Description

A query specification:

- defines the resultant table
- specified in the derived column list
- derived from the tables or views given in the table list,
- subject to the conditions imposed by the optional WHERE and/or HAVING clause
- and optionally grouped according to the GROUP BY clause.

Example:

The following describes the step-by-step processing of a query with the respective intermediate resultant tables. The abstract example uses a base table named T and columns named a, b, c and d. The apparent ordering of the intermediate resultant tables is due to ease of representation rather than of any predetermined ordering of the resultant tables.

```
SELECT a + 10, d, MAX(b) + 2
  FROM T
  WHERE c = 33
    GROUP BY a, d
    HAVING MIN(b) > 3;
```

1. The table list in the FROM clause actually defines all the candidate rows which may become part of the result. Conceptually, the first processing step of a query specification is to establish an intermediate resultant table containing all columns and all rows as defined in the table list. If only one table is involved, then the resultant table will be equivalent to the base table. However, should more than one table be listed, then all the tables in the list must be conceptually joined.

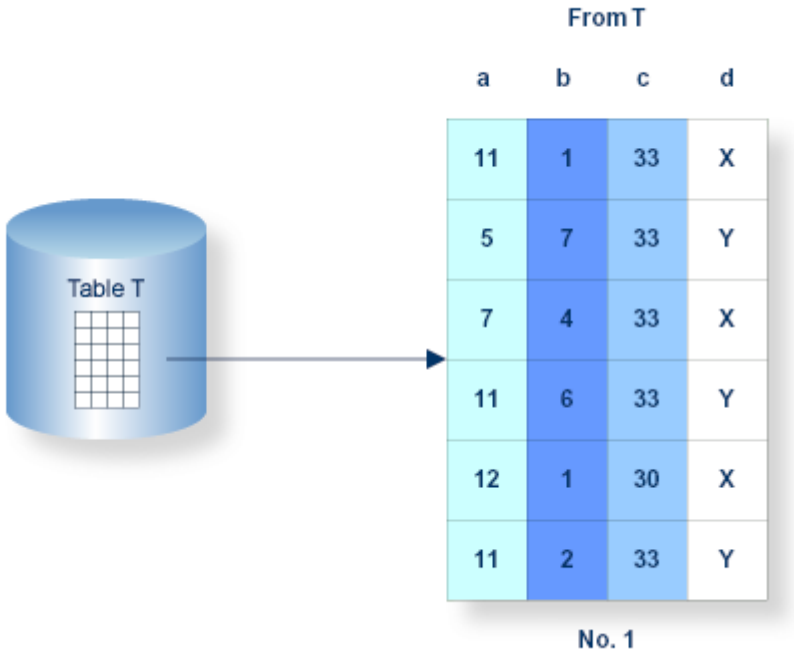


Figure for Processing Step 1

2. The next processing step concerns the WHERE clause. Each row in the intermediate resultant table is conceptually subjected to the search condition specified in the WHERE clause. If the condition equates to true, then the candidate row proceeds to the next stage. Otherwise, it is eliminated from further consideration, thus reducing the size of the final resultant table. Should no WHERE clause have been specified or the condition equate to true for all candidate rows then the subsequent resultant table will contain all rows as illustrated by the intermediate resultant table No.1.

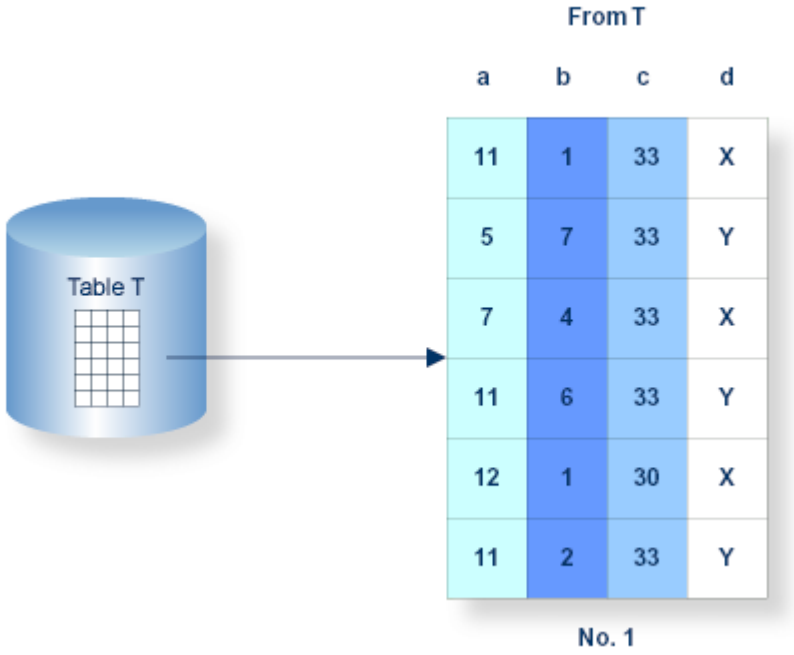


Figure for Processing Step 2

3. The next possible processing step concerns the GROUP BY clause. This step actually splits into two phases resulting in Tables No. 3 and No. 4. If built-in functions are used within a query, it is called a grouped query. The query is also grouped if a GROUP BY clause is specified, even if no functions are given. Built-in functions are aggregate operators which operate on a set of values in order to produce a single value as a result. These functions can be applied to the whole intermediate resultant table in order to produce a final resultant table of one row. In such a case, no GROUP BY clause is specified but the query is still grouped, as it uses built-in functions. Any column referenced within a grouped query must be an operand of a function, a grouping column or appear anywhere in the WHERE clause. This is because outside of the WHERE clause, the query is concerned with groups instead of mere rows. The converse, however, is not true. A grouping column may appear in a function. In the case of a special register, it must be specified exactly the same way as it appears in the SELECT list.

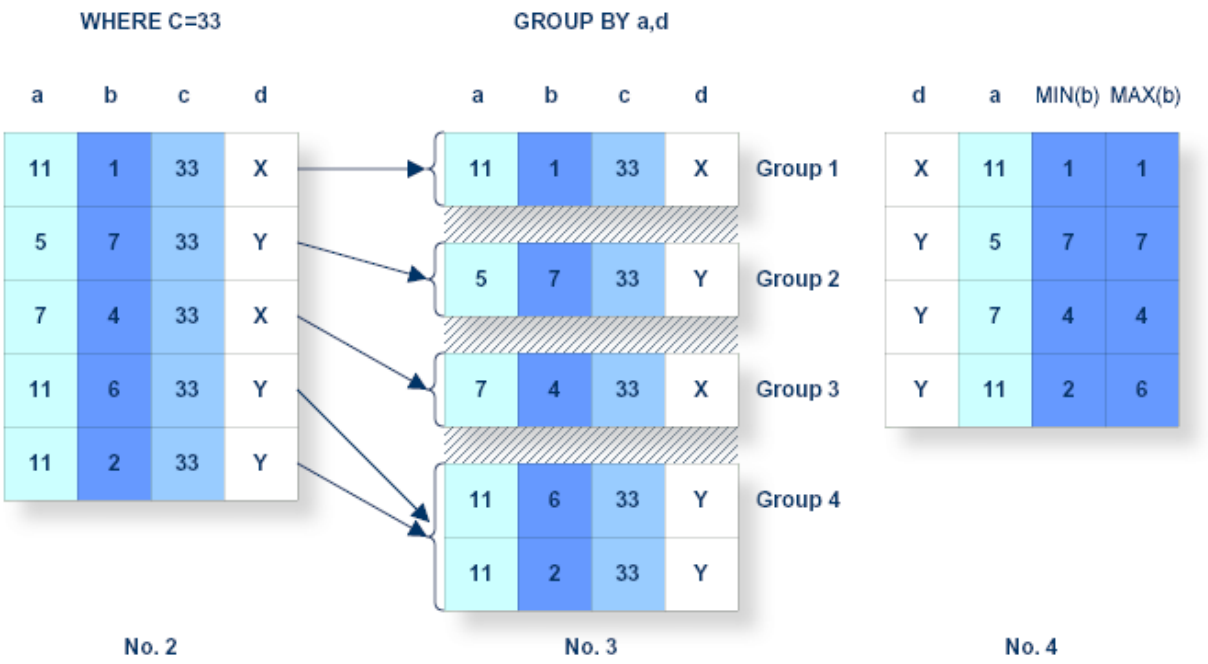


Figure for Processing Step 3

It is possible to divide the intermediate resultant table into groups. Groups are partitioned by specifying at least one grouping column in the GROUP BY list. A group is then established by extracting all candidate rows from the intermediate resultant table No. 2, where the value of the grouping column/s is/are equal. As many groups are established as there are differing values of the grouping column. There is no predetermined ordering of these groups.

Groups are established as follows:

- identical values in the first grouping column are identified,
- if a match has been made, the values of the second grouping columns are compared (same procedure for all other grouping columns),
- if all values in the grouping columns are identical, a candidate row has been identified.

At this point the second phase is initiated. The query is examined in order to produce a list of the columns required for intermediate resultant table No. 4. These new columns are either grouping columns or columns derived from functions applied to columns in intermediate resultant table No. 3. In either case, only columns or functions appearing in the derived column list or the HAVING clause have to be considered. Thus, aggregate functions are applied to each group in turn resulting in one candidate row per group in intermediate resultant table No. 4.

The aggregate functions can now be applied to each group in turn resulting in one candidate row per group for the next conceptual intermediate table.

In conclusion, the GROUP BY clause establishes candidate groups which, when operated upon by the aggregate functions, are transformed into candidate rows, one per group, which form the next intermediate resultant table No. 4.

4. The next possible processing step concerns the HAVING clause . Each row in the intermediate resultant table is conceptually subjected to the search condition specified in the HAVING clause. If the condition equates to true, then the candidate row proceeds to the next stage, otherwise it is eliminated from further consideration. As such, it is analogous to the WHERE clause except it eliminates candidate groups rather than candidate rows. It is therefore permissible to use functions in the search conditions. In

fact, columns which are not contained in a function must be specified in the GROUP BY list.

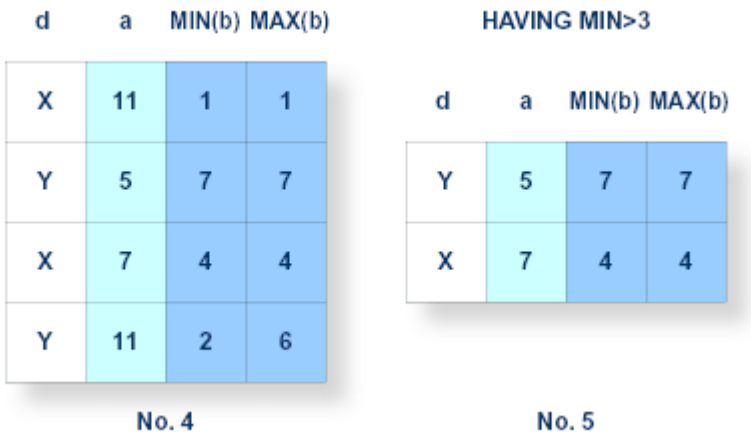


Figure for Processing Step 4

5. The final stage can now be executed, namely the production of the final resultant table . This is a derivation of the previous intermediate resultant table and is conceptually the same, regardless of whether it came from the HAVING, GROUP BY, WHERE or FROM clause. A resultant row is processed by evaluating each derived column in turn, based on the values contained in the corresponding row of the intermediate resultant table. This evaluation may be quite complex, depending on the nature of the expressions contained in the derived column's specification.

Step 5 finalizes the processing of this query by producing the final resultant table no. 6.

Figure for Processing Step 5

Derived Column List

A derived column list of at least one derived column must be specified. This may be done either as explicit expressions separated by commas or as an asterisk. The asterisk is an abbreviation representing all the columns as defined in the table list. An equivalent statement would simply list all columns explicitly, in the order in which they were defined in the original CREATE TABLE statement.

It is also possible to qualify the asterisk with a table specification which will result in all the columns belonging to the specified table only being derived.

Each derived column has an associated data type which is projected out of the subquery. The derived column may also have an identifier by which the derived column can be identified externally to the query specification e.g. from within an ORDER BY clause. If the derived column is based exclusively on a column of a base table, the identifier is the name of the column, in which case the derived column label is simply the fully qualified column specification. For all types of derived columns a new identifier can be specified with the 'AS<column identifier>' subclause.

It should be noted that the use of an asterisk with a table list made up of more than one table can lead to extremely large derived column lists.

Tables

- A query specification must have at least one table or view listed in the FROM clause. All column references must uniquely refer to one of these table references. If the same column name is present in more than one table in the FROM clause then it must be qualified by the appropriate table name, which itself may need to be explicitly qualified (see Table Specification for details).
- A table reference is in scope not only within the actual query specification in which it is declared but also for all subqueries that occur as part of this query specification. That is until the table is declared again in a lower subquery in which case columns referring to this table refer to the local declaration and not the outer one. Columns which refer to tables declared in an outer query specification are called outer references.
- Should more than one table be declared in the FROM clause, then the query is said to be joined. It is possible for a table to be joined with itself but in such a case, in order to make the table references unique within the FROM clause, at least one correlation name must be given.

Query Specification/Subqueries

- A subquery is a query specification which is subordinate to or nested in another query specification. In general, a subquery is also the origin of a value or a set of values. If this is the case, the number of derived columns in the derived column list of the query specification must be exactly one. The data type and length of such a value resulting from a subquery is the data type and length of that derived column.
- A correlation name is a means of giving an alternative label to a table within the query specification. Hence, if a column reference is qualified with the table name and a correlation name has been specified, then the qualification must be the correlation name.

Limitations

- A subquery may only return a derived column list with a cardinality of one. Within an unquantified COMPARISON predicate only one value may be returned. Please refer to COMPARISON, IN and EXISTS predicates for more details.
- Columns which are specified in grouped queries but are not themselves specified in functions are grouping columns and hence, must be listed in the GROUP BY list. This is only necessary for columns appearing either in the derived column list or in the HAVING clause, regardless of if they are referenced in a subquery of the grouped query or not. If there are no such columns then a GROUP BY list is not required, i.e the whole intermediate resultant table is considered to be a group. However one may be given if desired.
- A grouped query which is derived from a view can not reference columns from that view in any kind of expression.
- A DISTINCT directive may only appear once within the subquery. Hence, if the derived column list has been specified as DISTINCT then no functions may also

be specified with DISTINCT, whether they are in the derived column list, in the HAVING clause or even in a contained subquery.

ANSI Specifics

The keyword BY is mandatory in a GROUP BY clause.

Examples

The following example selects all contracts and associated cruise identifiers for all cruises booked on August 12th, 2002.

```
SELECT contract_id,id_cruise
    FROM contract
    WHERE date_booking = 20020812;
```

The following example creates a list of the different start harbors available.

```
SELECT DISTINCT start_harbor
    FROM cruise ;
```

The following example identifies all the contract IDs, customer IDs and cruise prices of all cruises that leave from Bahamas.

```
SELECT contract.contract_id, contract.id_customer,
cruise.cruise_price
    FROM contract,cruise
    WHERE cruise.start_harbor = 'BAHAMAS'
        and contract.id_cruise = cruise.cruise_id;
```

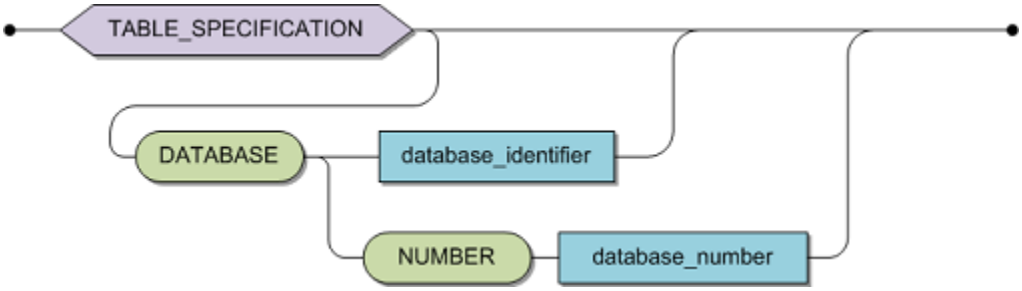
The following example selects the most expensive and least expensive cruise going to either Fethiye or Bodrum from Marmaris:

```
SELECT  start_harbor,
        destination_harbor,
        MAX(cruise_price),
        MIN(cruise_price)
        FROM cruise
        WHERE start_harbor = 'MARMARIS'
        GROUP BY start_harbor,destination_harbor
            HAVING destination_harbor = 'FETHIYE'
            OR destination_harbor = 'BODRUM' ;
```

Also see the detailed, illustrated examples earlier within this section.

Table Name Definition (Adabas only)

For each Adabas database, CONNX creates a base table. The base table is not part of a table cluster.



If database_number or database_identifier is omitted, CONNX uses the Adabas database identified by the catalog identifier (if provided) in the table specification. The specified Adabas database must be accessible for the SQL gateway.

Cluster Specification (Adabas only)

A cluster specification is used to identify a table cluster in a user session.

If **catalog_identifier** is omitted, the default catalog for the current session is used.

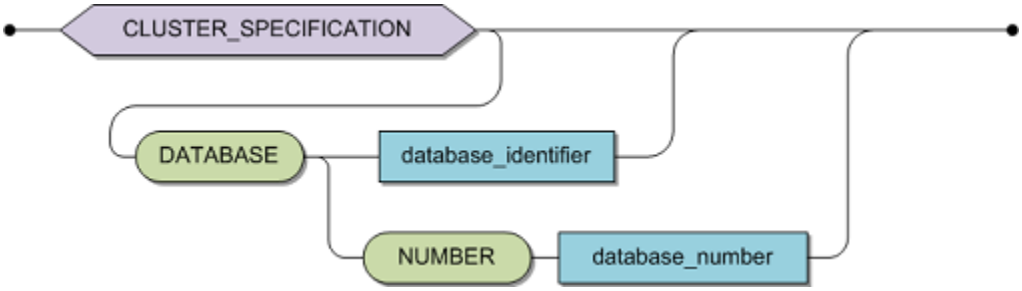
If no default has been set via the SET CATALOG statement, CONNX attempts to match the table identifier against all available catalogs in the data dictionary. If more than one match is found across multiple catalogs, an error is returned.

If **schema_identifier** is omitted, the default schema for the current session or user is used.

If a default hasn't been set for either the current session via the SET SCHEMA statement or for the current user during the CREATE USER statement, CONNX attempts to match the table identifier against all available schemas in the data dictionary. If more than one match is found across multiple schemas, an error is returned.

Cluster Name Definition (Adabas only)

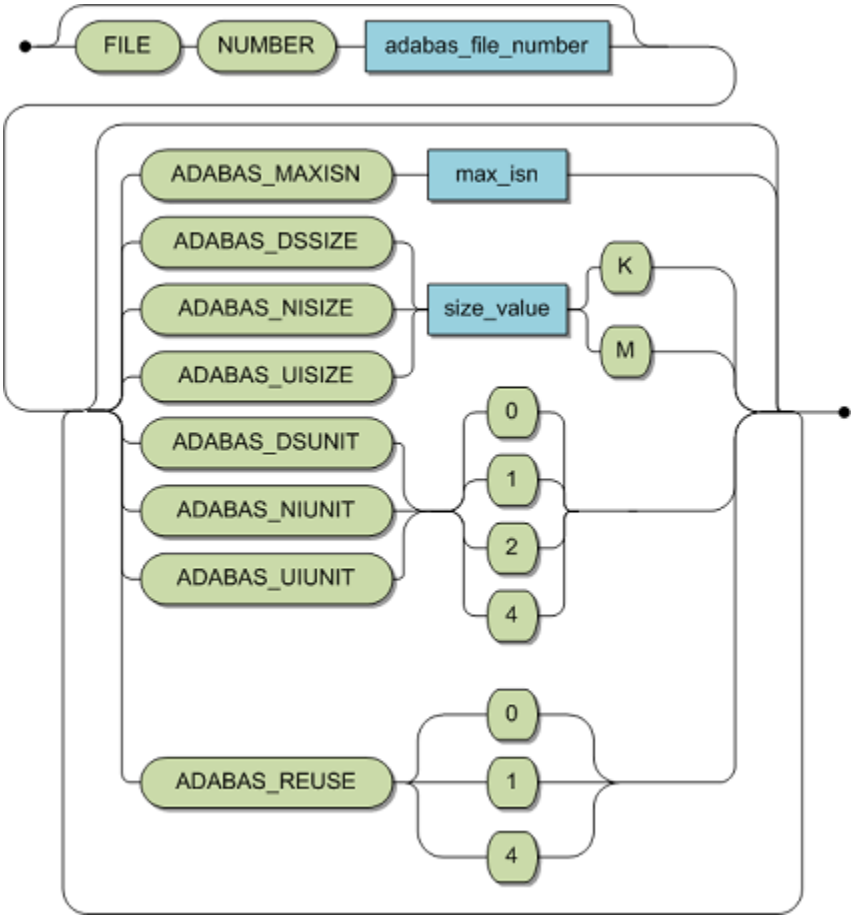
A table cluster is created in relation to a specific Adabas database.



If database_number or database_identifier is omitted, CONNX uses the Adabas database identified by the catalog identifier (if provided) in the cluster specification. The specified Adabas database must be accessible for the SQL gateway.

Adabas File Definition (Adabas Only)

Some Adabas file parameters can be specified when creating an Adabas file using a CREATE TABLE or a CREATE CLUSTER statement:



adabas_file_number	If specified, it has to be unused. If not, Adabas will choose a free one.
max_isn	The following identifiers are recognized: ADABAS_MAXISN
size_value	The following identifiers are recognized: ADABAS_DSSIZE ADABAS_NISIZE ADABAS_UISIZE By default, SIZE is specified in bytes. 'K' multiplies this value by 1024, 'M' multiplies this value by 1048576.
unit_param	The following identifiers are recognized: ADABAS_DSUNIT ADABAS_NIUNIT ADABAS_UIUNIT

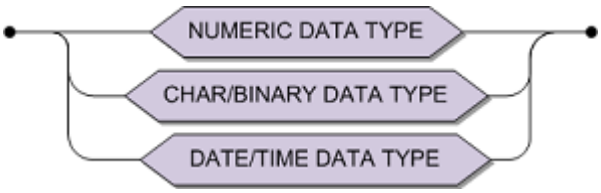
	0 = Default (Bytes) 1 = Blocks 2 = Bytes 4 = Megabytes
reuse_param	The following identifiers are recognized: ADABAS_REUSE 0 = Default (REUSEISN YES) 1 = REUSEISN NO 4 = REUSEISN YES

For the semantics of these parameters and also for default values not documented here, please refer to the Adabas documentation.

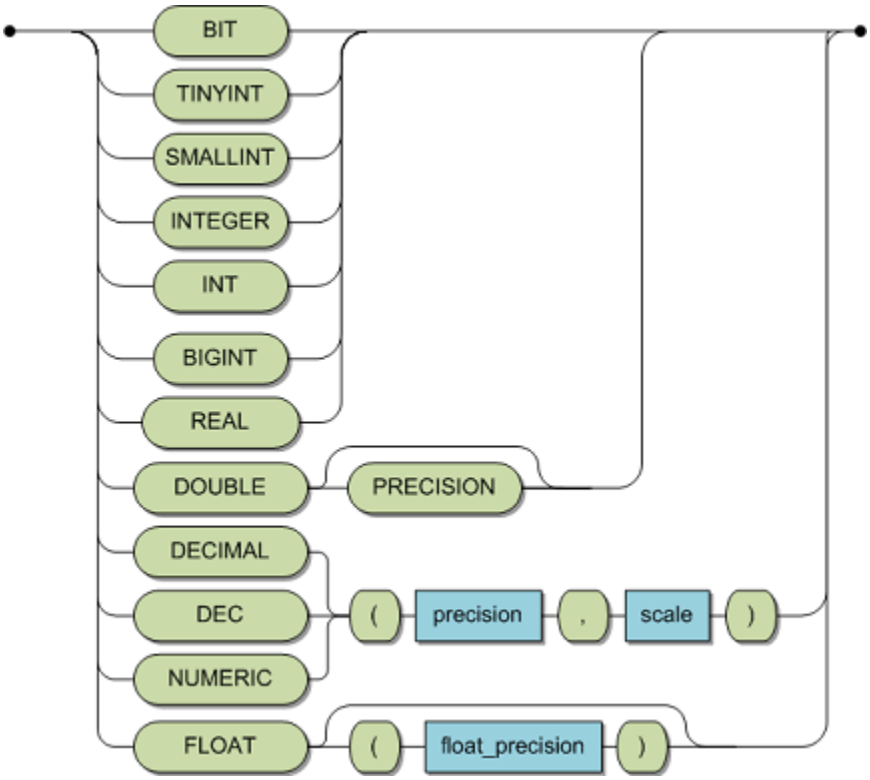
SQL Data Type

The following diagram shows the SQL statement syntax for multiple data types.

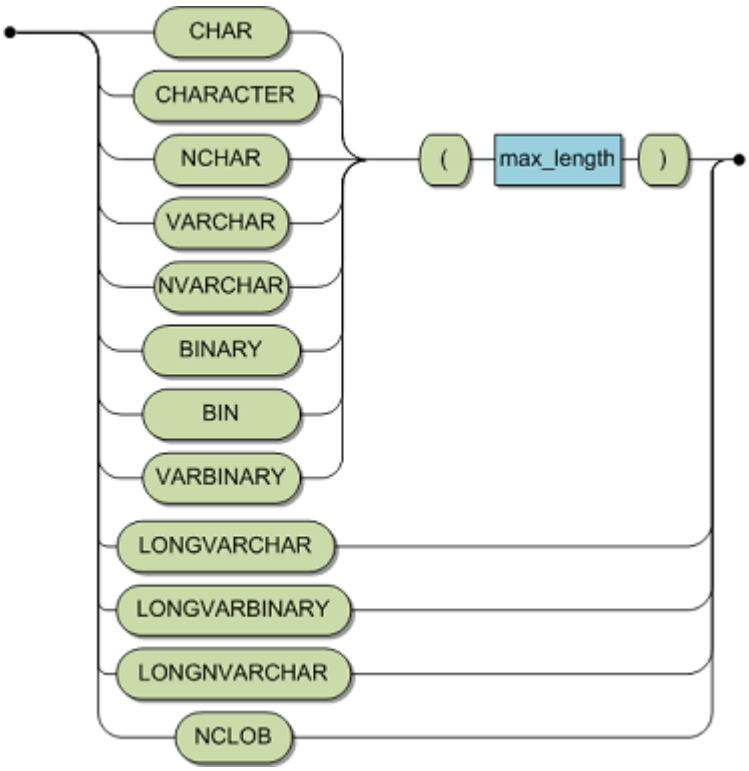
Syntax



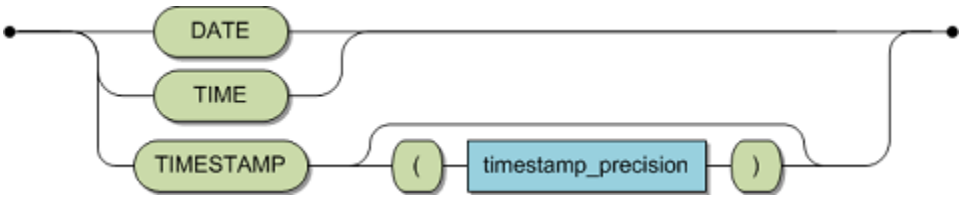
NUMERIC DATA TYPE:



CHAR/BINARY DATA TYPE:



DATE/TIME DATA TYPE:



Adabas Data Type

The following diagram shows the SQL statement syntax for the Adabas specific data types.

Syntax

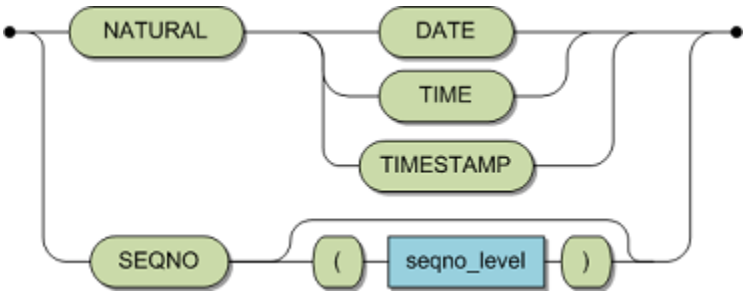


Table Elements (Adabas Only)

Adabas Column Clause

Function:

The Adabas column clause defines the table columns and attributes.

All columns must be unique within a table. Two table attributes may be the same when one attributes is a PRIMARY KEY or UNIQUE table constraint element and the other attribute is an INDEX table index element.

Invocation:

The Table Element specification is used with the following statements:

Create Table

Create Table Description

Create Cluster

Create Cluster Description

Syntax:

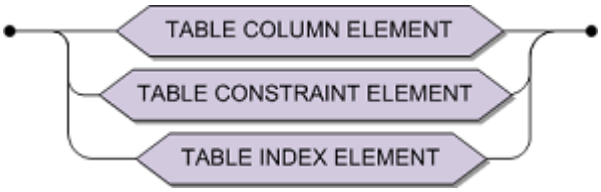


table column_element	Defines a column of a base table. A valid SQL table definition must contain at least one table column definition. See Table Column Element.
table constraint element	Specifies a UNIQUE, PRIMARY KEY or FOREIGN KEY constraint. See Table Constraint Element.
table index element	Specifies an index for the table. Table index element is not part of the ANSI SQL Standard. See Table Index Element.

Description:

The definition of foreign keys (part of Table Constraint Element) in the ANSI SQL standard, differs from that of Adabas SQL Gateway Embedded SQL.

A FOREIGN KEY table constraint element may only be specified in the CREATE CLUSTER/CREATE CLUSTER DESCRIPTION statements.

Table Column Element

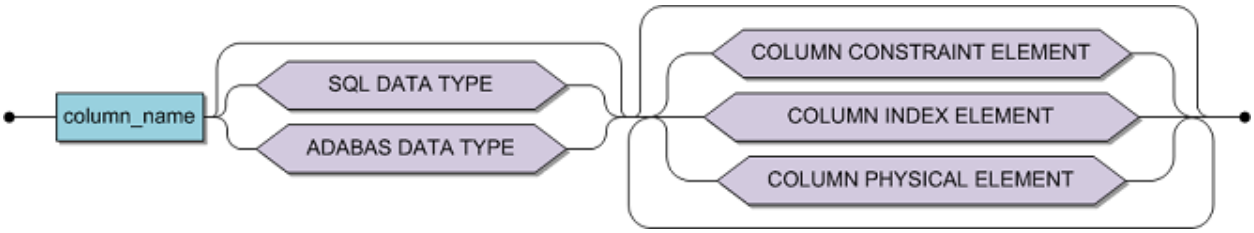
Function:

A table column element completely defines a base table column.

Invocation:

This is part of the table element and of the alter add element (ALTER TABLE Statement).

Syntax:



column_identifier	A valid identifier for a column and must conform to the rules specified earlier in Identifiers and Column Specification.
data_type	Specifies the data type of the column according to the rules specified below in SQL Data Types and Adabas Data Types.
column_constraint_element	Optional. Specifies constraints such as UNIQUE, NOT NULL, PRIMARY KEY, etc. See Column Constraint Element.
column_index_element	Optional. Specifies an index for a column. See Column Index Element.
column_physical_element	Optional. Describes the Adabas-specific information for each column, such as the short name, suppression, etc. See Column Physical Element.

Description:

The table column element specifies one column of a table with the attributes of this column (examples of attributes are constraints and indexes).

For each column, the column identifier and the data type definition are minimum requirements for the CREATE TABLE, CREATE CLUSTER or ALTER TABLE statements.

The column identifier must be a valid Adabas short name, else it is required to specify the Adabas short name (part of column physical element).

By default, all columns that do not have the explicit attribute NOT NULL, have implicitly the attribute NULL. There are two exceptions:

- for columns which are elements of a PRIMARY KEY, a NOT NULL constraint is generated automatically during the definition, if an explicit NOT NULL constraint is not provided.
- for columns in a table description having the FIXED attribute, a NOT NULL constraint is generated automatically during the definition, if an explicit NOT NULL constraint is not provided.

In CREATE TABLE DESCRIPTION and CREATE CLUSTER DESCRIPTION statements, any unspecified attributes that belong to the underlying Adabas field are automatically generated.

Limitations:

The column identifier must be unique within a table.

The following must be unique within a schema:

- Index identifier (if specified), one will be generated when not specified.
- Constraint identifier (if specified), one will be generated when not specified.

If a CREATE TABLE or CREATE CLUSTER statement is issued, then a table may only contain 926 columns. For CREATE TABLE DESCRIPTION and CREATE CLUSTER DESCRIPTION statements this limitation is lifted; you may specify elements of a PE or MU in a rotated format.

If a column has a Character data type and a precision greater than 253 characters, then the following must be true:

- The column attribute NOT NULL is mandatory.
- The column may not have attributes from Column Constraint Element (other than the above) or Column Index Element.
- The column may not have the attribute SUPPRESSION.

You cannot combine the following attribute pairs:

- SUPPRESSION and FIXED
- NULL and NOT NULL
- NOT NULL and DEFAULT NULL

The table below shows which parts of table column element are optional for which statements.

Statement	Data Type Definition	Column Constraint Element	Column Index Element	Column Default Element	Column Physical Element
Create Table **Create Cluster**	Mandatory	Optional	Optional	Optional	Optional (1)
Create Table Description **Create Cluster Description**	Optional	Optional	Optional	Optional	Optional (2)
Alter Table	Mandatory	Optional (3)	Optional	Optional	Optional (4)

(1) The SHORTNAME specification is not allowed in this statement.

(2) The SHORTNAME specification is mandatory for this statement.

(3) The NOT NULL attribute is allowed when combined with either DEFAULT ADABAS or SUPPRESSION.

(4) The only attributes allowed in this statement are NULL and SUPPRESSION.

ANSI Specifics:

The following elements are not part of the standard:

- Column Index Element
- Column Physical Element

- In Column Default Element the keyword ADABAS
- In Data Type Definition the keyword SEQNO

Adabas SQL Gateway Specifics:

None.

Example:

The following example creates one column of the base table CRUISE.

```
CREATE TABLE cruise
  (cruise_id NUMERIC(8) INDEX cruise1 NOT NULL UNIQUE);
```

Access to the underlying Adabas ISN is available via the pseudo column ISN_tablename, i.e. ISN_cruise.

Table Constraint Element

Function:

A table constraint specifies a constraint for a list of columns.

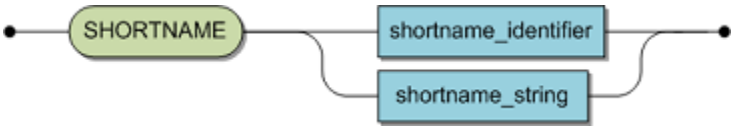
Invocation:

This element is part of the table element.

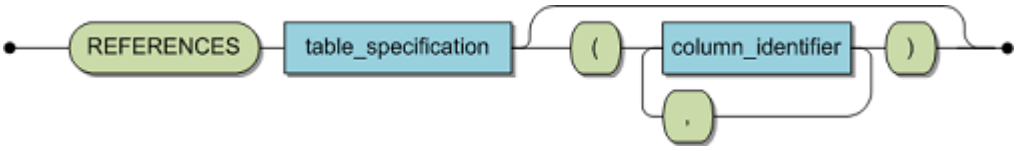
Syntax:

constraint identifier	A valid name for a constraint and must conform to the rules specified earlier in this section in section Identifiers.

Shortname clause:



Reference clause:



table_specification	Schema identifier. Table identifier is the expected format.
column_identifier	Optional. List of referenced columns.

Description:

UNIQUE and PRIMARY KEY constraints are called "unique constraints." A REFERENCES constraint is called "referential constraint." A table constraint element defines a constraint across one or more columns.

The following conventions hold true for the following explanations:

- Let CL = (c1,... , cn) be the column list of one or more columns for which this table constraint element is specified.
- Let T be the table where the columns of CL reside.
- PRIMARY KEY: There may only be a maximum of one PRIMARY KEY definition in a base table.
- A PRIMARY KEY constraint ensures that there are no two rows of T having identical values in the columns of CL. Each column of CL implies a NOT NULL constraint, that, if not specified, is generated automatically.
- PRIMARY KEYs on subtables of level one or two are limited to using all the columns of the associated FOREIGN KEY (the FOREIGN KEY that associates this level one or two table with its parent), plus a column of data type ISN_tablename, cnxarraycolumn and cnxarraycolumn_2 on the current level and any other columns of this level. The important point here is that only PRIMARY KEYs with a column of data type ISN_tablename, cnxarraycolumn and cnxarraycolumn_2 , for this table level, are classified as fulfilling the requirements for building a "unique constraint".

UNIQUE:

A UNIQUE constraint ensures that there are no two rows of T having identical values in the columns of C. Rows with NULL value(s), in any columns of C, do not effect this constraint.

FOREIGN KEY:

If specified, the REFERENCE's column list must conform to the following;

- The number of columns in CL and the number of columns in the references column list must be equal.
- The ith column of CL must be semantically the same as the ith column of the references list (i.e., the data type and attributes must match). The attributes UNIQUE and PRIMARY KEY should be converted to UQINDEX. The attribute REFERENCES is an exception to this rule.
- The columns of the references clause must match those of a "unique constraint" in the referencing table.
- All the columns of the "unique constraint" must have the attribute NOT NULL defined.

Limitations:

- All columns of Cl must exist in the defining base table, and a column of T may not appear twice within Cl.
- The FOREIGN KEY clause may only be used in a CREATE CLUSTER or CREATE CLUSTER DESCRIPTION statement.
- The SHORTNAME clauses may only be used in a CREATE CLUSTER DESCRIPTION or CREATE TABLE DESCRIPTION statement.
- There may be a maximum of one PRIMARY KEY for a bases table (this includes a column attribute of type PRIMARY KEY).
- When using a PRIMARY KEY constraint the attribute SUPPRESSION is not permitted.
- The NOT NULL attribute is not permitted when using a UNIQUE constraint in conjunction with a SUPPRESSION attribute.
- The NULL attribute is not permitted when using a UNIQUE or PRIMARY KEY constraint in conjunction with a DEFAULT ADABAS attribute.

ANSI Specifics:

- The columns of a UNIQUE constraint must under ANSI have the attribute NOT NULL specified.

Adabas SQL Gateway Embedded SQL Specifics:

None.

Table Index Element

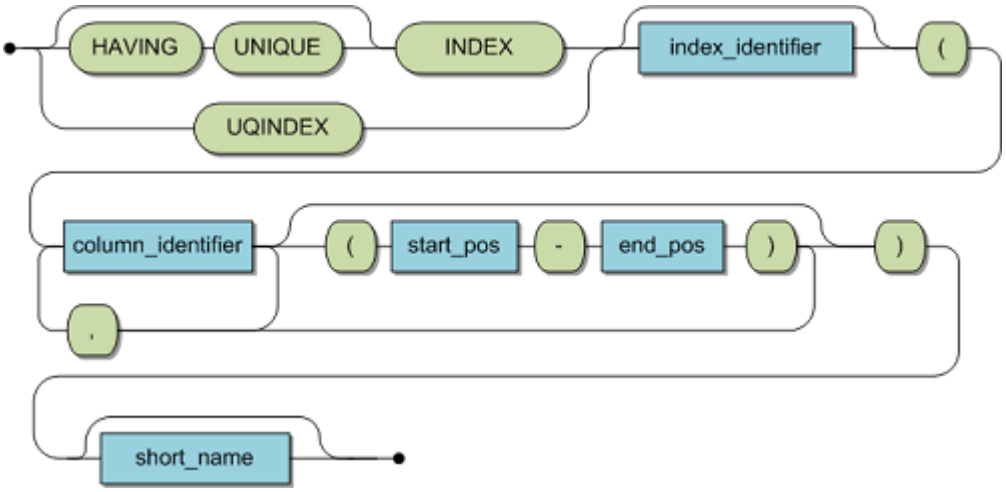
Function:

Specifies a set of columns as an index.

Invocation:

This element is part of the table element. The short-name identifier is only allowed in a CREATE TABLE DESCRIPTION or a CREATE CLUSTER DESCRIPTION statement.

Syntax:



index_identifier	Represents the name of an index and must conform to the rules specified earlier in this section in section Identifiers.
column_identifier	Name of a column to be used in the forming of an index.
start_position	Start position within the column when defining an Adabas descriptor.
end_position	End position within the column when defining an Adabas descriptor. The end position must be greater than the start position.
shortname_identifier	Adabas short name.

Description:

In order to improve an existing application's performance, establish an index for one or more base table column(s).

You can create an Adabas descriptor that reflects the capabilities of the Adabas database system's descriptors definition when you have the table index element. For a detailed discussion of Adabas descriptors, please refer to the Adabas documentation for your environment.

A ranges specification is when start and end positions are specified. This allows an index specification to be restricted to sub-elements of a column.

The following conventions hold true for the explanations below:

- Let Cl = (c1,... , cn) be a column list of one or more columns for which this table index element is specified.
- Let T be the table where the columns of Cl resides.

INDEX:

A INDEX is used to allow more efficient base table access. The index is based on one or more column(s) of a base table, when the listed columns' are considered as an entity.

An Adabas Superdescriptor will be generated if the number of columns in the column list is greater than one.
An Adabas Descriptor will be added if the index is in a single column with no range specified.
An Adabas Subdescriptor will be generated if a range specification is in a single column.

HAVING UNIQUE INDEX:

This feature is provided for compatibility and will be removed in future versions. Use a UNIQUE constraint instead.

UQINDEX:

A UQINDEX is an index that generates a unique Adabas sub- or superdescriptor on a sub-table column. SQL does not conisder this descriptor to be unique; it cannot be represented by a normal "unique constraint."

Limitations:

- The shortname identifier is only used in a CREATE TABLE DESCRIPTION or a CREATE CLUSTER DESCRIPTION statement.
- A specification of a UQINDEX is only valid for level 1 and level 2 base tables (subtables).
- You are not allowed to specify a UNIQUE INDEX together with a UNIQUE constraint or a PRIMARY KEY.
- When using a HAVING UNIQUE INDEX in conjunction with a SUPPRESSION attribute the attribute NOT NULL is not permitted.
- When using a HAVING UNIQUE INDEX in conjunction with a DEFAULT Adabas attribute, the attribute NULL is not permitted.

ANSI Specifics:

The table index element is not part of the Standard.

Adabas SQL Gateway Embedded SQL Specifics:

None.

Column Elements

Column Constraint Element

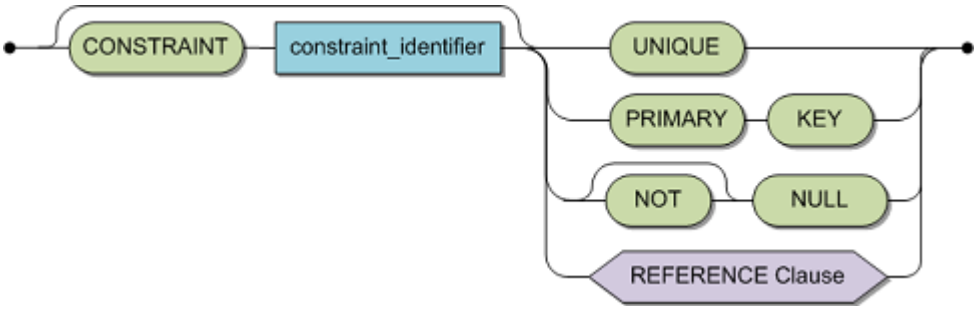
Function:

A column constraint element specifies the conditions which apply to each column.

Invocation:

This element is part of the table column element.

Syntax:



constraint_identifier	A valid identifier for a constraint conforming to the rules described in the section Identifiers.
UNIQUE	Only one UNIQUE constraint is allowed.
PRIMARY KEY	Only one PRIMARY KEY is allowed in a table.
NULL/NOT NULL	Indicates whether NULL values are permissible for this column.
reference_clause	Only allowed for subtables. The number of columns allowed in this particular case is one. For syntax regulations refer to Table Constraint Element.

For more restrictions, see CREATE CLUSTER and CREATE CLUSTER DESCRIPTION .

Description

A constraint is a base table sub-object that ensures actual data compliance with the specified conditions.

Adabas SQL Gateway Embedded SQL supports four different types of constraints:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY

A simple column constraint can be defined within a table column element. If a constraint refers to more than one column, it must be defined by a table constraint element. The constraint name (constraint identifier) must be unique within the schema. It will be generated automatically, if not specified.

UNIQUE and PRIMARY KEY constraints are called `unique constraints'. A REFERENCES constraint is called `referential constraint'.

The following conventions hold true for the following explanations:

- Let C be the column for which this constraint is specified.
- Let T be the table where column C resides.

UNIQUE:

A table can have many UNIQUE keys.

The UNIQUE constraint ensures that no two rows of T have the same value in column C. Rows with NULL values in column C do not affect this constraint.

PRIMARY KEY:

A table can have only one PRIMARY KEY.

The PRIMARY KEY constraint ensures that no two rows of T carry the same value in column C. When specifying a PRIMARY KEY constraint without an explicit NOT NULL constraint, an implicit one is generated.

NULL:

The NULL constraint indicate that you can have null values in any row of the table for the column C.

NOT NULL:

The NOT NULL constraint indicates that cannot have null values in any row of the table for the column C.

REFERENCES:

For details on how to define the reference clause, see Table Constraint Element (especially the FOREIGN KEY section). The number of columns allowed in this particular case is one.

Limitations:

The CREATE CLUSTER and CREATE CLUSTER DESCRIPTION statements have the following restrictions:

- A Column level REFERENCES constraint may only be used to build the referential constraint between tables of level 0 (base tables) and tables of level 1 (subtables).
- Only use the REFERENCES clause in the CREATE CLUSTER/ CREATE CLUSTER DESCRIPTION statements.
- There may be a maximum of one PRIMARY KEY for a bases table (this includes a table constraint of type PRIMARY KEY).
- Do not use the SUPPRESSION attribute with a PRIMARY KEY constraint.
- Do not use the NOT NULL attribute with a UNIQUE constraint
- Do not use the NULL attribute with a UNIQUE or PRIMARY KEY constraint.

ANSI Specifics:

The default referential triggered action differs from the ANSI standard. The default is CASCADE and not NO ACTION.

The NULL constraint is not part of the Standard.

Adabas SQL Gateway Embedded SQL Specifics:

Only the CASCADE option is supported.

Example:

The following example defines a column constraint which disallows NULL values and values which are not unique:

```
CREATE TABLE contract (
    contract_id integer NOT NULL UNIQUE );
```

Column Index Element

Function:

Specifies a column as an index.

Invocation:

This element is part of the table column element.

Syntax:

index_identifier	A valid identifier for an index conforming to the rules described in the section Identifiers.

Description:

The following conventions hold true for the following explanations:

- Let C be the column for which this column index element is specified.
- Let T be the table where column C resides.

INDEX:

If there is no range specified in an INDEX specification an Adabas descriptor will be added. If a range is specified across only one column, an Adabas Subdescriptor will be generated.

HAVING UNIQUE INDEX:

This feature is provided for compatibility and will be removed in future versions. Use the UNIQUE constraint instead.

UQINDEX:

A UQINDEX is an index that generates an Adabas unique sub- or superdescriptor on a subtable column. SQL does not consider this descriptor to be unique so it cannot be represented by a normal "unique constraint."

Limitations:

- A HAVING UNIQUE INDEX specification is not allowed in subtables. A UQINDEX specification is allowed in subtables.
- Do not specify a HAVING UNIQUE INDEX together with a UNIQUE constraint or a PRIMARY KEY.
- HAVING UNIQUE INDEX cannot have both the NOT NULL and the SUPPRESSION attributes.
- HAVING UNIQUE INDEX cannot have both the NULL and the Adabas DEFAULT attributes.

ANSI Specifics:

The Column Index Element is not part of the Standard.

Adabas SQL Gateway Embedded SQL Specifics:

None.

Column Physical Element

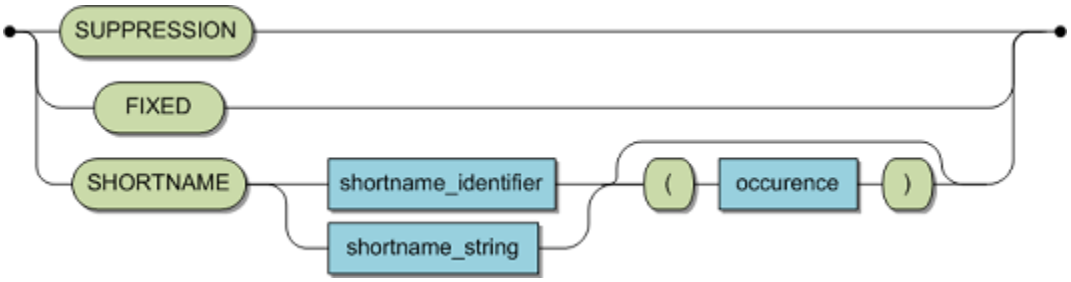
Function:

A column physical element is used to add Adabas-specific attributes to a column.

Invocation:

This element is part of the column element. This clause is only allowed in CREATE TABLE/ CREATE TABLE DESCRIPTION or a CREATE CLUSTER/CREATE CLUSTER DESCRIPTION statements.

Syntax:



shortname_identifier	Specifies an Adabas short name for a column.
string literal	Represents the Adabas short name for a column.
numeric literal	Optional. Only applies to MUs and PEs. Specifies a rotated field. Value must be less or equal to 191.

Description:

The following table associates an Adabas SQL Gateway Embedded SQL option with the corresponding Adabas option:

Adabas SQL Gateway Embedded SQL	Adabas
FIXED	FI
SUPPRESSION	NU

The SHORTNAME identifier specifies the Adabas short name of the corresponding Adabas field.

If the shortname_identifier is a numeric literal, it specifies a rotated field. Each occurrence is mapped to an individual column:

If a particular MU or PE field (semantically) has a non varying number of occurrences, then the field can be 'rotated.' Each occurrence is mapped to an individual column. For example, if an MU has 12 occurrences and each represents a month, then each occurrence could be mapped to the columns January through to December. A similar technique can be used for PEs, although each field within each occurrence must be individually mapped to a column.

Limitations:

- An Adabas short name must consist of exactly two characters; the first character must be between A and Z and the second can be between A and Z or between 0 and 9. Do not use the short names E0 to E9; they are Adabas reserved names. If your short name is a reserved word you must represent it in string format. For example, reserved word AS would be represented by SHORTNAME 'AS'.

- The short name specification is not case sensitive.
- Do not combine the FIXED keyword with the SUPPRESSION attribute.
- Use the FIXED and SUPPRESSION keywords only in the CREATE TABLE DESCRIPTION and CREATE CLUSTER DESCRIPTION statements. The underlying Adabas field must have these attributes.
- Do not use the PRIMARY KEY constraint with the SUPPRESSION attribute, .
- Do not use the NOT NULL attribute with either the SUPPRESSION attribute with the UNIQUE constraint or the SUPPRESSION attribute with the HAVING UNIQUE INDEX clause.

ANSI Specifics:

The column physical element is not part of the Standard.

Adabas SQL Gateway Embedded SQL Specifics:

None.

Example:

The following example stores bonus and sales for each month in a multiple-value field (each month is one occurrence). Each column is then rotated to be seen in one table (each month is a column of this table). An example of such a table description with rotated columns is:

```
CREATE TABLE DESCRIPTION rotated_table
DATABASE NUMBER 151  FILE NUMBER 53
(
id              CHAR(20)    SHORTNAME   "AA",
january_bonus   INTEGER     SHORTNAME   "DA"(1),
january_sales   INTEGER     SHORTNAME   "DB"(1),
february_bonus  INTEGER SHORTNAME   "DA"(2),
february_sales  INTEGER SHORTNAME   "DB"(2)

...
december_bonus  INTEGER     SHORTNAME
"DA"(12),
december_sales  INTEGER     SHORTNAME  "DB"(12),
)
```

where: DA and DB are the short names for the fields within the periodic group.

Privilege Specification

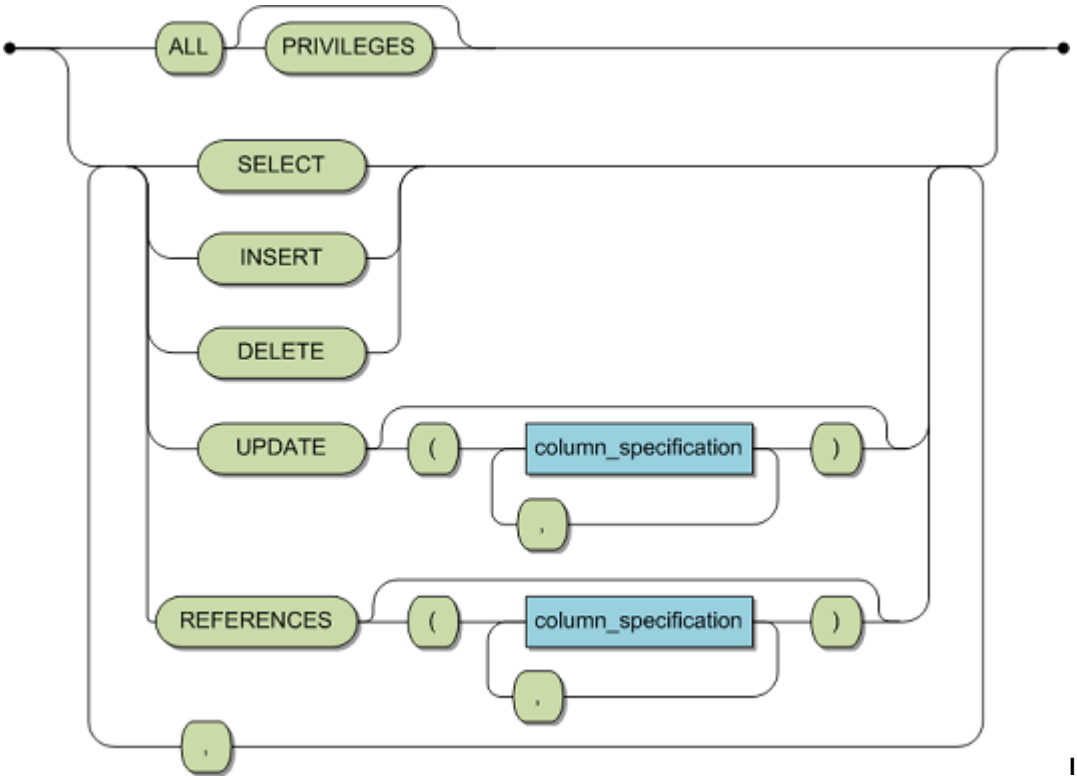
Function:

Defines the privileges which may be granted or revoked.

Invocation:

It can appear in GRANT and REVOKE statements.

Syntax:



column_specification	Identifies the column upon which the GRANT or REVOKE statements will be based. The column name must be a column defined on the specified table or view. If you specify more than one table or view, the column identifier must be valid for all tables or views.

Description:

Defines the privilege or set of privileges to be granted or revoked. These privileges are defined for specified tables or views.

The following privilege specifications may be defined:

SELECT	Enables the selection of data from the table(s) or view(s).
INSERT	Enables the insertion of data in the table(s) or view(s).
DELETE	Enables the deletion of rows from the specified table(s) or view(s).
UPDATE	Enables the updating of data in the specified table(s) or view(s). The

	UPDATE privilege can be specified for a list of columns within the table(s) or view(s).

Limitations:

If a view is based on more than one base table (read-only view), then the SELECT privilege is the only one to be granted in this case.

ANSI Specifics:

The keyword PRIVILEGES is mandatory when specifying ALL.

CONNX Specifics:

The keyword PRIVILEGES is optional when specifying ALL.

Example:

See the **GRANT/REVOKE** statements for examples.

Grantee Specification

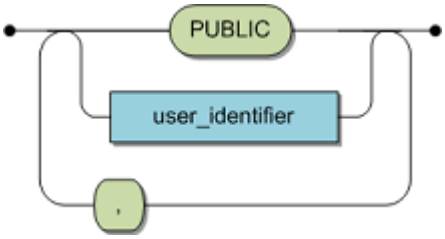
Function:

Identifies the individual(s) to whom privileges are to be granted or from whom privileges are to be revoked.

Invocation:

It can appear in GRANT and REVOKE statements.

Syntax:



user_identifier	Identifies the user to be granted/revoked privileges

Description:

Defines whether to grant to or to revoke the privilege or privilege set from either a particular user, from a list of users, or from all users. If you specify the PUBLIC option, granting or revocation the specified privilege will automatically affect all present and future users.

Limitations:

By default, table owners hold all the privileges for their tables and should not grant themselves, or revoke from themselves, additional privileges on their tables.

ANSI Specifics:

None.

CONNX Specifics:

None.

Example:

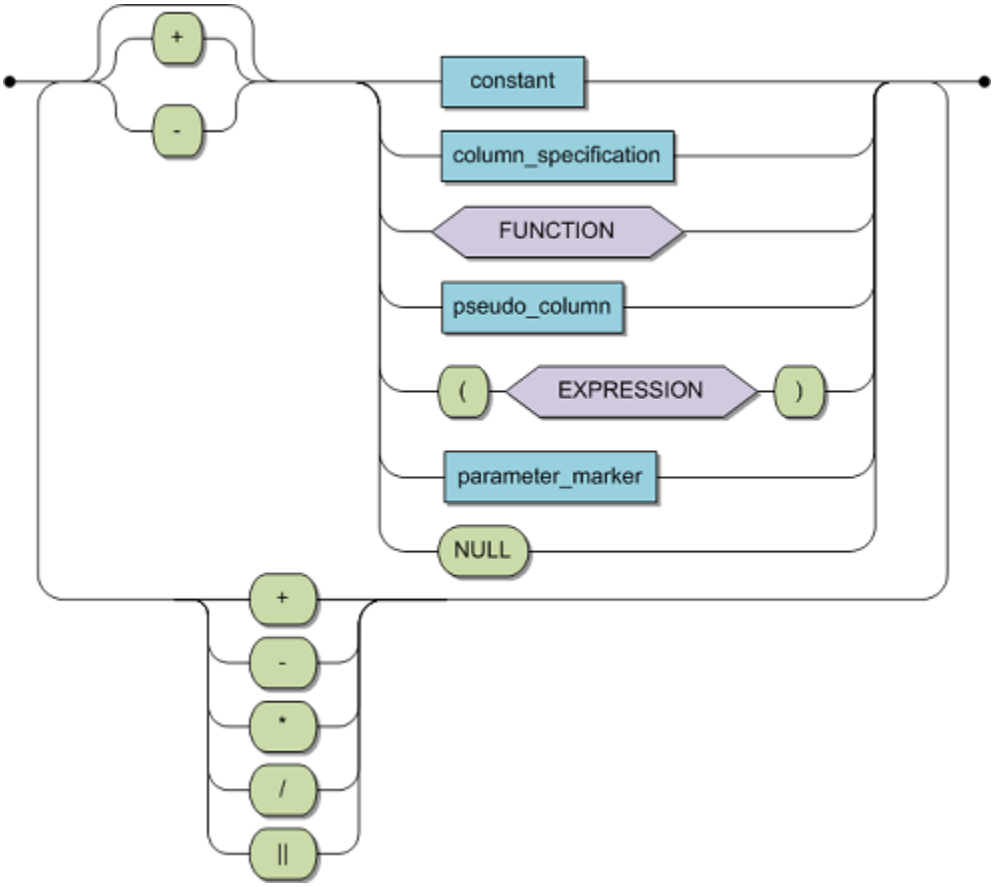
See the **GRANT/REVOKE** statements for examples.

Expressions

Expressions

An expression is a combination of operands separated by operators. An expression produces a result and is an value origin.

The following diagrams define the expression syntax:



Parameter Marker

For ODBC, OLE DB and JDBC, the literal for parameter markers is a single question mark (?)

For Adabas embedded SQL, this is a host variable name , in the format of :host_variable.

Expressions Without Operators

If an expression is used without operators, the result is the value represented by the object specified. For example. the result of an expression consisting of a column specification is the value represented in that column specification.

Expressions With Operators

The operators which can be used in expressions can be divided into monadic, diadic and concat operators:

- **Monadic Operators**

are prefix operators and have one operand. Monadic operators include the monadic plus (+) and the monadic minus (-) operators. The monadic plus operator does not change the value of its operand. The

monadic minus operator changes the sign of the value of its operand. Monadic operators can only be used with one numeric data type operand.

- **Diadic Operators**

are infix operators and have two operands. Diadic operators include the addition (+), subtraction (-), multiplication (*) and division (/) operators. Diadic operators can only be used with numeric data type operands. The resulting data type of an expression with two operands and a diadic operator depends on the data types of the two operands and on the operator. See Numeric Expressions - Dyadic operations.

- **Concat Operator**

is an infix operator and has two operands. The concat operator (||) will first implicitly convert non character values to character strings. Then the two strings are combined.

Assignments and Comparisons

All operations in SQL can be broken down to two basic operations: value assignment and value comparison.

Values are assigned during FETCH, UPDATE, INSERT and single-row SELECT statement processing. Value comparison takes place during predicate statement execution. Both assignment and comparison operations have two operands. An assignment operation has a receiving operand and a sending operand. In an assignment, the receiving operand gets the sending operand value. A comparison operation has two comparison operands whose values are compared with each other. Both assignment operands and comparison operation operands must have comparable data types.

In this example, assume Operand 1 has data type x. Operand 2 has a comparable data type only if its data type is:

- x or
- a data type which can be converted to x or
- a data type to which x can be converted, unless operand 1 is the receiving operand of an assignment operation. In this case, the data type is fixed and can not be changed.

Normally, character-string, Binary, and numeric data types are not comparable.

If both operands have different but yet comparable data types and a conversion has to be performed, this is always done from a `lower' data type to a `higher' data type (see **Mixed Operands** above).

Character-String Assignment

When a data type character-string value is assigned to a value recipient (either a host variable or a column), the value length and the value recipient's defined length are compared:

- If both lengths are the same, the recipient is assigned the value. After the assignment, the value and the recipient value are identical.
- If the value length is smaller than the recipient length, the value is padded with blanks.
- If the value length is greater than the recipient length, the value is truncated. If the INDICATOR variable was specified, it will show the number of truncated characters.

Numeric Assignment

When a numeric data type value is assigned to a recipient, data type conversion is performed when the value and the recipient data types are not identical.

Binary Assignment

When a binary data type value is assigned to a value recipient (either a host variable or a column), the value length and the value recipient's defined length are compared:

- If both lengths are the same, the value is assigned to the recipient. After assignment, the value and the value recipient are identical.
- If the value length is greater than the recipient length, an error condition is raised.
- If the value length is smaller than the recipient length, the value's most significant missing digits are appended with zeros.

If the application program is a remote client and Adabas SQL Gateway Embedded SQL resides on a server machine where ASCII/EBCDIC and/or byte swapping conversions would normally be induced during client/server communication, these conversions are suppressed for host variables. The host program must interpret the host variable contents.

For further information refer to the Adabas SQL Gateway Embedded SQL Programmer's Guide, sections: Dynamic SQL and Embedding SQL Statements in Host Languages.

Character-String Comparison

To compare two character-string data types, CONNX compares the corresponding character in each string. If the two strings do not have the same length, the shorter one of the two is appended with as many blanks as necessary, so both strings have the same length. The padding is done with the appropriate environment-dependent hexadecimal representation for a blank (e.g. x'20' for an ASCII environment and x'40' for an EBCDIC environment) and that padding is either to the right or to the left, depending on the underlying hardware architecture.

- Two values of data type character-string are equal if and only if both strings are empty (have a length of zero), or every corresponding character is the same. The comparison is done either from left to right or from right to left depending on and according to the underlying hardware architecture.
- Two values of data type character-string are unequal if at least one corresponding character is found to be unequal. The order of two unequal character-string values is determined by the first unequal character found during the comparison process (either from the left or from the right depending on the underlying hardware architecture). The order is then determined by the EBCDIC or ASCII collating sequence.

Numeric Comparison

The comparison of two values of data type numeric is performed following the normal algebraic rules taking the sign into account.

Example:

-5 is less than -3

Numeric comparison is always done between two values of the same data type. If two numeric values do not have the same data type, data type conversion is performed.

Binary Comparison

To compare two binary data type values, CONNX compares the corresponding bit digit in each value. The two values are equal if every corresponding digit is identical.

If the two values are of different lengths, the most significant missing digits of the shorter value are appended with the value `0'.

The comments regarding host variables and binary assignment, as described above, also apply to comparison.

Precedence of Operators and Parentheses

The operators in an expression are processed in a certain order. This order of precedence can be influenced by the use of parentheses. Operators of equal precedence are applied from left to right.

The following table lists all operators and parentheses in the order of their precedence:

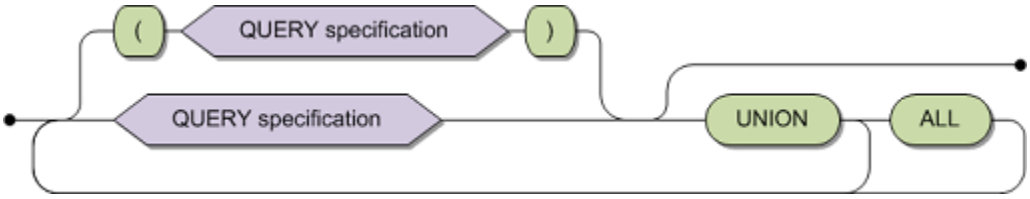
Operator	Function	Example
( )	Parentheses override precedence rules. Operations inside parentheses are applied first.	(x+y) * (x-y)
+ -	Monadic plus/monadic minus	-1
* /	Multiply, divide (diadic)	y/2
+ -	Add, subtract (diadic)	y-2

Query Expression

Function

A query expression is an expression involving one or more query specifications connected using the UNION operator.

Syntax:



ALL	Duplicate rows originating from different UNION operands are to be retained.
UNION	A diadic operator which takes specifications of resultant tables as its operands. These operands can be query specifications or deeper nested query expressions.
query_specification	The basic element of a query expression. It specifies a resultant table derived from a query. See Query Specification.

Description:

A query expression specifies a resultant table created from the possible UNION of several resultant tables as specified in corresponding QUERY specifications. A query expression can consist of a single query specification. You can use the UNION operator to add subsequent resultant tables to this initial query specification to produce a larger result.

The result of the UNION operation is a resultant table. The resultant table contains all rows from both operands after eliminating duplicate rows.

The result of a UNION operation with two base tables is a table that contains all rows belonging to either or both the operands.

- If the expression does not contains ALL you do not have to include DISTINCT in any of the query specifications because duplicates are automatically eliminated.
- If the expression does contain ALL, duplicated rows are retained. In that case, including DISTINCT in the query specification will affect the resultant table.

When ALL is either always specified with each UNION operator or never specified within the query expression, you do not have to use parentheses. If the ALL qualifier is only partially used, then the order of evaluation determines the final result; in that case, you may need to use parentheses.

Query expressions specified within parentheses are evaluated first; following that evaluation, expressions are evaluated from left to right.

When a UNION operator is specified, then the columns of the resultant table do not have derived column labels.

Limitations:

The two operands must be UNION-compatible; the derived column lists of the two operands must be of the same format.

Each derived column list must have the same number of derived columns and each derived column must be of the same data type as its corresponding derived column in the derived column list of the other operand.

ANSI Specifics:

None.

CONNX Specifics:

None.

Example:

The following example selects all cruise IDs for any contracts that require final payment or start before the 30th December 1991:

```
SELECT cruise_id
    FROM cruise
    WHERE start_date < = 19911230
UNION
SELECT id_cruise
    FROM contract
    WHERE date_payment < = 19911230;
```

Data Conversion

Data Type Priority

CONNX will implicitly convert data to similar or otherwise appropriate datatypes where possible in expressions.

Arithmetic operations:

CONNX will uplift the datatype of the dissimilar operands according to the table below. Two dissimilar data types will be converted to the type with the highest priority.

SQL Data Type	Priority
SQL_CHAR	0
SQL_UNICODE	1
SQL_VARCHAR	2
SQL_UNICODE_VARCHAR	3
SQL_LONGVARCHAR	4
SQL_UNICODE_LONGVARCHAR	5
SQL_DATE	6
SQL_TIME	7
SQL_TIMESTAMP	8
SQL_BINARY	9
SQL_VARBINARY	10
SQL_LONGVARBINARY	11
SQL_BIT	12
SQL_TINYINT	13
SQL_SMALLINT	14
SQL_INTEGER	15
SQL_BIGINT	16
SQL_DECIMAL	17
SQL_NUMERIC	18
SQL_REAL	19
SQL_FLOAT	20
SQL_DOUBLE	21
SQL_QFLOAT	22

Numeric Expressions - Dyadic operators

Non numeric data types are implicitly converted to numeric data types if possible. If the conversion is not possible the resulting value is zero.

Once both operands have been converted to a numeric type, the following table is used to determine the final data type of the expression.

Operand (A)	Operand (B)	Resulting Expression Data Type
bit, tinyint, smallint, or integer	bit, tinyint, smallint, or integer	integer
bit, tinyint, smallint, integer, bigint	bigint	bigint
bit, tinyint, smallint, integer bigint, decimal, numeric	decimal, numeric	numeric
bit, tinyint, smallint, integer, bigint, decimal, numeric, real, float, double	real, float, double	double
bit, tinyint, smallint, integer, bigint, decimal, numeric, real, float, double, qfloat	qfloat	qfloat

For integral data type operand combinations, remainders will be lost for multiplication and division operators.

If the resulting data type of the expression is decimal or numeric, the following table is used to determine the precision and scale of the result. P1 & S1 are the precision and scale of the first operand respectively. P2 & S2 are the precision and scale of the second operand respectively. M is a maximum SQL numeric precision of 38.

Operator	Resulting Precision	Resulting Scale
Addition & Subtraction	min (M, max (P1 - S1, P2 - S2) + max (S1, S2) + 1)	max (S1, S2)
Multiplication	min (M, P1 + P2 + 1)	min (M, S1, + S2)
Division	P1 - S1 + S2 + max(6, S1 + P2 + 1)	max(6, S1 + P2 + 1)

For division expressions, if the resulting precision exceeds M, then both the resulting precision and resulting scale are reduced equally until the resulting precision equals M. If the resulting scale after reduction is less than 6, it is set to a value of 6.

Aggregate Function Data Conversion

The table below shows the resulting data type for all aggregate functions based on the input data type (only numeric input types are listed). For aggregate functions that only accept numeric values as input, if a non-numeric value is passed to the function, it is implicitly converted to a numeric value automatically. For aggregate functions that accept non-numeric inputs, the data type of result will match the data type of the input value.

Aggregate function	Input converted to numeric implicitly	bit	tinyint	smallint	integer	bigint	decimal	numeric	real	float	double
AVG	Yes	numeric (7,6)	numeric (9,6)	numeric (11,6)	numeric (16,6)	numeric (26,6)	numeric*1	numeric*1	double	double	double
SUM	Yes	numeric (11,0)	numeric (11,0)	numeric (11,0)	numeric (16,0)	numeric (26,0)	numeric*2	numeric*2	double	double	double
MIN	No	bit	tinyint	smallint	integer	bigint	decimal	numeric	real	double	double
MAX	No	bit	tinyint	smallint	integer	bigint	decimal	numeric	real	double	double
COUNT	N/A	integer	integer	integer	integer	integer	integer	integer	integer	integer	integer
AVEDEVMEAN	Yes	double	double	double	double	double	double	double	double	double	double
AVEDEVMEDIAN	Yes	double	double	double	double	double	double	double	double	double	double
COEFVARPCT	Yes	double	double	double	double	double	double	double	double	double	double
COEFVARPCTP	Yes	double	double	double	double	double	double	double	double	double	double
FIRST	No	bit	tinyint	smallint	integer	bigint	decimal	numeric	real	float	double
KTHLARGEST	Yes	bit	tinyint	smallint	integer	bigint	decimal	numeric	real	float	double
KTHSMALLEST	Yes	bit	tinyint	smallint	integer	bigint	decimal	numeric	real	float	double
KURTOSIS	Yes	double	double	double	double	double	double	double	double	double	double
KURTOSISP	Yes	double	double	double	double	double	double	double	double	double	double
LAST	No	bit	tinyint	smallint	integer	bigint	decimal	numeric	real	float	double
MEDIAN	No	bit	tinyint	smallint	integer	bigint	decimal	numeric	real	float	double
MIDDLE	No	bit	tinyint	smallint	integer	bigint	decimal	numeric	real	float	double
MODE	No	bit	tinyint	smallint	integer	bigint	decimal	numeric	real	float	double
MULTIMODALCOUNT	No	integer	integer	integer	integer	integer	integer	integer	integer	integer	integer
MULTIMODALOCCUR	No	integer	integer	integer	integer	integer	integer	integer	integer	integer	integer
QUANTILE	Yes	double	double	double	double	double	double	double	double	double	double
QUARTILE1	Yes	double	double	double	double	double	double	double	double	double	double
QUARTILE3	Yes	double	double	double	double	double	double	double	double	double	double
RANGE	Yes	bit	tinyint	smallint	integer	bigint	decimal	numeric	real	float	double
RMSERROR	Yes	double	double	double	double	double	double	double	double	double	double
RMSERRORP	Yes	double	double	double	double	double	double	double	double	double	double
SKEWNESS	Yes	double	double	double	double	double	double	double	double	double	double
SKEWNESSP	Yes	double	double	double	double	double	double	double	double	double	double
SORTFIRST	No	bit	tinyint	smallint	integer	bigint	decimal	numeric	real	float	double
SORTLAST	No	bit	tinyint	smallint	integer	bigint	decimal	numeric	real	float	double
SORTMIDDLE	No	bit	tinyint	smallint	integer	bigint	decimal	numeric	real	float	double
STDDEV	Yes	double	double	double	double	double	double	double	double	double	double
STDDEVP	Yes	double	double	double	double	double	double	double	double	double	double
TRIMEAN	Yes	double	double	double	double	double	double	double	double	double	double
VARIANCE	Yes	double	double	double	double	double	double	double	double	double	double
VARIANCEP	Yes	double	double	double	double	double	double	double	double	double	double

*1 - Precision & Scale are calculated as follows:

P & S represent the scale & precision of the input respectively:

if S < 6, output precision = P + 6 - S, else output precision = P.

if S < 6, output scale = 6 else output scale = S.

*2 - Precision & Scale are calculated as follows:

P & S represent the scale & precision of the input respectively:

If P+6 < 11, output precision = 11 else if P+6 > 37, output precision = 37 else output precision = P+6.

output scale = S.

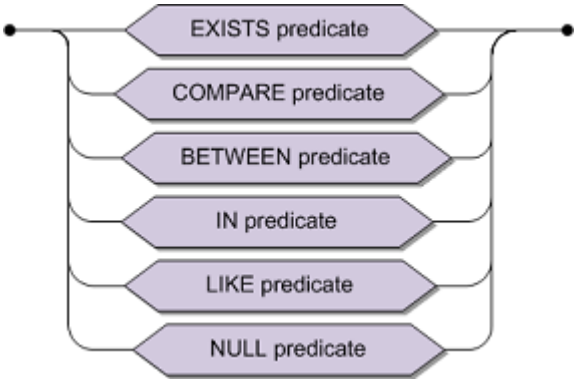
Predicates

What are Predicates?

A predicate is a tri-state (true, false, unknown) Boolean expression.

Predicates are composed of a search term contained within a search expression.

A predicate can take one of six forms.



For more information, see:

EXISTS Predicate
COMPARISON Predicate
BETWEEN Predicate
IN Predicate
LIKE Predicate
NULL Predicate

Between Predicate

Function:

This predicate checks to see if a specified value lies within the range defined and returns a tri-state boolean result.

Invocation:

One of the six predicates which constitute a search term.

Syntax:

expression_1	A valid expression as described in the section Expressions.
NOT	An operator which negates the result of the predicate.
expression_2 & expression_3	Each is a valid expression as described in the section Expressions.
AND	Simply separates expressions 2 and 3. Do not confuse with use as a boolean operator.

Description:

The BETWEEN predicate checks if the value specified by expression 1 lies within the range specified by the values derived from expression 2 and expression 3 respectively. It is equivalent to the following pair of COMPARISON predicates:

```
expression1 BETWEEN expression2 AND expression3

(expression1 >= expression2) AND (expression1 <= expression3);
```

CONNX processes the BETWEEN predicate as if it were expressed in this form.

The NOT operator negates the result of the boolean expression.

All expressions must have comparable data types. Should either of the expressions evaluate to NULL, then the predicate returns the tri-state value of unknown.

Limitations:

None.

ANSI Specifics:

None.

CONNX Specifics:

None.

Example:

The following example selectes the cruises that have a cruise price between and including 800 and 2000.

```
SELECT cruise_id
```

```
FROM cruise
WHERE cruise_price BETWEEN 800 and 2000 ;
```

Comparison Predicate

Function:

This predicate performs a comparison between two expressions and returns a tri-state boolean result.

Invocation:

One of the six predicates which constitute a search term.

Syntax:

expression 1	A valid expression as described in Expressions.
operator	One of the possible operators which must be chosen in order to perform the desired comparison.
expression 2	A valid expression as described in Expressions.
query specification	Contained in parentheses and may be given instead of the expression 2 .
ALL, ANY, SOME	Keywords which can be used to transform the comparison expression from unquantified to quantified.

Description:

Operands, expressions and query specifications must have a comparable data type. Should either of the expressions evaluate to NULL, then the predicate returns the tri-state value of UNKNOWN.

Expression 2 may be a query specification instead of a valid expression. This kind of query specification is a subquery or a subselect and has a cardinality of one. Since the data types must be comparable, the subquery may only specify one resultant column in its derived column list.

When used within an unquantified COMPARISON predicate, the resultant table may only return one value, thus `mimicking' a normal expression. If the subquery produce more than one result, or no result at all, it will return a runtime error. This cannot be checked at compilation time. Should the query return a NULL value, then the predicate equates to unknown.

The operator specifies the actual comparison operation to be performed. There are various alternative representations for the operators, depending upon which mode is current, as shown below.

A predicate is quantified if the subquery contains the keywords ALL, ANY or SOME. The subquery can return more values; it is no longer restricted to zero or one. When you use ALL, the predicate equates to true if the comparison with expression 1 is true for all values returned by the subquery. When you use ANY, only one of the comparisons need be true for the predicate to be true. The keyword SOME is equivalent to ANY.

Should any particular value equal NULL, then the predicate returns the value UNKNOWN.

Strings can also be greater or less than other strings. For example `Swindon' < `Swinton' would equate to true.

Limitations:

If a subquery is used in an unquantified comparison predicate, the subquery cannot contain a either a GROUP BY or HAVING clause; this would violate the requirement to return just one value. The subquery may not reference a grouped view as its source table.

ANSI Specifics:

ANSI only allows the following operator representations:

= > < <> <= >=

Example:

If used within an unquantified COMPARISON predicate, the subquery must only return one result. The following example selects cruises which are less expensive than the price for a cruise with Yacht ID 145.

```
SELECT cruise_id, destination_harbor, cruise_price
   FROM cruise
   WHERE cruise_price <
   ( SELECT MIN (cruise_price)
      FROM cruise
      WHERE id_yacht=145);
```

If used within a quantified COMPARISON predicate, the subquery may return more than one result. The following describes the step-by-step processing of a query with the respective intermediate resultant tables. The example uses a base table named T1 with columns named a, b, c and d and T2 with columns named e, f and g. The apparent ordering of the intermediate resultant tables is due to ease of representation rather than of any predetermined ordering of the resultant tables.

```
SELECT a,d
   FROM T1
   WHERE b < ALL
      (SELECT e
         FROM T2
          WHERE f = 10);
```

1. Establishes an intermediate resultant table containing all columns and all rows as defined in the table list for T1.
   For each row of IRT I, the subquery is evaluated and as described in Query Specification; IRT III is established from IRT II. This step is then performed for each occurrence of a row in IRT 1, as the result of the subquery may depend on values contained in IRT I. This occurs when the subquery contains an outer reference in its search condition.

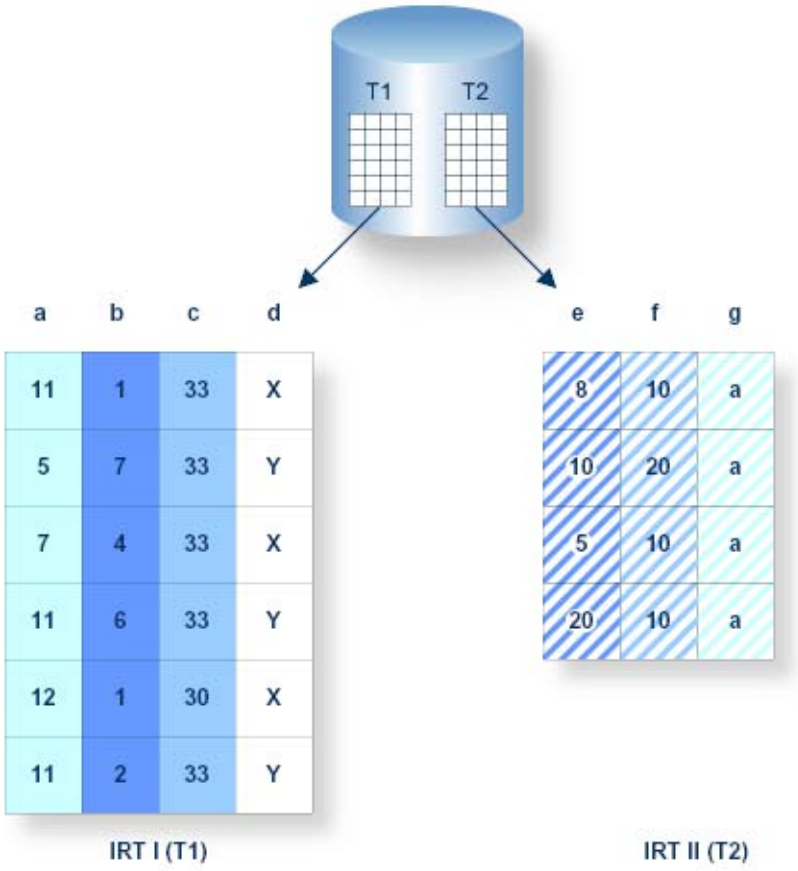


Figure for Processing Step 1

2. During this step, the subquery has been established as intermediate resultant table (IRT) III. The comparison can now take place.

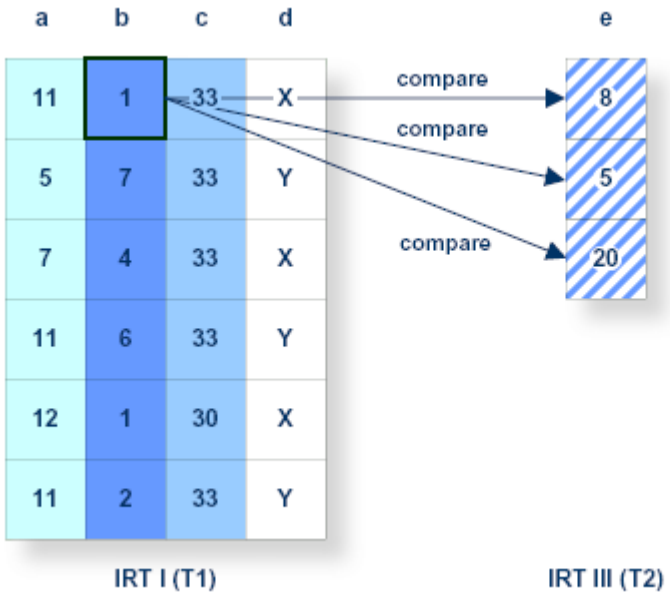


Figure for Processing Step 2

3. During this step, all rows of T1 containing a value in column b which is smaller than ALL values in column e of T2 qualify for the intermediate resultant table IV which is the final result of the query.

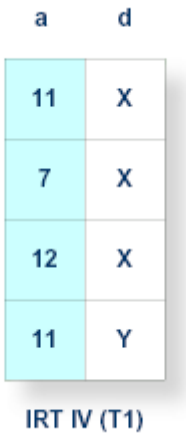


Figure for Processing Step 3

Exists Predicate

Function:

This predicate tests to see if a particular resultant table, as specified by the given subquery, actually exists. The resultant table will exist if any resultant rows were identified.

Invocation:

One of the six predicates which constitute a search term.

Syntax:

where query specification is the subquery whose resultant table is to be tested for existence.

Description:

Because the resultant table nature does not matter, the subquery may specify a derived column list of any desired cardinality and of any number of resultant rows. The exists predicate only evaluates whether the resultant table does or does not exist. It does not evaluate the derived column list.

If the resultant table does exist, the predicate is true; otherwise it is false. The predicate never equates to unknown.

WHERE `op' is any valid COMPARISON predicate operator:

- **WHERE x op (SELECT y FROM t)**

  WHERE EXISTS (SELECT * from t WHERE x op y)

Note: *After the transformation, the subquery no longer has to have a single value result.*

- **WHERE x op ANY (SELECT y FROM t)**
  WHERE EXISTS (SELECT * FROM t WHERE x op y);

- **WHERE NOT x op ALL (SELECT y FROM t)**
  WHERE NOT EXISTS (SELECT * FROM t WHERE x NOT op y);

- **WHERE NOT a op ALL (SELECT y FROM t)**
  WHERE EXISTS (SELECT * FROM t WHERE x NOT op y)

Note: *You can only do the second third and fourth example transformations if x and y cannot have a NULL value result.*

Limitations:

None.

ANSI Specifics:

None.

CONNX Specifics:

None.

Example:

The following example selects all cruises where the destination harbor is not a starting point for any other cruise:

```
SELECT cruise_id FROM cruise x
   WHERE NOT EXISTS (SELECT * FROM cruise
            WHERE x.destination_harbor = start_harbor) ;
```

In Predicate

Function:

This predicate tests whether a given value is contained within a specified set of values and returns a tri-state boolean result.

Invocation:

One of the six predicates which constitute a search term.

Syntax:

expression	A valid expression as described in the section Expressions.
NOT	An operator which negates the effect of the membership test.
host variable specification	A valid single host variable specification and its value specifies a set member.
query specification	Contained in parentheses and may be given instead of an explicit list separated by commas.
constant	A valid constant and its value specifies a set member

Description:

The IN predicate is a search expression containing comparison predicates linked by the OR operator.

CONNX processes the IN predicate this way:

- **x IN (1,2,3)**
  x = 1 OR x = 2 OR x = 3

- **x IN (subquery)**
  x = ANY (subquery)

The expression and all members of the set, no matter whether they are explicit or returned as the result of the subquery, must have comparable data types. If the expression or any of the set members evaluate to the NULL value, then the predicate returns the tri-state value of unknown.

The query specification follows the rules for subqueries within a quantified COMPARISON predicate. The subquery may only specify one resultant column in its derived column list, although it may return many different values/rows.

String comparison follows the same rules as specified for the COMPARISON predicate.

Limitations:

None.

ANSI Specifics:

In ANSI compatibility mode, the special USER register is not supported.

CONNX Specifics:

None. SQL 2 Standard allows query expressions and list of expressions as set.

Examples:

The following example selects all skippers who are on cruises starting from BAHAMAS, PANAMA or TRINIDAD:

```
SELECT id_skipper
    FROM cruise
```

```
    WHERE start_harbor IN ( 'BAHAMAS', 'PANAMA', 'TRINIDAD' );
```

The following example identifies all customers who will be starting a cruise from MIAMI:

```
SELECT id_customer
 FROM contract
 WHERE id_cruise IN ( SELECT cruise_id
    FROM cruise
    WHERE start_harbor = 'MIAMI' );
```

Like Predicate

Function:

The LIKE predicate compares a column of a base table or view with a pattern.

Invocation:

One of the six predicates which constitute a search term.

Syntax:

column specification	A column of a base table or view which is to provide the value against which the comparison is to be made. The column must be of data type character-string.
NOT	An operator which negates the result of the LIKE predicate.
pattern	The form to which the column must conform. It can be expressed as either a hard coded constant or a single host variable specification of the data type character string. The use of wildcard characters is supported.
escape character	A single escape character. The wildcard characters themselves can be considered in any pattern matching by preceding them with an escape character.

Description:

The LIKE predicate performs a comparison between the specified column value and a given pattern. If a match is found, the predicate equates to true; otherwise it is false. If the column or the pattern equates to a NULL value, the predicate has an unknown result.

Wildcard Characters

For a true predicate, there needs to be a one-to-one match between the two strings. You can use wildcard characters to make the comparison more flexible.

You can specify wildcard characters anywhere in the pattern.

The wildcard character '_'

"_' takes the place of any single character in the pattern. If a particular position in the string be of no significance, use an underscore character in the pattern to mask it out.

For example, with a pattern of `ABCDE', only `ABCDE' will result in `true'. However, a pattern of `AB_DE' will not only give a true result for `ABCDE' as before but also for `ABZDE' or, in fact, for any string that is five characters long and starts with `AB' and ends with `DE'. Note the comparison of `ABZZDE' would fail for this pattern as an extra character has been introduced.

The wildcard character '%'

'%' takes the place of zero or more characters in the pattern.

If the pattern were specified as `AB%DE', then a column value of `ABZZDE' would give a true result as would a string of any length that started with `AB' and finished with `DE'.

If the pattern is not of an identical size to the column, no space padding takes place and so, no match will be found. This is opposite to a normal COMPARISON predicate.

For example, if the column first_name has provision for 10 characters and contains the value 'TIMOTHY' then the following COMPARISON predicate will evaluate to true:

```
WHERE first_name = 'TIMOTHY'
```

However, the following LIKE predicate will evaluate to false:

```
first_name LIKE 'TIMOTHY'
```

This is because no space padding takes place. The following two LIKE predicates would evaluate to true:

```
first_name LIKE 'TIMOTHY   '

first_name LIKE 'TIM%'
```

Note: *In the above case, the wildcard character % would also result in a row containing the value 'TIMMY ', for example, being found.*

The escape character '^'

If either or both of the wildcard characters are required to have their actual meaning, then specify an escape character. An escape character is any single character which must precede either the `%'or the `_' thus signifying that the following wildcard character is to be taken literally.

For example, if an exact match for the string 'AB_DE' was required and the escape character had been defined as '^', then the pattern would have to be specified as 'AB^_DE'.

Limitations:

Should the column reference a view, then this viewed column must be derived exclusively from a column of a base table. This applies to all three modes.

ANSI Specifics:

None.

CONNX Specifics:

None.

Example:

The following example selects a person whose name ends with the characters 'ann' :

```
SELECT person_id
   FROM person
   WHERE first_name_1 LIKE '%ann';
```

Null Predicate

Function:

The NULL predicate tests a particular column to see if it contains the NULL value.

Invocation:

One of the six predicates which constitute a search term.

Syntax:

expression	May reference any valid column even if it does not support NULL values.
NOT	Operator which negates the result of the predicate.

Description:

This predicate tests to see if a given expression evaluates to a NULL value. The NULL predicate can only return either true (column IS NULL) or false (column holds a definite value). The result can never be unknown.

Limitations:

None.

ANSI Specifics:

None.

CONNX Specifics:

None.

Example:

The following example determines if any cruises were offered for which no reservations have been made:

```
SELECT ID_CRUISE
   FROM CRUISE, CONTRACT
   WHERE CRUISE_ID = ID_CRUISE AND DATE_RESERVATION IS NULL;
```

Search Condition

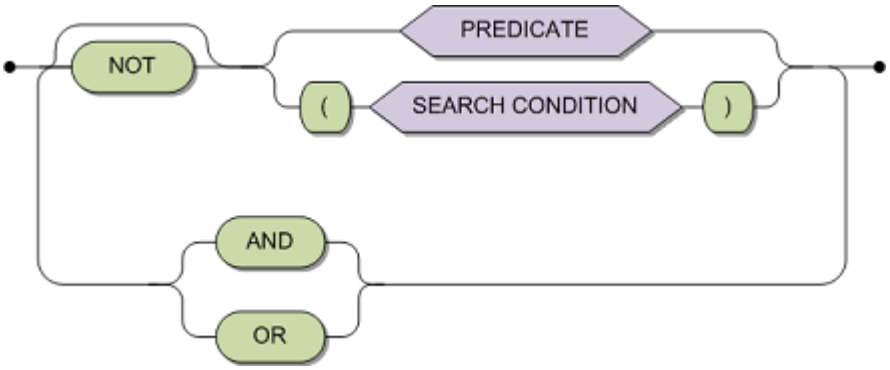
Function:

A search condition is a multi-predicate boolean expression which defines whether a candidate row or group is to be included in the resultant table of the query, depending upon whether the condition equates to true.

Invocation:

A search condition may appear as the body of a WHERE clause in either a query specification or a searched DELETE or UPDATE statement and as the body of a HAVING clause.

Syntax:



predicate	The basis of a search condition and constitutes one of the possible search terms. All predicates equate to true, false or unknown.
search_condition	A recursive construction enabling complex search conditions. Complex recursive constructions must be enclosed in brackets. As they are built upon predicates, search conditions also equate to true, false or unknown and constitute the other possible search term.
NOT	An operator which negates the result of either the predicate or the included search condition.
AND/OR	Boolean operators which combine predicates and parenthesized search conditions to form a final search condition.

Description:

If a search condition in a WHERE clause equates to true, then the evaluated candidate row is considered to be a member of the resultant table. Otherwise it is rejected.

If the search condition is in a HAVING clause, then the candidate group is included if the search condition equates to true.

Individual search terms of the search condition can be combined using the boolean operators AND or OR. The order of precedence of the operators is NOT followed by AND followed by OR. Operators of the same precedence are evaluated from left to right. Search terms which are search conditions are evaluated first.

Because predicates can result in the state UNKNOWN, the operators are able to evaluate `tri-state logic'. The truth tables are as follows:

NOT	TRUE	FALSE	UNKNOWN
	FALSE	TRUE	UNKNOWN
AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN
OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

Limitations:

None.

ANSI Specifics:

None.

CONNX Specifics:

None.

Example:

The following example selects the IDs of all people who have a surname which starts with the letter W, and are not from the city of DERBY:

```
SELECT person_id
   FROM person
   WHERE surname LIKE 'W%' AND NOT city = 'DERBY';
```

The following example deletes all contract data for those persons who made a reservation on the 4th of September 1991, where the cruise does not cost more than 2000 or the amount deposited is not more than 700:

```
DELETE FROM contract
   WHERE date_reservation = 19910904
      AND ( NOT price > 2000 OR NOT amount_deposit > 700 );
```

The following example determines the average price of all cruises that go to MARMARIS, start from RHODOS or FETHIYE, and have a starting time of 16.00 or 17.00:

```
SELECT start_harbor, destination_harbor, start_time,
AVG(cruise_price)
   FROM cruise
   WHERE destination_harbor = 'MARMARIS'
      GROUP BY start_time,start_harbor
```

```
HAVING (start_harbor = 'RHODOS' OR start_harbor = 'FETHIYE')
AND (start_time = 16 OR start_time = 17);
```

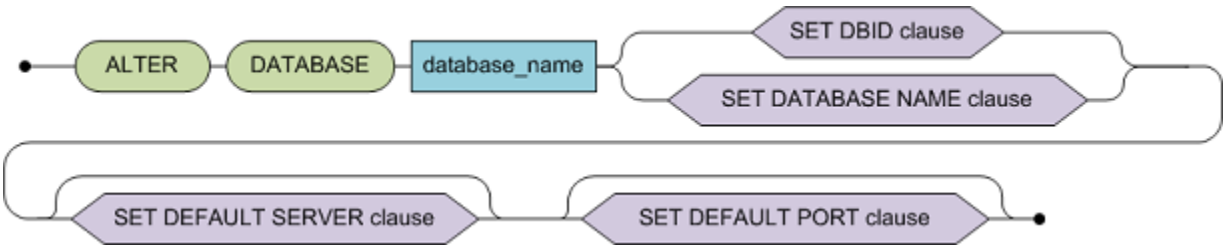
Primary SQL Commands

ALTER DATABASE

Function

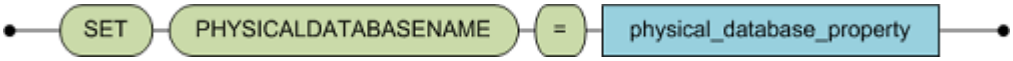
ALTER DATABASE updates the database's Connection Properties.

Syntax

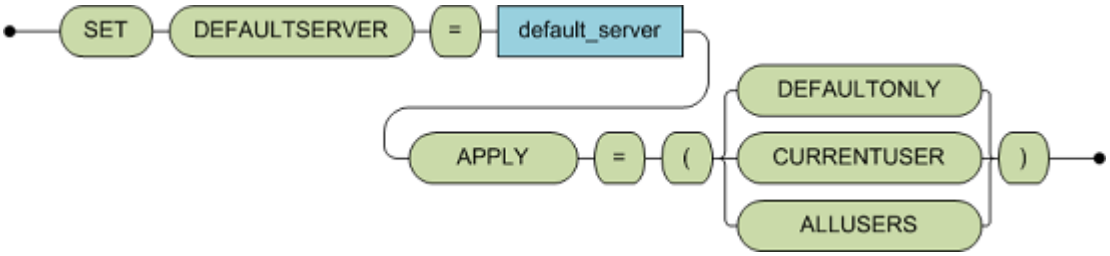


SET DBID clause:

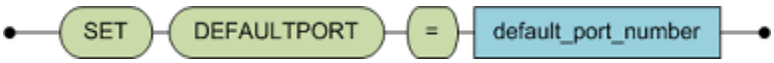
SET DATABASE NAME clause:



SET DEFAULT SERVER clause:



SET DEFAULT PORT clause:



Description

ALTER DATABASE changes the database connection attributes, including the physical database name, the port, and (for Adabas databases) the dbid. ALTER DATABASE has two formats:

- For Adabas only:

```
ALTER DATABASE <dbname> [set dbid=<number>]  | [set
   defaultserver=<name> apply=(allusers, currentuser, nouser)] | [set
   defaultport=<port#>]
```

where

dbname = Name of the Database (example localhost, Sales, ...)

set dbid = Number from 1-255

set defaultserver = default server

apply = level of affected users

allusers = change apply to all who access this database (default values)

currentuser = changes only apply to the current (logged in) user

defaultonly = no changes (only applies to future users)

set defaultport = default port

- For all other databases:

```
ALTER DATABASE <dbname> [set physicaldatabasename = <physical database
   property> | [set defaultserver=<name> apply=(allusers, currentuser,
   nouser)] | [set defaultport=<port#>]
```

where

dbname = Name of the Database (example localhost, Sales, ...)

set physicaldatabasename = physical database property

set defaultserver = default server

apply = level of affected users

allusers = change apply to all who access this database (default values)

currentuser = changes only apply to the current (logged in) user

defaultonly = no changes (only applies to future users)

set defaultport = default port

Limitations

- Only administrators (members of the group CONNXCDDAdministrators) can use ALTER DATABASE.
- Both the source database (dbname) and the target database (dbid or physicaldatabasename/defaultserver) must exist prior to issuing this command.
- Every table mapped in the CDD in the source database must be in the target database.
- Every shared source and target table mapped in the CDD must have identical structure.
- defaultport cannot be the only attribute that changes.
- Both the source and target databases must have the same type. You cannot use this command to change the fundamental database type (for example, no RMS to VSAM).

ANSI Specifics

The ALTER DATABASE statement is not part of the ANSI Standard.

Examples

1. Change the dbid to 57

```
Alter Database EMPLOYEES set dbid=57
```

2. Change the default server to PRODUCTION, and change any users that are registered for EMPLOYEES to connect to the PRODUCTION server.

```
Alter Database EMPLOYEES set defaultserver=PRODUCTION apply=allusers
```

3. Change the default server to PRODUCTION, and change only the user that is logged in to point to PRODUCTION for their database connection to EMPLOYEES.

```
Alter Database EMPLOYEES set defaultserver=PRODUCTION apply=currentuser
```

4. Change the default server to PRODUCTION, but ensure this change does not affect users that are pointing to the prior default server.

```
Alter Database EMPLOYEES set defaultserver=PRODUCTION apply=defaultonly
```

5. Change the default port to 7505

```
Alter Database EMPLOYEES set defaultport=7505
```

6. Change the physical database name to cnxdir:connx_rdb_examples

```
Alter Database EMPLOYEES set physicaldatabasename =
   cnxdir:connx_rdb_examples
```

ALTER TABLE (Adabas only)

Function

Add column to an existing Adabas file.

Syntax

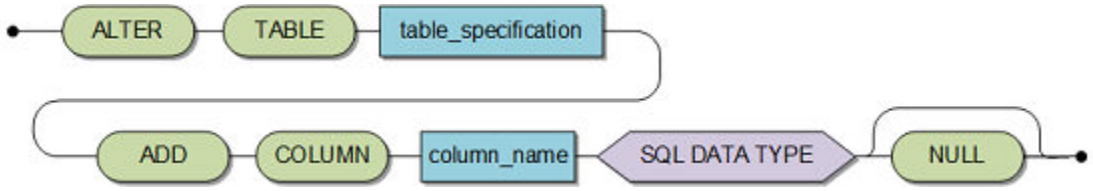


Table Specification:

See Table Specification.

SQL Data Type:

See SQL Data Types under SQL Language Elements.

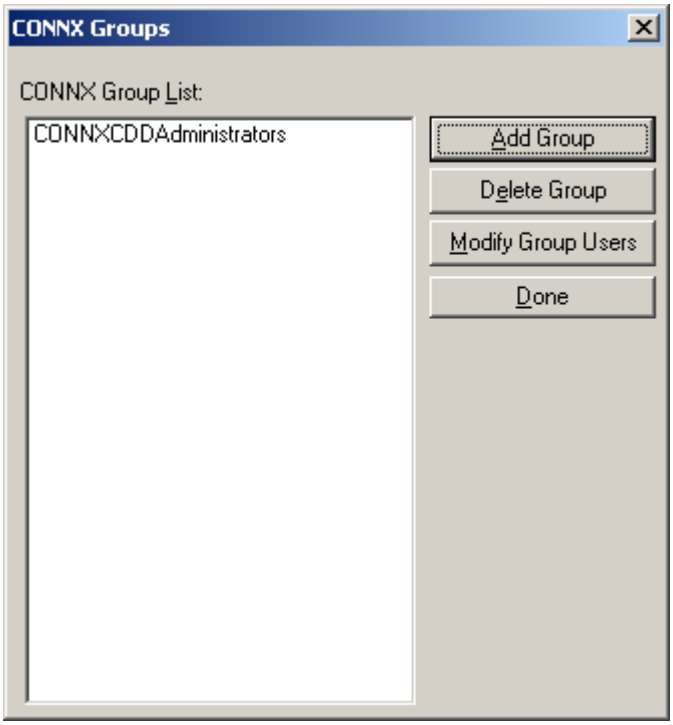
column_name	SQL identifier for the new column.

Description

ALTER TABLE adds the specified column to the Adabas file. Any column added by ALTER TABLE is nullable.

Limitations

Only the DBA may execute this statement for any user. All other users can only change their user information.



***Caution**: This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Before you execute this statement, complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.*

ANSI Specifics

The ALTER TABLE statement is not part of the ANSI Standard.

Examples

The following example shows a user adding a column called test to an Adabas file called ORDERS.

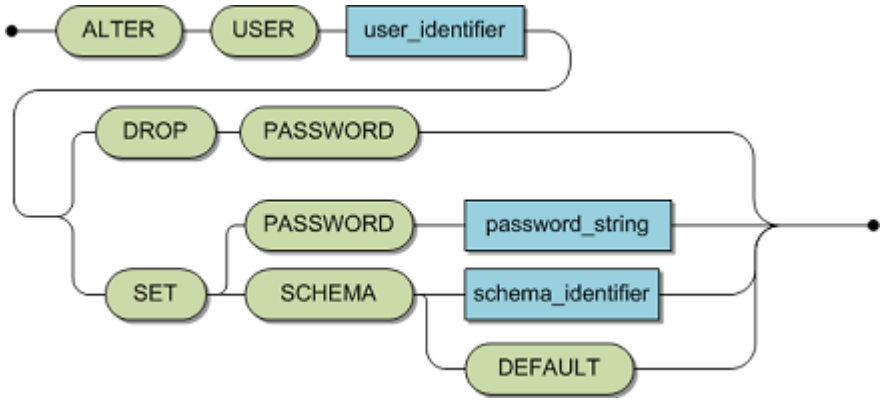
```
ALTER TABLE ORDERS ADD COLUMN TEST CHAR(40) NULL;
```

ALTER USER

Function

ALTER USER changes an existing user's attributes.

Syntax



user_identifier	Unique identifier for an existing user.
password_string	User password.
schema_identifier	Identifier for an existing or non-existing schema.

Description

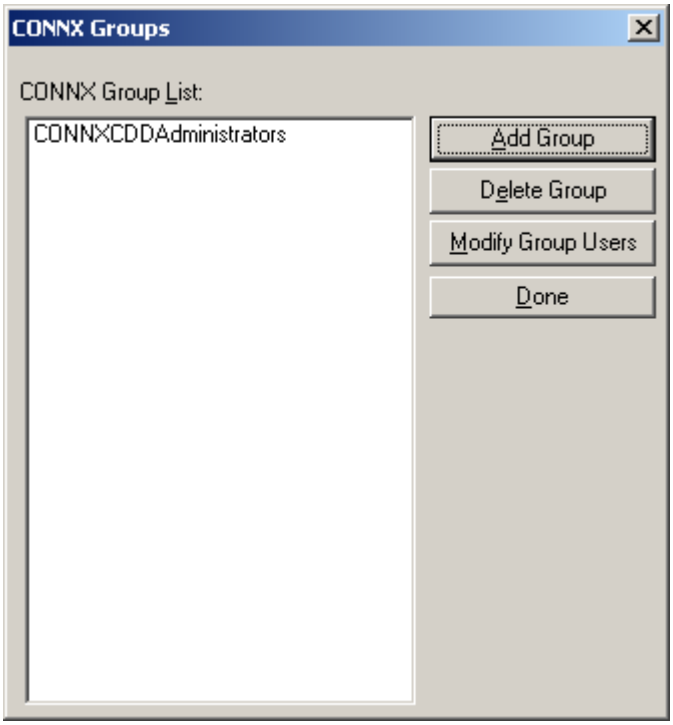
DROP PASSWORD deletes the password requirement. After using DROP PASSWORD, users can only access CONNX with their user ID.

SET PASSWORD adds or changes an existing CONNX user's integrated security password. This password should not be confused with the database password for data sources that support authentication.

SET SCHEMA can change the default schema's setting after the user issues a CONNECT command to CONNX. The default is the user identifier. The database and schema are checked to see if they exist.

Limitations

Only the DBA may execute this statement for any user. All other users can only change their user information.



***Caution**: This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Before you execute this statement, complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.*

ANSI Specifics

The ALTER USER statement is not part of the ANSI Standard.

Examples

The following example shows a user changing their own password.

```
ALTER USER TIM SET PASSWORD 'XIYIZ';
```

The following example shows the DBA deleting the password for user PETER.

```
ALTER USER PETER DROP PASSWORD;
```

COMMIT

Function

The COMMIT statement terminates a transaction and makes permanent all changes that were made to the database during the terminated transaction.

Syntax

Description

The COMMIT statement terminates the current transaction and starts a new transaction. All changes to the database that have been made during the terminated transaction are made permanent. All cursors that have been opened during the current transaction are closed.

ANSI Specifics

The keyword WORK is mandatory.

CONNX Specifics

Example

The following example commits all changes made to the database in the current transaction.

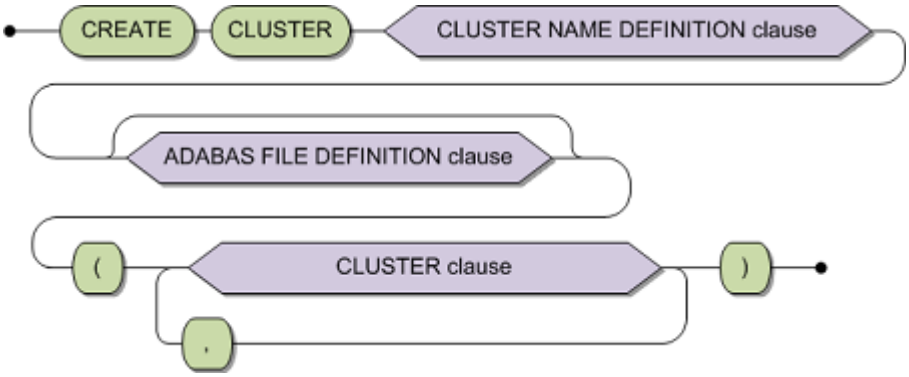
```
COMMIT WORK;
```

CREATE CLUSTER

Function

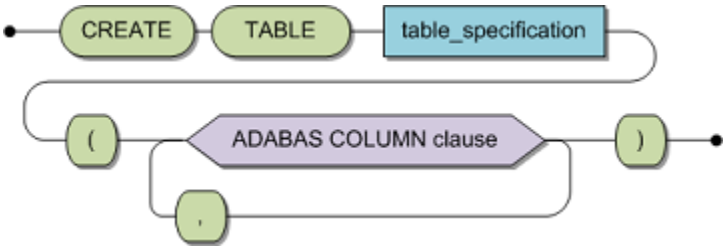
The CREATE CLUSTER statement is used to combine a number of base tables in one internal table.

Syntax



cluster name definition clause	Identification of the table cluster to be created. Optionally with specification of the assigned Adabas database. See Cluster Name Definition.
adabas file definition clause	See Common Elements, section Adabas File Definition.
cluster_element	A base table in the described cluster.

CLUSTER Clause:



table_specification	Identifier of the subtable to be created for this cluster, qualified by the schema identifier. See Table Specification.
Adabas column clause	See Adabas Column Clause.

Description

A CREATE CLUSTER statement is used to combine a set of base tables into one Adabas file.

Each subtable within the CLUSTER represents either a PE group or an MU field. It is also possible to group MU's together into one single subtable, this assumes that all MUs have the same number of occurrences and that when accessing them, the occurrence number of each MU will be equal.

Limitations

The DBA can execute this statement for all users. All other users can use this statement only in a schema owned by the user.

The column attributes/table clause SHORTNAME definition may not be specified in this statement.

A CREATE CLUSTER statement will always represent tables of level one as a PE group within Adabas. Following rules apply:

1. Foreign keys reference only primary keys. A subtable contains exactly one foreign key.
2. The same rules apply for the columns, constraints and indexes of the master table as for a CREATE TABLE statement.
3. Columns which are not an element of a foreign key and not of a SEQNO type are called data columns. The limitations under rules 4 - 7 apply to data columns in subtables.
4. The data columns of a level 1 table correspond only to fields of a single PE group.
5. The data columns of a level 2 table correspond to MU fields within a specific PE group - the group containing those fields which the data columns in the referenced table correspond to.
6. Not more than one data column may correspond to each field (with rotated fields, each subscript counts as its own field).
7. With parallel MU fields, it is assumed that in all Adabas records, the respective counter values are the same.
8. For x=1 or x=2, a unique constraint of a level x table encompasses the elements of the foreign keys and a column of the type SEQNO(x). Other unique constraints on subtables are not allowed.
9. For indexes to subtables, the same rules apply as for level-0 tables, plus the following additional constraints:
10. HAVING UNIQUE INDEX is not allowed. In order to model the Adabas UQ option, UQINDEX is used.

    Note: *A unique constraint is defined as either a UNIQUE or PRIMARY KEY constraint.*

11. All level 0 columns must be grouped within one CREATE TABLE of a cluster.

Note: *In the case of a PE data structure containing MU fields only, it is necessary to use an Adabas short name on the SEQNO(1) of the PE-subtable.*

Caution:

This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Therefore, before executing this statement, it is strongly recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.

ANSI Specifics

The CREATE CLUSTER statement is not part of the Standard.

Adabas SQL Gateway Embedded SQL Specifics

None

Example

The following example creates the cluster city_guide.

```
CREATE CLUSTER city_guide
(
  CREATE TABLE states
```

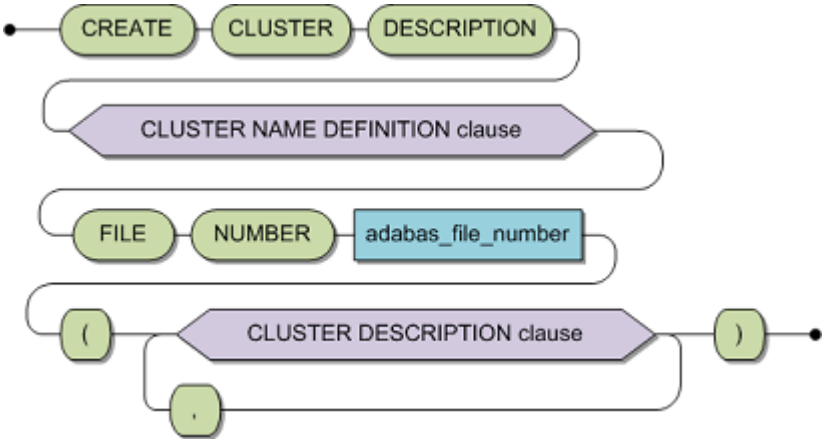
```
    (
      abbreviation     CHAR (2)  PRIMARY KEY NOT NULL DEFAULT ADABAS,
      state_name       CHAR (20) UNIQUE NOT NULL DEFAULT ADABAS,
      capital          CHAR (20) INDEX state_capital,
      population       INT
    ),
   CREATE TABLE cities
    (
      state_abbrev     CHAR (2)  UQINDEX NOT NULL DEFAULT ADABAS,
      city_seqno       SEQNO (1) NOT NULL,
      city_name        CHAR (20),
      population       INT,
      PRIMARY KEY (state_abbrev, city_seqno),
      FOREIGN KEY (state_abbrev)
      REFERENCES states(abbreviation),
      UQINDEX city_state (city_name, state_abbrev)
    ),
   CREATE TABLE buildings
    (
      state_abbrev     CHAR (2) UQINDEX NOT NULL DEFAULT ADABAS,
      city_seqno       SEQNO(1)  NOT NULL,
      building_seqno   SEQNO(2)  NOT NULL,
      building_name    CHAR (20) NOT NULL SUPPRESSION,
      height           INT       NOT NULL SUPPRESSION,
      PRIMARY KEY (state_abbrev, city_seqno, building_seqno),
      FOREIGN KEY (state_abbrev, city_seqno)
      REFERENCES  cities (state_abbrev, city_seqno)
    ),
  CREATE TABLE places
    (
      state_abbrev     CHAR (2) UQINDEX NOT NULL DEFAULT ADABAS,
      city_seqno       SEQNO(1)  NOT NULL,
      place_name       CHAR (20) NOT NULL SUPPRESSION,
      FOREIGN KEY (state_abbrev, city_seqno)
      REFERENCES  cities (state_abbrev, city_seqno)
    )
  );
```

CREATE CLUSTER DESCRIPTION (Adabas only)

Function

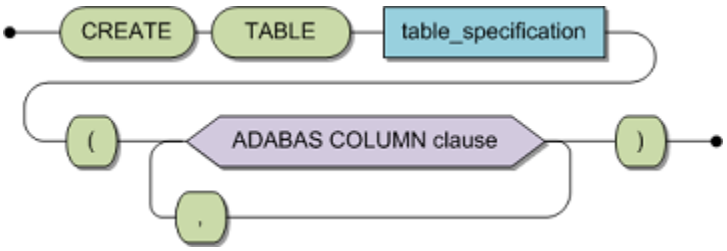
This statement introduces an existing Adabas file including MU/PE fields to the SQL environment. CREATE CLUSTER DESCRIPTION enables users to import metadata from Adabas scripts using DDL.

Syntax



cluster name definition clause	Identification of the table cluster to be created. Optionally with specification of the assigned Adabas database. See Cluster Name Definition.
adabas_file_number	The Adabas file number.

CLUSTER DESCRIPTION clause:



table_specification	The base tables (master tables and subtables) of the described cluster. See Table Specification.
adabas column clause	Defines the columns of the table. See Adabas Column Clause.

Description

CREATE CLUSTER DESCRIPTION defines the description of an existing Adabas file that contains Adabas multiple value fields (MU) and/or periodic groups (PE).

Limitations

The DBA can execute this statement for all users. All other users can use this statement only in a schema owned by the user.

The following rules apply:

1. Foreign keys reference only unique constraints. A sub-table contains exactly one foreign key.
2. The same rules apply for the columns, constraints and indexes of the master table as for a CREATE TABLE DESCRIPTION statement.
3. Columns which are not an element of a foreign key and not of a SEQNO type are called data columns. The limitations under rules 4 - 7 apply to data columns in subtables.
4. The data columns of a level 1 table correspond either to MU fields which do not lie within a PE group, or to fields within a single PE group.
5. The data columns of a level 2 table correspond to MU fields within a specific PE group - the group containing those fields which the data columns in the referenced table correspond to.
6. Not more than one data column may correspond to each field (with rotated fields, each subscript counts as its own field).
7. With parallel MU fields,CONNX assumes that the respective counter values in all Adabas records are the same.
8. For x=1 or x=2, a unique constraint of a level x table encompasses the elements of the foreign keys and a column of the type SEQNO(x). Other unique constraints on subtables are not allowed.
9. For indexes to subtables, the same rules apply as for level-0 tables, plus the following additional constraints:

   HAVING UNIQUE INDEX is not allowed. In order to model the Adabas UQ option, UQINDEX is used.

   Note: *A unique constraint is defined as either a UNIQUE or PRIMARY KEY constraint.*
10. All level 0 columns must be grouped within one CREATE TABLE of a cluster.

The BLOCK SIZE has the following limitations:

- default value= 7
- minimum value = 1
- maximum value = 191

Note: *In the case of a PE data structure containing MU fields only, use an Adabas short name on the SEQNO(l) of the PE-subtable.*

Caution: *This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Therefore, before executing this statement, it is strongly recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.*

ANSI Specifics

This statement is not part of the ANSI standard.

Adabas SQL Gateway Specifics

Example

```
CREATE CLUSTER DESCRIPTION city_guide
```

```
FILE NUMBER 134
(
CREATE TABLE DESCRIPTION states (
     abbreviation      SHORTNAME 'AA' PRIMARY KEY DEFAULT ADABAS,
     state name        SHORTNAME 'AB' UNIQUE NOT NULL DEFAULT ADABAS,
     capital           SHORTNAME 'AC' INDEX,
     population        SHORTNAME 'AD'
     ),
CREATE TABLE DESCRIPTION cities (
     state_abbrev      SHORTNAME 'AA'
     city segno        SEQNO(1) NOT NULL DEFAULT ADABAS
     city name         SHORTNAME 'BA' INDEX NULL SUPPRESSION,
     population        SHORTNAME 'BB' NULL SUPPRESSION
     PRIMARY KEY (state abbrev, city_seqno),
     FOREIGN KEY (state abbrev) REFERENCES states,
     UQINDEX ( city name, state abbrev)
     ),
CREATE TABLE DESCRIPTION buildings (
     state_abbrev       SHORTNAME 'AA',
     city_segno         SEQNO(1) NOT NULL DEFAULT ADABAS,
     building_segno     SEQNO(2) NOT NULL DEFAULT ADABAS,
     building name      SHORTNAME 'CA' NOT NULL SUPPRESSION,
     height             SHORTNAME 'CB' NOT NULL SUPPRESSION,
     PRIMARY KEY (state abbrev, city_segno, building segno),
     FOREIGN KEY (state abbrev, city segno) REFERENCES cities
     ),
CREATE TABLE DESCRIPTION places
     state abbrev       SHORTNAME 'AA' NOT NULL,
     city_segno         SEQNO(l) NOT NULL DEFAULT ADABAS,
     place name         SHORTNAME 'DA' NULL SUPPRESSION,
     FOREIGN KEY (state abbrev, city_segno) REFERENCES cities
```

Below is the corresponding Adabas FDT definition:

Level	I	Name	I	Length	I	Format	I	Options	I Comment
1	I	AA	I	2	I	A	I	DE, UQ	I states, abbreviations
1	I	AB	I	20	I	A	I	DE, UQ	I states.state_name
1	I	AC	I	20	I	A	I	DE, NC	I states.capital
1	I	AD	I	4	I	F	I	PE	I states.population
1	I	B0	I		I		I	DE, NU	I cities
2	i	BA	I	20	I	A	I	NU	I cities.city_name
2	i	BB	I	4	I	F	I	NU, MU	I cities.population
2	I	CA	I	20	I	A	I	NU, MU	I buildings.building_name

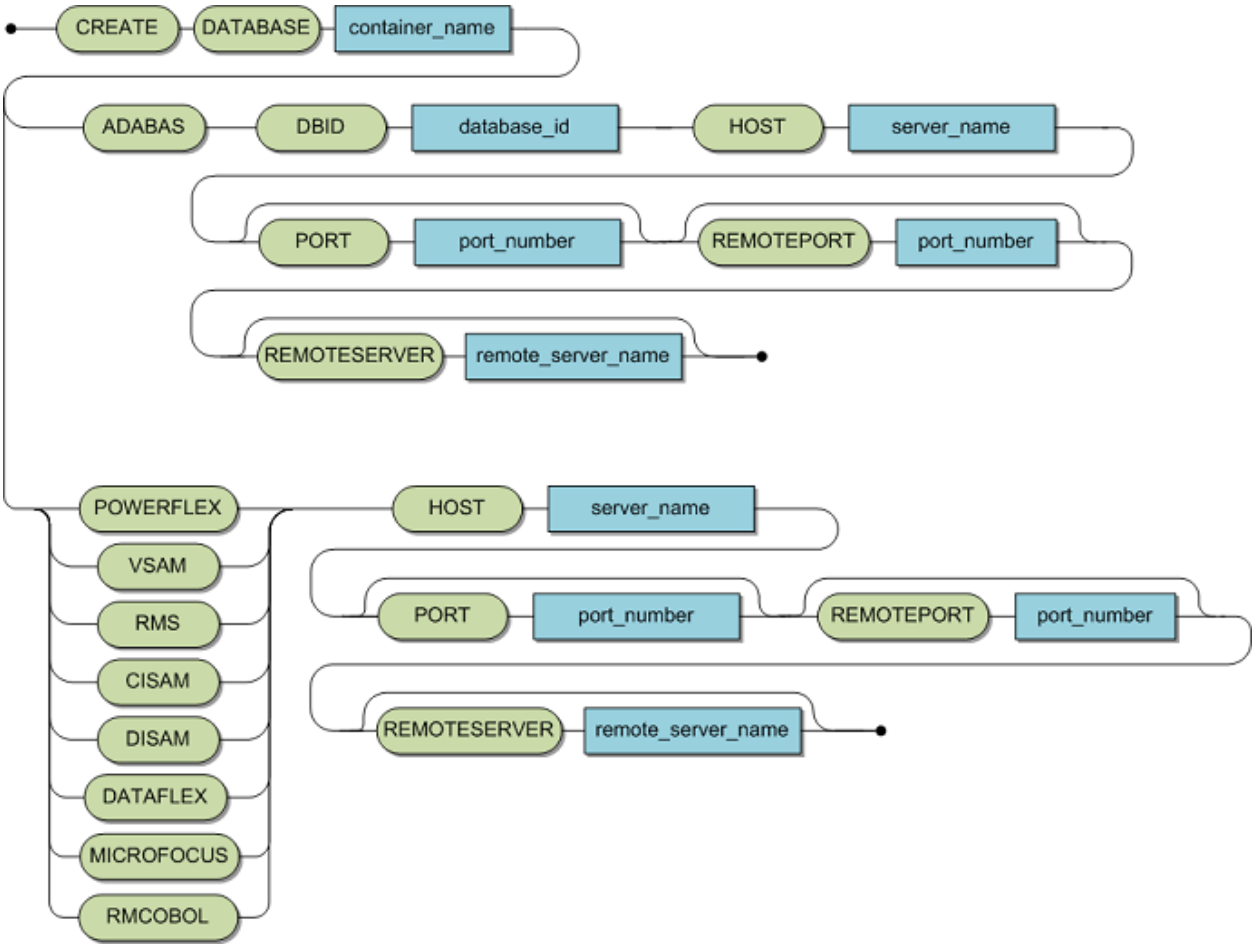
2	I	CB	I	2	I	F	I	NU, MU	I buildings.height
2	I	DA	I	20	I	A	I	NU, MU	I places.place_name
Type Fmt	I	Name	I	Length	I	Format	I	Options	I Parent field(s)
Super	I	X1	I	22	I	A	I	NU, UQ, PE	I BA (1 - 20) A
	I		I		I		I		I AA (1 - 2) A

CREATE DATABASE

Function

This statement creates a new database entry in a data dictionary using DDL..

Syntax



container_name	The logical name for the database in the data dictionary.
database_id	The ADABAS DBID.
server_name	The machine the database server is running on.
port_number	The TCP/IP port that the CONNX server is listening on.
remote_server_name	The name of the server running the enterprise server service.

CREATE DATABASE only applies to ADABAS, VSAM, RMS, CISAM, DISAM, DATAFLEX or POWERFLEX, MICROFOCUS, and RMCOBOL databases.

CREATE DATABASE does not actually create a database; instead it links CONNX to an existing database for further processing.

With CREATE DATABASE, you can create a link to an Adabas database using a script instead of using the CONNX Data Dictionary (CDD) Manager.

Execute a DISCONNECT prior to using the DATABASE container.

Execute a CREATE DATABASE before a CONNECT; otherwise the CONNECT will fail.

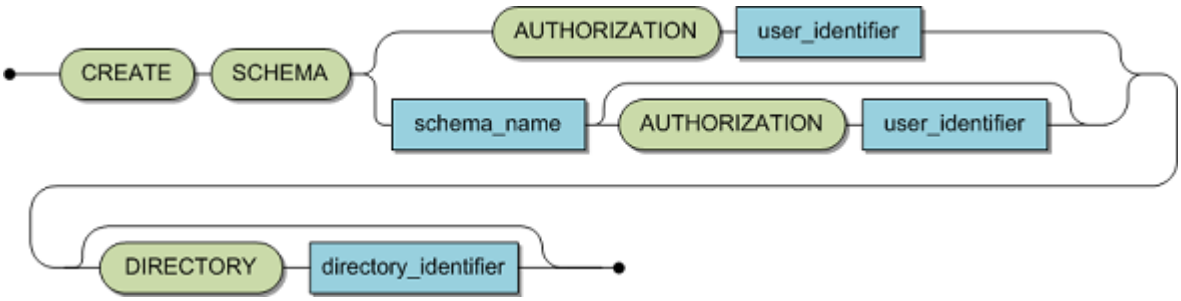
After executing a CREATE DATABASE, you can execute CREATE TABLE DESCRIPTION to complete the CDD.

CREATE SCHEMA

Function

CREATE SCHEMA creates an SQL schema which serves as a logical container for the subsequent creation of catalog resident objects such as tables and views.

Syntax



schema_name	A valid schema identifier representing the schema to be created.
user_identifier	A valid user identifier of an existing user.
directory_identifier	A valid directory to be used for file creation for the data source used in association with this schema.

Description

CREATE SCHEMA causes a schema to be entered into the catalog. The name of the schema is:

- explicitly provided as the schema identifier in the CREATE SCHEMA statement or
- derived from the user identifier, if the schemas identifier has been omitted.

A schema must have an owner. This owner can be explicitly specified in the AUTHORIZATION clause as a user identifier. If the owner is not explicitly specified, the user identifier is derived from the user identifier of the statement executor. In both cases, the resulting user identifier must be the same as a known server user.

If the statement is invoked statically, any user identifier does not have to exist in the catalog until statement execution.

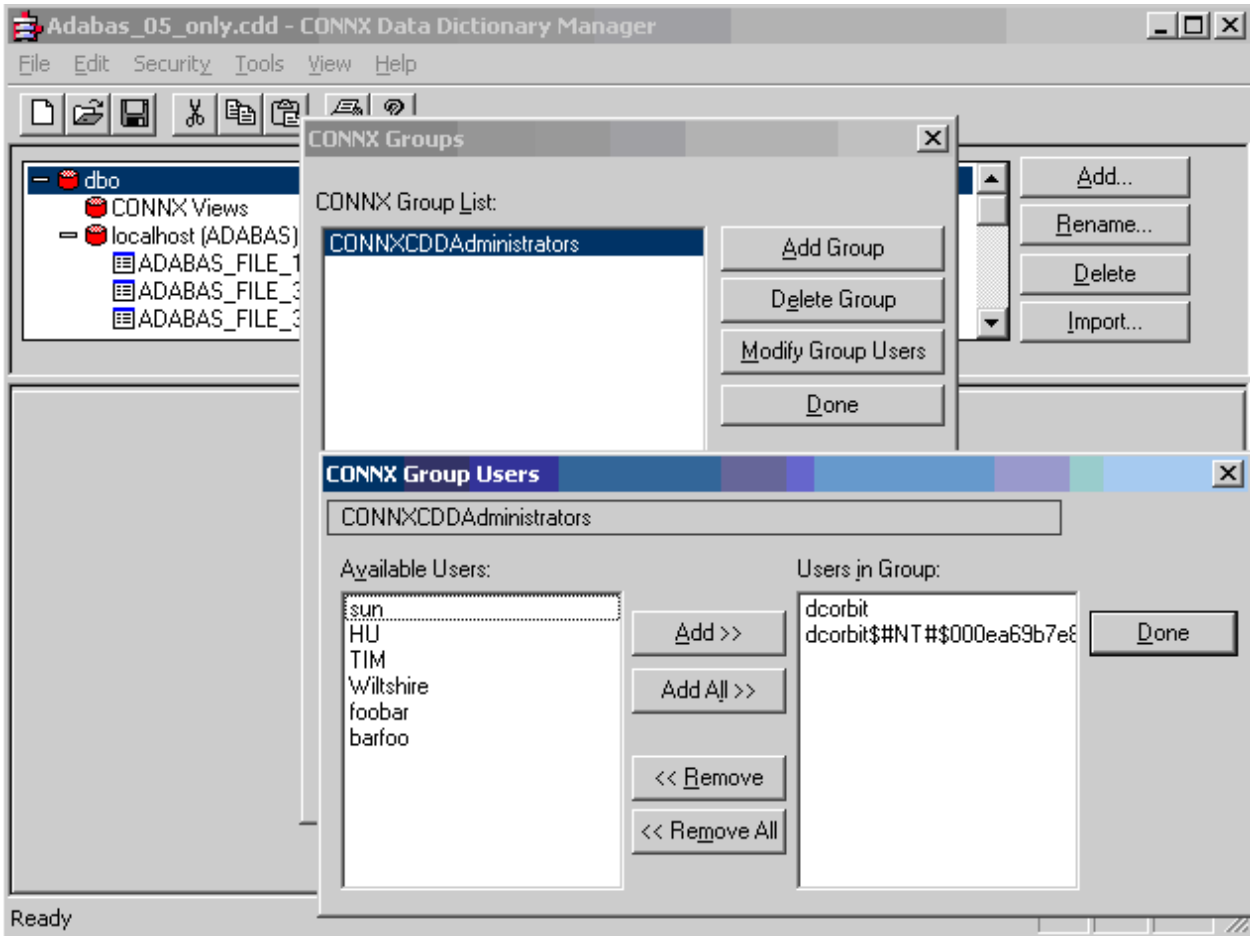
Limitations

The resulting schema identifier must be unique within the catalog.

The user identifier must be equal to an already defined user identifier as defined using a CREATE USER statement.

The contents of dir_identifier will be ignored if the database is not a ROSIO data source.

Note: *The CREATE SCHEMA statement can only be executed by the DBA. The DBA must be a member of the group CONNXCDDAdministrators:*



***Caution**: This statement is not subject to transaction logic. An implicit COMMIT is performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK is performed. Therefore, before executing this statement, it is strongly recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.*

ANSI Specifics

The CREATE SCHEMA statement provided by CONNX is a reduced version of that specified in the SQL-2 standard. In particular, it is not possible to specify the object that belongs to a schema in this statement (i.e., base tables, views and constraints).

Example

The following example creates a schema with the name Wiltshire and assign it to the owner TIM.

```
CREATE SCHEMA Wiltshire AUTHORIZATION 'TIM' ;
```

The following example creates a schema with the name TIM and assigns it to the owner TIM.

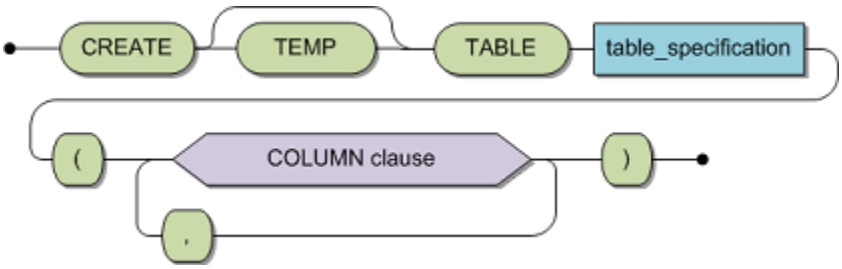
```
CREATE SCHEMA AUTHORIZATION 'TIM' ;
```

CREATE TABLE

Function

CREATE TABLE command will create the specified table on the specified target database. For non-relational databases, CONNX creates a data structure on the target system that closely approximates the requested relational table..

Syntax



COLUMN Clause:

Table Specification:

See Table Specification.

SQL Data Type:

See SQL Data Types under SQL Language Elements.

Note: CREATE TABLE is currently not supported by CONNX for VSAM or Codasyl DBMS.

Using the TEMP keyword, CREATE TABLE can create temporary tables. Temporary tables can contain indexes; they support all the standard SQL data types, including CLOBs BLOBs, and unicode data. Temporary tables are private to the connection on which they were created and will be deleted when the connection is closed.

Example:

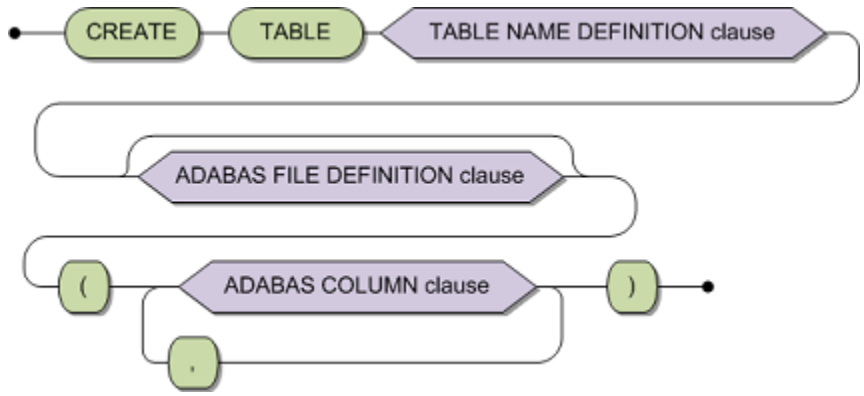
CREATE table test (companyname char(20), companyid integer, companybudget double)

CREATE TABLE (Adabas only)

Function

The CREATE TABLE statement defines a base table in the catalog and physically creates the table in the Adabas nucleus.

Syntax



table_name_definition	Identification of the table to be created. Optionally with specification of the assigned database. See Table Name Definition.
adabas file definition clause	See Adabas File Definition.
adabas column clause	See Adabas Column Clause.

Description

The CREATE TABLE statement defines the logical structure of a base table. From this logical structure the physical structure is derived, which is represented as an Adabas FDT in the related database.

Compiling the statement does not create a table. Therefore, other statements cannot reference the table.

Temp Tables

Using the TEMP keyword, CREATE TABLE can create temporary tables.

Temporary tables can contain indexes; they support all the standard SQL data types, including CLOBs BLOBs, and unicode data. Temporary tables are private to the connection on which they were created and will be deleted when the connection is closed.

Limitations

The DBA can execute this statement for all users. All other users can use this statement only in a schema owned by the user.

The table specification must be unique within a schema. A table must have at least one column.

Adabas SQL Gateway adheres to the relational database theory where tables are strictly two dimensional and, therefore, periodic groups and multiple fields which are available in Adabas are not permitted for tables specified with the CREATE TABLE statement. For cases where nested data structures, i.e., MU/PE fields are involved, a special statement, CREATE CLUSTER, is provided.

SHORTNAME and UQINDEX cannot be specified in this statement. Referential constraints are not supported.

Caution: This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Therefore, before executing this statement, it is strongly recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.

ANSI Specifics

The column default value of "ADABAS" is not part of the ANSI SQL Standard.

The current <authorization identifier> must be equal to the <authorization identifier> that owns the schema identified by the implicit or explicit <schema name> of the <table name>. For SQL-2, this access rule is valid only if a <table definition> is contained in a <module>. This access rule is valid for every <table definition> with one exception. For the current <authorization identifier>:='DBA', this access rule does not apply. The <authorization identifier>:='DBA' has the privilege for a <table definition> regardless who owns the schema identified by the implicit or explicit <schema name> of the <table name>.

Adabas SQL Gateway Specifics

n/a

Example

The following example defines and creates the table CONTRACT.

```
CREATE TABLE contract ( contract_id
    integer index ind_contract not null unique,
    price            numeric (13,3) not null,
    date_reservation  integer,
    date_booking      integer,
    date_cancellation integer,
    date_deposit      integer,
    amount_deposit    integer,
    date_payment      integer,
    amount_payment    numeric (13,3),
    id_customer       integer not null,
    id_cruise         integer not null );
```

CREATE TABLE DESCRIPTION (Adabas only)

Function

This option enables users to import metadata from Adabas using SQL..

Syntax

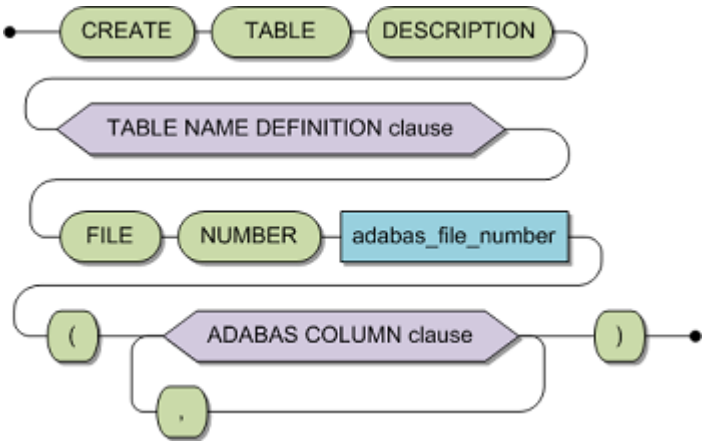


table name definition clause	Identification of the table to be created. Optionally with specification of the assigned Adabas database. See Table Name Definition.
adabas_database_number	The Adabas database number. This is optional. The Adabas database number can also be determined by the catalog used in the table name.
adabas_file_number	The Adabas file number.
adabas column clause	See Adabas Column Clause.

Description

CREATE TABLE DESCRIPTION specifies already existing Adabas files to the SQL environment in the catalog.

The statement consists of a table specification and a list of table elements . If the table specification contains a schema identifier, then the table identifier will be explicitly qualified, otherwise CONNX uses the current default schema identifier.

Compiling the statement does not create the table. Other statements can not reference the table specified in the CREATE TABLE DESCRIPTION statement until the statement has been successfully executed.

If the statement is invoked statically, then during pre-compilation, the schema need not exist in the catalog. For successful execution, however, the schema must exist in the catalog, regardless of how it is invoked.

This statement may also use the technique of rotating a MU/PE's fields into base columns. This allows each element of a MU/PE field, to be referenced as a separate column within a base table.

Because this statement executes on an existing Adabas file, you can specify minimal information; the SQL compiler will generate the rest. You must be specify the column identifier and the Adabas short name for this column; all other information will be generated from the underlying Adabas file.

Limitations

The DBA can execute this statement for all users. All other users can use this statement only in a schema that they own.

The following rules apply:

1. Foreign keys reference only unique constraints. A sub-table contains exactly one foreign key.
2. The same rules apply for the columns, constraints and indexes of the master table as for a CREATE TABLE DESCRIPTION statement.
3. Columns which are not an element of a foreign key and not of a SEQNO type are called data columns. The limitations under rules 4 - 7 apply to data columns in sub-tables.
4. The data columns of a level 1 table correspond either to MU fields which do not lie within a PE group, or to fields within a single PE group.
5. The data columns of a level 2 table correspond to MU fields within a specific PE group - the group containing those fields which the data columns in the referenced table correspond to.
6. No more than one data column may correspond to each field (with rotated fields, each subscript counts as its own field).
7. With parallel MU fields, CONNX assumes that in all Adabas records, the respective counter values are the same.
8. For x=1 or x=2, a unique constraint of a level x table encompasses the elements of the foreign keys and a column of the type SEQNO(x). Other unique constraints on subtables are not allowed.
9. For indexes to subtables, the same rules apply as for level-0 tables, plus the following additional constraints:

   HAVING UNIQUE INDEX is not allowed. In order to model the Adabas UQ option, UQINDEX is used.

   Note: A unique constraint is defined as either a UNIQUE or PRIMARY KEY constraint.
10. All level 0 columns must be grouped within one CREATE TABLE of a cluster.

Note: If a PE data structure only contains MU fields, use an Adabas short name on the PE-subtable SEQNO(l).

Caution: This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Therefore, before executing this statement, it is strongly recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.

ANSI Specifics

This statement is not part of the ANSI standard.

The column default value of "ADABAS" is not part of the SQL standard.

Adabas SQL Server Specifics

The following examples define and create the table CONTRACT. The Adabas file already exists but there is no table definition in the catalog.

Example

```
Detailed format

CREATE TABLE DESCRIPTION contract DATABASE NUMBER 1 FILE NUMBER 21
     (contract-id             integer             SHORTNAME AA
                                   index ind_contract not null unique,
     price                 NUMERIC (13,3)         SHORTNAME AB not
        null,
     date_reservation         INTEGER             SHORTNAME AD,
     date_booking             INTEGER             SHORTNAME AG,
     date_cancellation        INTEGER             SHORTNAME AH,
     date_deposit             INTEGER             SHORTNAME AJ,
     amount_deposit        NUMERIC                SHORTNAME BA,
     date_payment             INTEGER             SHORTNAME BB,
     amount_payment        NUMERIC                SHORTNAME BE,
     id_customer              INTEGER             SHORTNAME CA not
        null,
     id_cruise             INTEGER                SHORTNAME CD not
        null);


Minimal format

CREATE TABLE DESCRIPTION contract
FILE NUMBER 21
        (contract_id          SHORTNAME AA,
     price                 SHORTNAME AB,
     id_cruise             SHORTNAME CD);
```

The effect of the minimal format on the catalog differs in some points from detailed format: First, there will be no entry for the index IND CONTRACT in the catalog, but this has no effect on the DML processing for which only the Adabas file structure is relevant. Second, the columns with NUMERIC data type will have the scale 0, since no scale information is held by the Adabas file structure.

The next example shows how the elements of an MU may be rotated into base columns. The examples is for a table containing bonuses for each month of the current year.

```
CREATE TABLE DESCRIPTION sales-bonuses
DATABASE CNXDB1
FILE NUMBER 15
( id          SHORTNAME "AA",
surname       SHORTNAME "AB",
first name    SHORTNAME "AC",
jan bonus     SHORTNAME "AD" ( 1 ),
feb_bonus     SHORTNAME "AD" ( 2 ),
mar bonus     SHORTNAME "AD" ( 3 ),
apr_bonus     SHORTNAME "AD" ( 4 ),
may_bonus     SHORTNAME "AD" ( 5 ),
jun_bonus     SHORTNAME "AD" ( 6 ),
jul bonus     SHORTNAME "AD" ( 7 ),
aug_bonus     SHORTNAME "AD" ( 8 ),
```

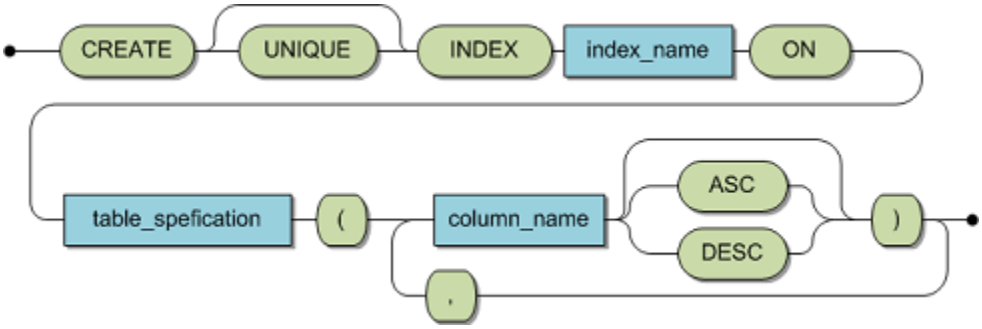
```
sept bonus   SHORTNAME "AD" ( 9 ),
oct bonus    SHORTNAME "AD" ( 10 ),
nov_bonus    SHORTNAME "AD" ( 11 ),
dec_bonus    SHORTNAME "AD" ( 12 ));
164
```

CREATE INDEX

Function

This command will create an index over the specified columns, on the specified table.

Syntax



Note: CREATE INDEX is not supported by CONNX for VSAM/CICS.
For the Adabas SQL Gateway, the ASC and DESC keywords are not supported.

Example:

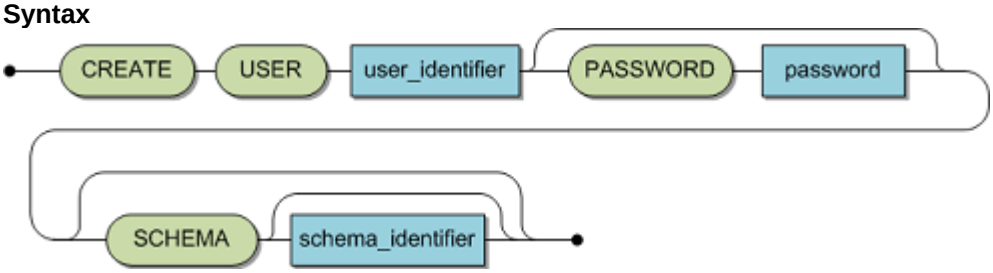
CREATE INDEX test_index on test(Companyname DESC, companyid ASC)

CREATE USER

Function

The CREATE USER statement establishes a user as a prerequisite to having access.

Syntax



user_identifier	A unique identifier for a user.
password	A valid password string with a maximum length of 20 bytes.
schema_identifier	A valid default schema identifier.

Description

CREATE USER defines a user with an optional password and optional character set. If a password has been specified, the user must enter it to gain access to the system. If no password has been specified, this user will have no password protection.

The SCHEMA option has an effect on the setting of the default schema after the user's CONNECT. The default is equal to the user identifier. If the schema does not already exist, it is created implicitly with the specified user as owner.

The CHARACTER SET option defines a default character set which is used when the user connects to the server without specifying a character set explicitly. The default is 'US-ASCII'.

All settings specified here can be changed later using the ALTER USER statement.

Note: *The password to be provided is visible in this statement. It will then be encrypted internally.*

Limitations

Only the DBA may execute this statement.

The user identifier must be unique.

Caution: *This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Therefore, before executing this statement, it is strongly recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.*

ANSI Specifics

The CREATE USER statement is not part of the Standard.

CONNX Specifics

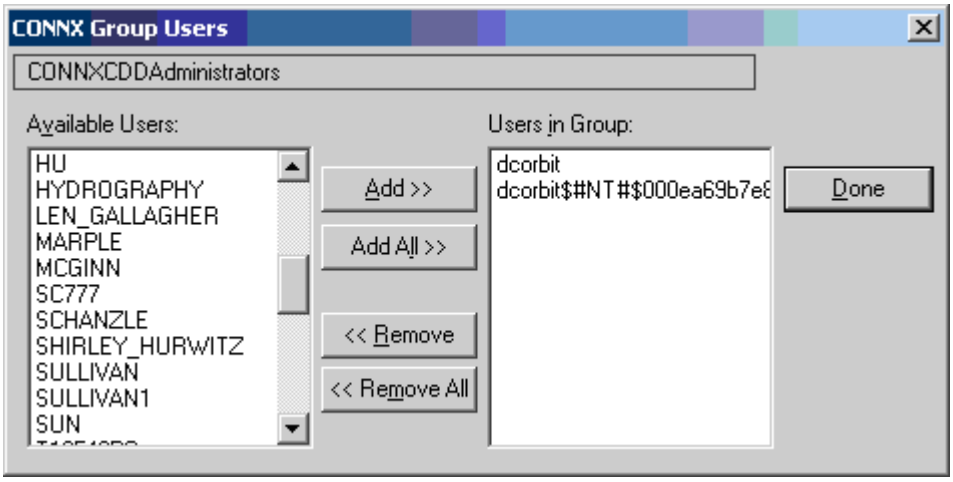
None

Example

The following example defines the user TIM with a password and a schema.

```
CREATE USER 'TIM' PASSWORD '&M%I?T' SCHEMA 'TIM';
```

In order to create users, you must be a database administrator. Database administrators are identified in the CONNX CDD as members of the group CONNXCDDAdministrators.

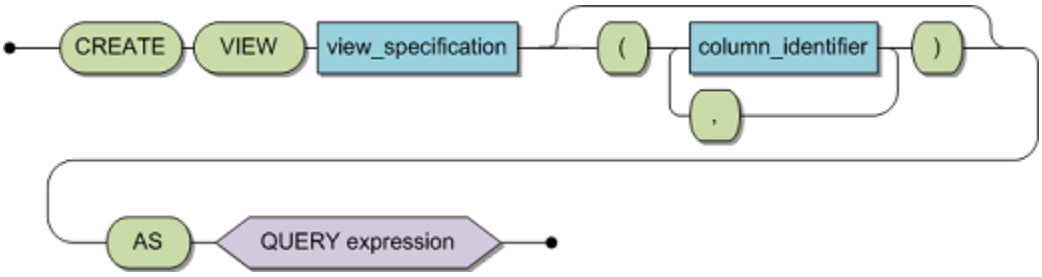


CREATE VIEW

Function

The CREATE VIEW statement is used to create a view derived from one or more base tables or other views.

Syntax



view_specification	The expected format is: schema identifier.table identifier. The default schema identifier is assumed if only a table identifier is specified. All views are placed in the CONNX catalog of CONNXDB.
column_identifier	Column identifier of a view
query_expression	Must be any valid query specification. See Query Expression.

Description

The CREATE VIEW statement is used to specify a viewed table, also called view. A view is a virtual table and therefore, has no physical representation. Values are conceptually derived from base tables as the need arises. If a schema identifier is given in the table specification, then the table identifier will thus be explicitly qualified, otherwise the current default schema identifier will be used.

The column identifier list specifies the number and order in which the columns will appear in the view. The number of column identifiers must equal the number of derived columns defined in the query specification. The nth column identifier represents the nth derived column and assumes its data type. Furthermore, two columns within the column identifier list may not be called the same.

If no column identifier list is specified, then the columns of the view are identified by the unqualified derived column labels of the query specification. If there is no label for a particular derived column, then the complete column identifier list must be specified.

- A view is called a "joined view" if more than one table has been specified or a joined view has been referenced in the FROM clause.
- A view is called a "grouped view" if the view is derived from a grouped query specification.
- A view is called a "read-only view" if the view is either grouped or joined or at least one of the derived columns does not have a label.

Only after successful execution of the statement is the view generally available. During execution the view description is stored in the catalog.

Limitations

The DBA can execute this statement for all users. All other users can use this statement only in a schema owned by the user. Creation of a view must include the following limitations:

- The table specification must be unique within an SQL environment, at runtime.

- The number of column identifiers specified in the desired column list must be identical to the number of derived columns given in the query specification .
- If no column identifier list is specified, all derived columns must have labels.
- The query specification may not reference host variables.
- The query specification may not reference the view which is the subject of the CREATE VIEW statement.

A view can not be updated when:

- it is a joined or grouped view, as described above
- a derived column is a literal (CREATE VIEW xyz AS SELECT col1, 'London' FROM table1)
- a derived column is an expression (CREATE VIEW xyz AS SELECT col1+3 FROM table1)

Caution: This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Therefore, before executing this statement, it is strongly recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.

ANSI Specifics

Within the ANSI concept the CREATE VIEW statement must be embedded in a CREATE SCHEMA statement. The SCHEMA as defined in ANSI is not fully supported by Adabas SQL Gateway Embedded SQL.

CONNX Specifics

None

Example

The following example creates a view named United States.

```
CREATE VIEW united_states
   AS SELECT * FROM persons
   WHERE country = 'USA' ;
```

Once the above view is created, it can be used to access information as if it were a normal table.

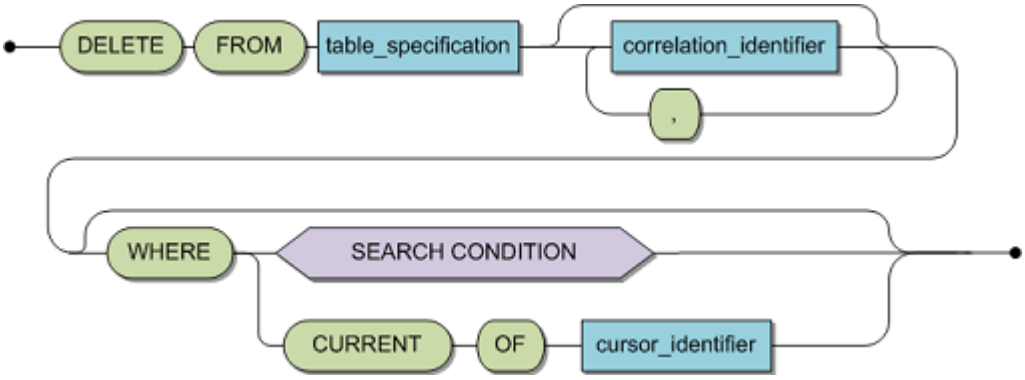
```
SELECT person_id
   FROM united_states
   WHERE united_states.city = 'PHILADELPHIA'
```

DELETE

Function

The DELETE statement removes a particular row or set of rows from the target table. There are two forms of the statement, positioned DELETE and searched DELETE.

Syntax



table_specification	The table to be amended. The table must be defined at compilation time. If the table specification is a view reference, then the view must be updatable. See Table Specification.
correlation_identifier	Allows the table to be referenced by another SQL identifier. See Correlation Identifiers.
search_condition	The specification of a resultant table which is to be deleted from the target base table. See Search Condition.
cursor identifier	A valid identifier of no more than 18 characters and which has not previously been used as a cursor identifier within the same compilation unit.

Description

A DELETE statement removes from the target table the row or rows identified in the WHERE clause.

Rows in Level 1 or level 2 tables can not be deleted directly using a DELETE statement. They can only be removed by deleting the associated level 0 row in the master table. The referencing level 1 and level 2 rows are automatically deleted with the level 0 row. This is analogous to a DELETE CASCADE in pure referential integrity terminology.

A DELETE statement with a WHERE CURRENT OF cursor identifier as its means of identifying the row to be deleted is called a positioned DELETE statement .

If the DELETE statement is positioned, then only the row to which the cursor is currently pointing is deleted. Hence, the cursor must be OPEN and pointing to a row otherwise a runtime error will occur. In addition, the cursor must be in itself updatable. See DECLARE CURSOR for further details. Once the row has been deleted, the cursor is not advanced, it simply no longer points to a row.

A DELETE statement with a WHERE search condition is called a searched DELETE statement. If the DELETE is searched, a resultant table is established at execution time in a similar manor to a query

specification. Each row in the target table which has a corresponding row in the resultant table is, then deleted.

A DELETE statement without any WHERE clause is really a special case of the searched DELETE alternative as a resultant table is established which contains all the rows of the target table. In such a case, all rows of the table are deleted.

Limitations

If the specified table is in fact a view, then that view must be updatable.

ANSI Specifics

A positioned DELETE statement must appear in the same compilation unit as the associated DECLARE and OPEN and must appear physically after the DECLARE.

The use of correlation identifiers in this context is not supported in ANSI-compatibility mode.

CONNX Specifics

A positioned DELETE statement can be in a different compilation unit to that of the associated DECLARE as long as a FOR UPDATE clause is specified. If the DELETE is in the same compilation unit as the associated DECLARE CURSOR statement, then there is no restriction as to the relative positions of the two statements.

The possibility to use a correlation identifier is CONNX extension.

If your ADABAS file uses superdescriptors, Superdescriptor Handling contains information to minimize search time.

Example

The following example deletes all cruises that depart from VIRGIN ISLANDS.

```
DELETE FROM cruise
   WHERE start_harbor = 'VIRGIN ISLANDS';
```

The following example deletes ALL information contained within table `cruise'.

```
DELETE FROM cruise;
```

The following example deletes the row in table cruise to which a cursor named 'cursor1' is currently pointing.

```
DELETE FROM cruise
  WHERE CURRENT OF cursor1;
```

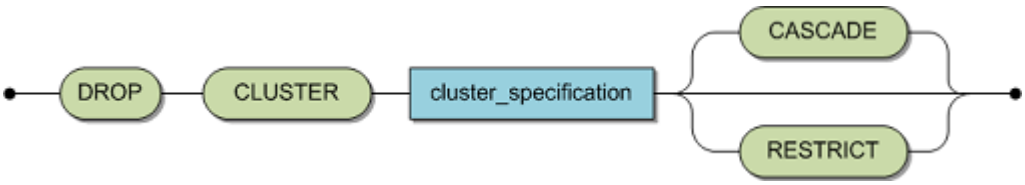
DROP CLUSTER (Adabas only)

Function

This statement deletes the logical and physical representation of a cluster.

Note: *Dropping a cluster causes all the data contained within to be destroyed. Once the statement has been executed, there is no way to recover the data.*

Syntax



cluster_specification	Schema identifier and table identifier of the master table in the cluster to be dropped. The default schema name is assumed if not specified here. See Cluster Specification.

Description

A cluster and all associated information will be deleted from Adabas SQL Gateway Embedded SQL's catalog and the underlying Adabas file will be deleted. Any other statements referencing this table will no longer be valid. In addition, any attempts to compile statements which reference this table will fail. Even if the table is re-specified, all previously compiled statements remain invalid.

- If the CASCADE option is specified, all view descriptions based on the cluster to be dropped will be deleted as well.
- If the RESTRICT option is specified, the statement execution will be rejected if there are dependent views.
- If neither of these two options is specified, RESTRICT is assumed.

Limitations

The DBA can execute this statement for all users. All other users can use this statement only in a schema owned by the user.

The specified table specification must denote an existing cluster at runtime.

Caution: *This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Therefore, before executing this statement, it is strongly recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.*

ANSI Specifics

The DROP CLUSTER statement is not part of the ANSI standard.

Adabas SQL Gateway Embedded SQL Specifics

None.

Example

The following example deletes all data and data structures of the cluster city_guide and all related views.

```
DROP CLUSTER city_guide CASCADE;
```

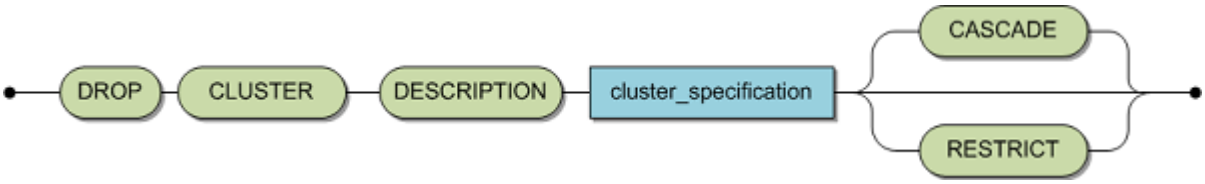
DROP CLUSTER DESCRIPTION (Adabas only)

Function

This statement deletes the logical representation of a cluster but not the underlying Adabas file.

Note: *Dropping a cluster description does not cause all data contained within to be deleted. Only the cluster description in the catalog is deleted.*

Syntax



cluster_specification	Schema identifier and table identifier of the master table in the cluster description to be dropped. The default schema name is assumed if not specified here. See Cluster Specification.

Description

A cluster definition and all its associated information will be deleted from Adabas SQL Gateway Embedded SQL's catalog. Any other statements referencing this cluster will no longer be valid. In addition, any attempts to compile statements which reference this cluster will fail. Even if the cluster description is re-specified, all previously compiled statements remain invalid.

- If the CASCADE option is specified, all dependent view descriptions based on the cluster to be dropped, will be deleted as well.
- If the RESTRICT option is specified, the statement execution will be rejected if there are dependent views.
- If neither of these two option is specified, RESTRICT is assumed.
- If the statement is invoked statically, then during pre-compilation, the schema need not exist in the catalog. For successful execution, however, the schema must exist in the catalog, regardless of how it is invoked.

Limitations

The DBA can execute this statement for all users. All other users can use this statement only in a schema owned by the user.

The specified table specification must denote an existing cluster at runtime.

Caution: *This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Therefore, before executing this statement, it is strongly recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.*

ANSI Specifics

The DROP TABLE DESCRIPTION statement is not part of the Standard.

Adabas SQL Gateway Embedded SQL Specifics

None.

Example

The following example deletes all cluster descriptions of the cluster city_guide and all related views.

```
DROP CLUSTER DESCRIPTION city_guide CASCADE;
```

DROP DATABASE

Function

DROP DATABASE removes a database entry in the CONNX Data Dictionary

Syntax

database_identifier	The name of the database in the CONNX Data dictionary to be removed.

Description

DROP DATABASE drops the link between the CONNX Data Dictionary and the Adabas database. The link to all objects (tables, stored procedures and views) underneath the database are also removed. The physical objects are not dropped. The physical database is not removed.

DROP INDEX

Function

This statement removes an index from a table.

Syntax

index_specification	the index_spericication for CONNX has the following special format: [CONNX table-name]_INDEX_[CONNX index-number]

Description

The specified index is removed from the specified base table.

If the statement is invoked statically, then during pre-compilation, the schema need not exist in the catalog. For successful execution, however, the schema must exist in the catalog, regardless of how it is invoked.

ANSI Specifics

This statement is not part of the Standard.

CONNX Specifics

None.

Example

DROP INDEX customers_INDEX_1

Note: *DROP INDEX is not supported by VSAM, Codasyl,DBMS or IMS.*

DROP SCHEMA

Function

DROP SCHEMA removes the schema from the catalog.

Syntax

schema_specification	A valid schema identifier representing the schema to be dropped.

Description

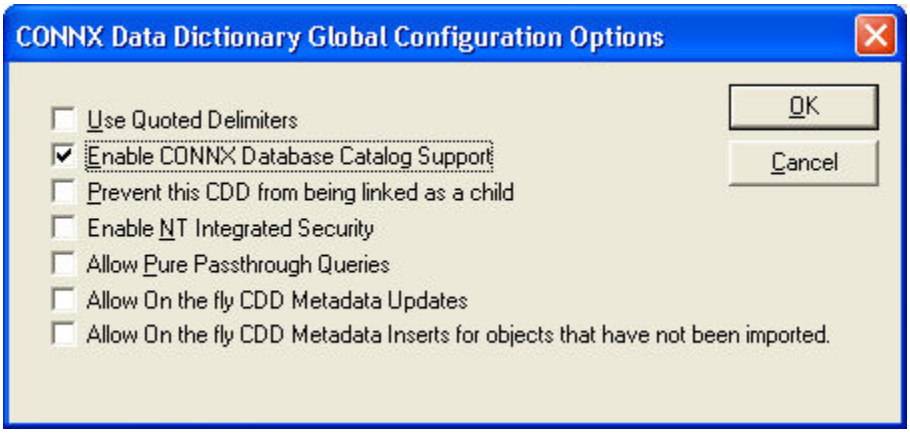
DROP SCHEMA deletes an SQL schema entry from the CONNX catalog.

If DROP SCHEMA is statically invoked, the schema need not exist in the catalog during pre-compilation. The schema must exist at the time it is executed.

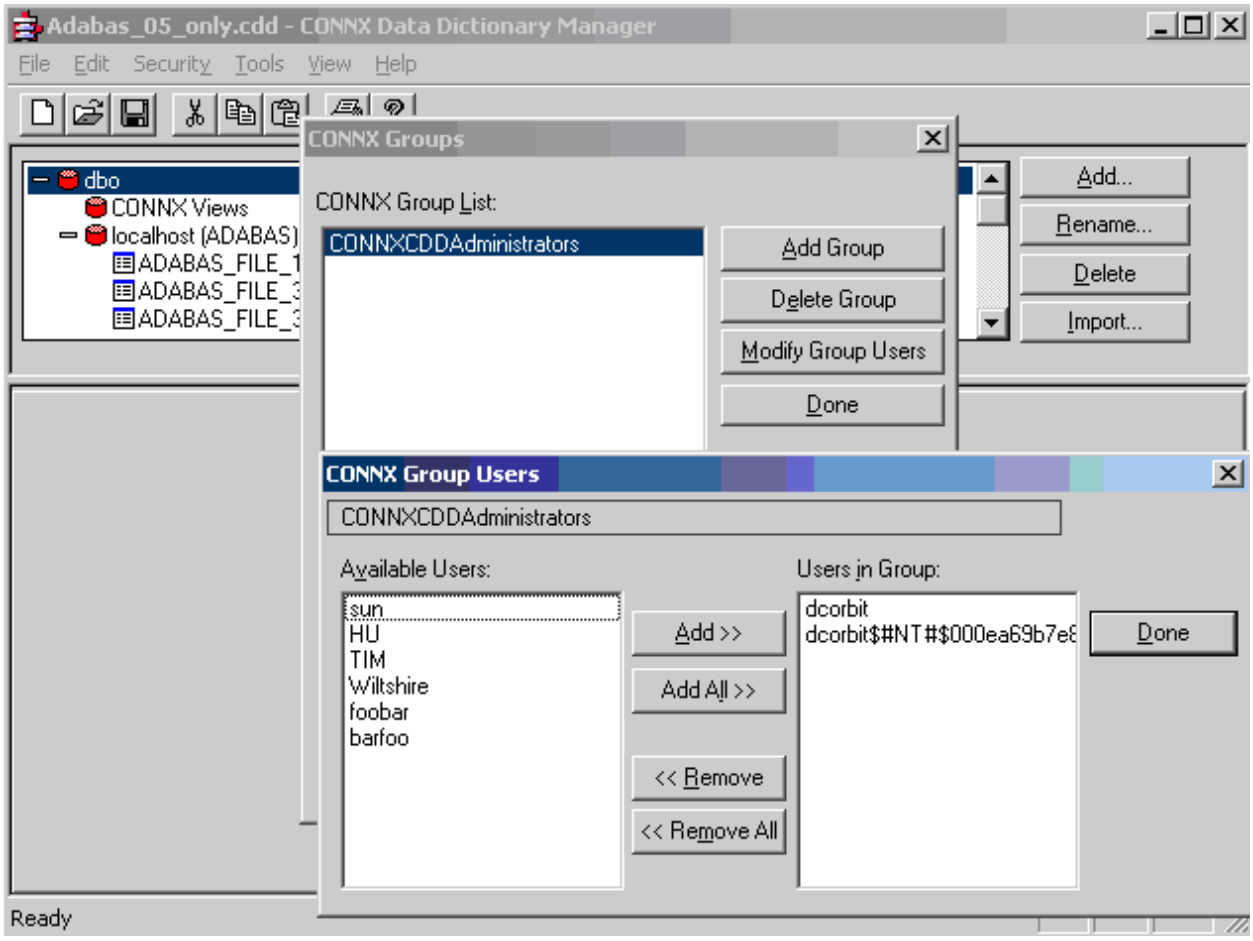
Limitations

It is not possible to drop the following schemas: DBO, PUBLIC, and INFORMATION_SCHEMA.

The CONNX CDD must have full catalog support enabled:



Only the designated DBA and not even the owner is permitted to drop a schema. The DBA must be a member of the CDD group called CONNXCDDAdministrators:



Caution: *This statement is not subject to transaction logic. An implicit COMMIT is performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK is performed. Therefore, before executing this statement, it is strongly recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.*

CONNX Specifics

The inclusion of the keyword RESTRICT is optional. RESTRICT is assumed if no keyword is specified.

Example

The following example drops the schema "Wiltshire".

```
DROP SCHEMA Wiltshire ;
```

DROP TABLE

Function

The statement removes a base table, all dependent views and all data.

Note: *Dropping a table causes all the data contained within to be destroyed. Once the statement has been executed, there is no way to recover the data.*

Syntax

table_specification	The expected format is catalog . schema . table identifier. The default schema and catalog is assumed if omitted. See Table Specification.

Description

A table and all associated information will be deleted from CONNXs catalog and the underlying Adabas file will be deleted. Any other statements referencing this table will no longer be valid. In addition, any attempts to compile statements which reference this table will fail. Even if the table is re-specified, all previously compiled statements remain invalid.

- If the CASCADE option is specified, all view descriptions based on the table to be dropped, will be deleted. Statement execution will be rejected if attempts are made to drop a table with dependent views but without the CASCADE option.
- If the statement is invoked statically, then during pre-compilation, the schema need not exist in the catalog. For successful execution, however, the schema must be existent in the catalog, regardless of how it is invoked.

Limitations

The DBA can execute this statement for all users. All other users can use this statement only in a schema owned by the user.

The specified table specification must denote an existing table at runtime.

If a table has been created as a part of a cluster, then it can not be dropped individually. The cluster must be dropped in order to remove this table.

Caution: *This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Therefore, before executing this statement, it is strongly recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.*

ANSI Specifics

Drop Table is part of the ANSI standard.

CONNX Specifics

None.

Example

The following example drops the table 'cruise' and all dependent views.

```
DROP TABLE cruise CASCADE
```

DROP TABLE DESCRIPTION

Function

The statement removes a base table description in the catalog but does not delete the data.

Note: *Dropping a table description does not cause all data contained within to be deleted. Only the table description in the catalog is deleted.*

Syntax

table_specification	The expected format is catalog . schema . table identifier. The default schema and catalog is assumed if omitted. See Table Specification.

Description

A table description and all its associated information will be deleted from Adabas SQL Gateway Embedded SQL's catalog. Any other statements referencing this table will no longer be valid. In addition, any attempts to compile statements which reference this table will fail. Even if the table is re-specified, all previously compiled statements remain invalid.

If the CASCADE option is specified, all dependent view descriptions based on the table to be dropped, will also be deleted. Statement execution will fail when attempting to drop a table description with dependent view descriptions but without the CASCADE option.

If the statement is invoked statically, then during pre-compilation, the table description need not exist in the catalog. For successful execution, however, the table description must exist in the catalog, regardless of how it is invoked.

Limitations

The DBA can execute this statement for all users. All other users can use this statement only in a schema owned by the user.

The specified table specification must denote an existing table at runtime.

If a table description has been created as a part of a cluster, then it can not be dropped individually. The cluster must be dropped in order to remove this table description.

Caution: *This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Therefore, before executing this statement, it is strongly recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.*

ANSI Specifics

The DROP TABLE DESCRIPTION statement is not part of the Standard.

Adabas SQL Gateway Embedded SQL Specifics

None.

Example

The following example drops the description of the table "cruise" and all dependent view descriptions.

```
DROP TABLE DESCRIPTION cruise CASCADE;
```

DROP USER

Function

DROP USER deletes a previously defined CONNX user from the data dictionary.

Syntax

user_identifier	An existing user identifier

Description

DROP USER removes an existing user identifier and the associated password.

Limitations

Before issuing this statement, the DBA will have to make sure that the user to be removed does not own any objects in the catalog. Otherwise, an error message will be issued. The statement may only be executed by the DBA.

Caution: *This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Therefore, before executing this statement, it is strongly recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.*

ANSI Specifics

The DROP USER statement is not part of the Standard.

CONNX Specifics

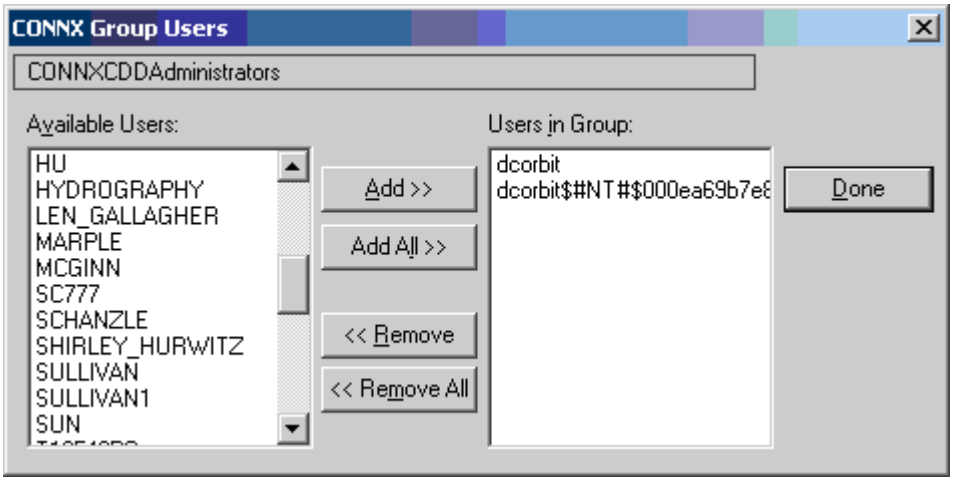
None

Example

The following example deletes the user 'Peter'.

```
DROP USER 'PETER'
```

In order to drop users, you must be a database administrator. Database administrators are identified in the CONNX CDD as members of the group CONNXCDDAdministrators.


CONNX Group Users
CONNXCDDAdministrators
Available Users:
HU
HYDROGRAPHY
LEN_GALLAGHER
MARPLE
MCGINN
SC777
SCHANZLE
SHIRLEY_HURWITZ
SULLIVAN
SULLIVAN1
SUN
Add >>
Add All >>
<< Remove
<< Remove All
Users in Group:
dcorbit
dcorbit$#NT#$000ea69b7e8
Done

DROP VIEW

Function

The statement deletes a view.

Note: *Dropping a view does not delete any underlying data as a view is a logical table and not a physical or base table.*

Syntax

view_specification	The expected format is: schema . table identifier . The default schema identifier is assumed if only a table identifier is specified. CONNX Views are always created in a CONNX special catalog called CONNXDB.

Description

A view and its description in CONNX is deleted. Any other statements referencing this view will no longer be valid. In addition, any attempts to compile statements which reference this view will fail. Even if the view is re-specified, all previously compiled statements remain invalid.

- If the CASCADE option is specified, all views based on the view to be dropped, will be deleted. Statement execution will fail when attempting to drop a view description with dependent views without the CASCADE option.
- If the statement is invoked statically, then during pre-compilation, the view need not exist in the catalog. For successful execution, however, the view must exist in the catalog, regardless of how it is invoked.

Limitations

The DBA can execute this statement for all users. All other users can use this statement only in a schema owned by the user.

The specified table specification must denote an existing view, at runtime.

Caution: *This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Therefore, before executing this statement, it is strongly recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.*

ANSI Specifics

None..

CONNX Specifics

None.

Example

The following example drops the view 'Canada' with all its related views.

```
DROP VIEW Canada CASCADE
```

GRANT

Function

GRANT gives users privileges to access tables or views.

Syntax

table_specification	The table or view for which the grant is to be performed. The table or view name should only be specified once. See Table Specification.
privilege_specification	A list of one or more privileges that are to be granted. See Privilege Specification.
grantee_specification	A user, a list of users or PUBLIC for which the grant is to be executed. A user should only be specified once. See Grantee Specification.

Description

For the specified tables or views, GRANT gives the specified privileges to a user, a list of users, or to PUBLIC. Do not specify the user identifier, the table, or the view identifier, multiple times.

By default, owners of a table have all privileges for that table. If you are the table owner, do not grant yourself rights on that table.

A privilege given with the WITH GRANT OPTION permits this user to grant other users privileges on the specified tables or views. The WITH GRANT OPTION can be specified for ALL PRIVILEGES, and so enables the grantee to grant all privileges to another user; or it can be specified for a particular set of privileges (see examples below).

Unsuccessful execution of the GRANT ALL PRIVILEGES statement results in response code 0, even though the ANSI Standard prescribes a Warning.

LIMITATIONS

General rules:

- Each user can have a privilege granted once. If Peter received the privilege SELECT on CRUISE from Tim, no one else can grant Peter the same SELECT on CRUISE privilege.
- Privileges on views are not automatically granted, just because privileges have been granted for the underlying base tabless. For example, Peter has created an updatable view based on the base table CRUISE. He has SELECT and UPDATE privilege on this view. If Peter is granted INSERT on the base table CRUISE, this will not result in INSERT privilege on the view.
- Granting UPDATE privileges on a table always means an implicit UPDATE on all columns of the table on which the grantor also has the GRANT option. In addition, a table privilege means that, when a column is added, all grantees that have the table privilege also receive the column privilege for the new column.

Authority to grant privileges:

- The table or view owner can grant privileges.
- Anyone the table or view owner has granted privileges to (with the WITH GRANT OPTION) can grant those privileges to others.

Granting privileges on Views:

- The view creator must have at least the SELECT privilege on all the base tables.
- For a read-only view, the SELECT privilege is the only privilege the owner has and may grant.
- The grantee must have at least a SELECT privilege, as above.
- If the above is not true, then the owner of the view must have at least the SELECT privilege plus the "WITH GRANT OPTION" to be able to grant privileges to other users for all base tables.
- The execution of the GRANT statement is closed by an implicit COMMIT and is, therefore, not capable of ROLLBACK.

Caution: *This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Therefore, before executing this statement, it is strongly recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.*

ANSI Specifics

The optional keyword TABLE in ON TABLE table specification is not supported.

CONNX Specifics

The keyword TABLE in ON TABLE table specification is optional.

Examples

Tim decided to GRANT ALL privileges to Peter on his table CRUISE.

```
GRANT ALL ON CRUISE TO 'PETER';
GRANT ALL PRIVILEGES ON CRUISE TO 'PETER'; [ ANSI-specific grant ]
```

Tim decided to GRANT the privilege SELECT to Anne on his table CRUISE.

```
GRANT SELECT ON CRUISE TO 'ANNE';
```

Tim decided to GRANT the privileges SELECT, INSERT and DELETE to Martin with the "WITH GRANT OPTION" on his table CRUISE. Martin then decides to GRANT the select privilege to Chris.

Tim :

```
GRANT SELECT, INSERT, DELETE ON CRUISE TO 'MARTIN'
WITH GRANT OPTION;
```

Martin :

```
GRANT SELECT ON CRUISE TO 'CHRIS';
```

Peter decided to GRANT Roland the SELECT privilege on table CRUISE for which he has no "WITH GRANT OPTION". Roland himself has no privileges for the table CRUISE. This statement fails as Peter has no privileges to perform this operation.

```
GRANT SELECT ON CRUISE TO 'ROLAND';
```

Peter decided to GRANT Roland the SELECT privilege on his view CRUISE_YACHTS.

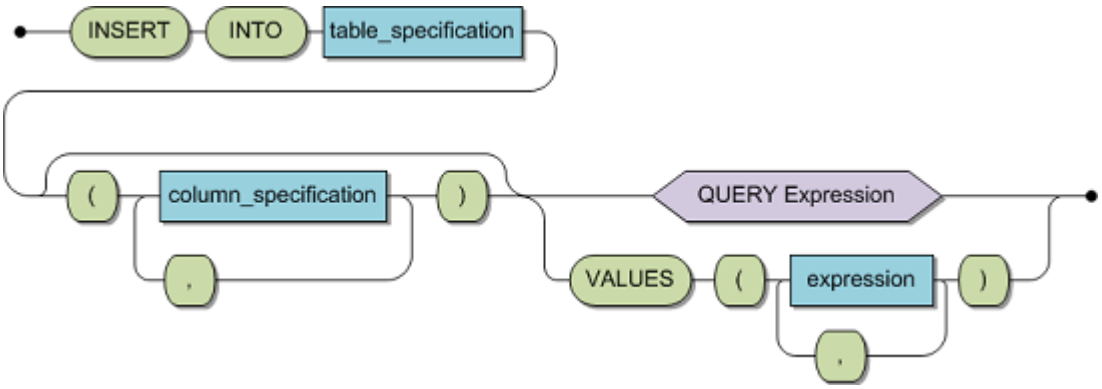
```
GRANT SELECT ON TABLE CRUISE_YACHTS TO 'ROLAND';
```

INSERT

Function

The INSERT statement inserts a new row into the target table using values derived from the row amendment expression .

Syntax



table_specification	A qualified or unqualified identifier which refers to the table to be amended. The table must be defined at this statement's compilation time. If the table specification is a view reference this view must be updatable. See Table Specification for more details.
expression	Defines the value which is to be assigned to the specified column.
query_expression	Defines a resultant table which will be used as a source of values for assignment to the specified columns. See Query Expression.

Description

An INSERT statement inserts a number of new rows into the target table as specified by the row amendment expression.

- If the target table in an INSERT statement is a subtable, then the values assigned to the foreign key columns must be equal to the values contained in the associated referenced key columns of the master table. This is compatible with the concepts of referential integrity. CONNX uses these key values to identify the record, and in case of a level 2 target table the periodic group within the record, into which the new candidate row is to be inserted. The insertion of a row should not result in an insertion of a new record but rather in the insertion of a new occurrence. If the specified foreign key values do not correspond to any referenced key values, then a referential constraint violation is issued.
- If the row amendment expression uses a query specification as its means of defining the input, then multiple rows may be inserted, otherwise the insertion of a single row will result.

- If the query specification results in no rows, then no rows are inserted and the field sqlcode in the SQLCA is set to +100.

Limitations

- If the target table is a view, this view must be updatable as described in the section DECLARE CURSOR.
- It is only possible to insert rows into subtables, if the corresponding referenced key columns in the master table exist and are specified with the same value.
- The special register SEQNO must not be specified as a target column. However, a value can be specified for a level 0 named SEQNO column. This value will then be the Adabas ISN. The values for level 1 and level 2 named SEQNO columns are occurrence numbers which are generated automatically and can not be specified in an insert operation. One exception is that the level 1 named SEQNO column may be assigned a value when the target table is a level 2 table and the value assigned to the level 1 named SEQNO column already exists.
- An empty string or zero value can not be inserted into columns which have been defined with SUPPRESSION (i.e. the Adabas NU option) and with the NOT NULL option, as these two values actually represent the NULL value.
- An empty string or zero value can not be inserted into a column that maps to an Adabas multiple-value field defined with SUPPRESSION, as these values are not representable under these conditions.

ANSI Specifics

None.

CONNX Specifics

None.

Example

The following example inserts a new row into the table cruise.

```
INSERT INTO cruise (cruise_id ,   start_date,  start_time,  end_date,  end_time,  start_harbor ,
   destination_harbor,  cruise_price,  bunk_number, bunks_free,  id_yacht,
   id_skipper,  id_predecessor ,   id_successor )
values ( 5037, 1234,19920925,12,19921206,14,'ACAPULCO',
  'LIVERPOOL', 2050, 7, 10 146, 244, 5037, 5039)
```

SELECT

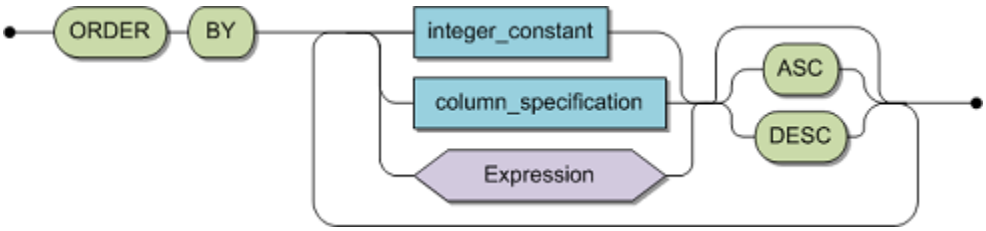
Function

The SELECT statement obtains data from the database according to the specified conditions.

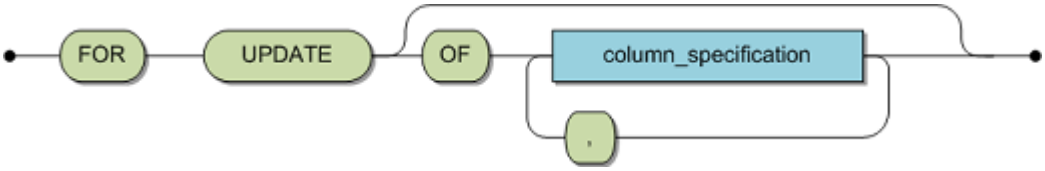
Note: If your ADABAS file uses super or sub descriptors, Sub/Super Descriptor Handling contains information to minimize search time.

Syntax

ORDER BY CLAUSE:



UPDATABILITY CLAUSE:



query expression	See Query Expression.
order_by_clause	The specification of a user-defined ordering of the resultant table. Otherwise the resultant table is not ordered.
updatability_clause	The explicit indication that this cursor is to be used in conjunction with either an UPDATE and/or DELETE WHERE CURRENT OF CURSOR statement.

Description

The SELECT statement is used to obtain data from the database. Please refer to the description of a query expression or query specification (Common Elements) for information on the processing of a SELECT statement.

- When submitted either dynamically the statement must be associated with a PREPARE statement and an associated dynamic cursor. The statement may then select more than one row.
- When used interactively, the statement may again select more than one row. The use of the INTO clause is not permitted.

Limitations

- The use of an ORDER BY clause is only valid within a dynamic or interactive SELECT statement. Its use enables the resultant table to be sorted in a user-defined sequence.
- The use of the FOR UPDATE clause is only valid within a dynamic or interactive SELECT statement.

ANSI Specifics

- The use of a FOR UPDATE clause is not supported.

CONNX Specifics

None.

Examples:

Simple SELECT Example:

```
SELECT customername, customerid FROM customers WHERE customerid = "ALCAO"
```

Inner Join Example:

```
SELECT customername, orderid FROM customers c, orders o WHERE c.customerid =
o.customerid
```

Outer Join Example:

```
SELECT customername, orderid FROM customers c, orders o WHERE c.customerid *=o.
customerid
```

Sub-query as a Table Example:

```
SELECT * FROM (SELECT customername, customerid FROM customers WHERE
customerid = "ALCAO") a
```

GROUP BY Example:

```
SELECT customername, sum(o.productquantity * p.productprice) FROM customers c,
orders o, products p WHERE c.customerid = o.customerid AND o.productid =
p.productid GROUP BY c.customername
```

GROUP BY Ordinal Example:

```
SELECT customername, sum(o.productquantity * p.productprice) FROM customers c,
orders o, products p WHERE c.customerid = o.customerid AND o.productid =
p.productid GROUP BY 1
```

ORDER BY Example:

```
SELECT customername, sum(o.productquantity * p.productprice) FROM customers c,
orders o, products p WHERE c.customerid = o.customerid AND o.productid =
p.productid ORDER BY c.customername
```

ORDER BY Ordinal Example:

```
SELECT customername, sum(o.productquantity * p.productprice) FROM customers c,
orders o, products p WHERE c.customerid = o.customerid AND o.productid =
p.productid ORDER BY 1
```

SELECT Sub-Query:

```
SELECT * from (SELECT * FROM CONNXDataSync.datasync.TableSynchronizations a,
CONNXDataSync.datasync.TableSynchronizations b where a.tableid = b.tableid)  a
```

REVOKE

Function

For the specified tables or views. REVOKE removes privileges from a user, a list of users or from PUBLIC.

Syntax

table_specification	A table or view for which the revocation is to be performed. Specify the table or view name only once. See Table Specification.
privilege_specification	A list of one or more privileges to be revoked. See Privilege Specification.
grantee_specification	A user, a list of users, or PUBLIC, for which the revocation is to be executed. Specify the user only once. See Grantee Specification.

Description

REVOKE revokes the specified privileges from a user, a list of users or from PUBLIC for the specified table or view. Do not specify the user identifier or the table or view identifier more than once.

Unsuccessful execution of the REVOKE ALL PRIVILEGES statement results in response code 0, even though the ANSI Standard prescribes a Warning.

For details about what privileges are possible, what they mean and the constraints on them, see Privilege Specification.

LIMITATIONS

General rules:

- If a revoke from PUBLIC is specified then only those privileges that have been granted to PUBLIC will be revoked.
- You can not revoke privileges from yourself.
- The keyword RESTRICT only affects the current user plus constraints.
- For the privilege UPDATE, a revocation of the table privilege causes an implicit revocation of all column privileges for the specified table. If only the column privilege is revoked, an existing table privilege remains unaltered.
- REVOKE CASCADE is not supported yet. If the revokee has granted the privilege to a third grantee, the privilege cannot be revoked from the revokee unless he has revoked it from the third grantee. Trying to revoke these privileges will fail and result in an error condition.

Authority to revoke privileges:

- The revoker is the owner of the table or view.
- The revoker gave the privileges that are to be revoked.

Revoking privileges from Views:

- Revoking privileges from a base table which would affect any view that relies upon that table will fail and result in an error conditions. To revoke these privileges, the view must be dropped first.
- The execution of the REVOKE statement is an atomic action that is closed by an implicit COMMIT and can, therefore, not be rolled back.

Caution: *This statement is not subject to transaction logic. An implicit COMMIT will be performed after successful execution of this statement. If an error is detected during execution of this statement, an implicit ROLLBACK will be performed. Therefore, before executing this statement, it is strongly*

recommended to complete any open transaction containing INSERT, UPDATE and/or DELETE statements by issuing an explicit COMMIT or ROLLBACK statement.

ANSI Specifics

You must specify either CASCADE or RESTRICT.

CONNX Specifics

- It is optional (and has no effect) to specify the keyword TABLE in ON TABLE table specification.
- If neither CASCADE nor RESTRICT is specified, then RESTRICT is the default action.
- The CASCADE functionality is not yet implemented.

Examples

Simple revocation:

Tim has given Peter ALL privileges on table CRUISE. Tim then decides to revoke the DELETE privilege from Peter.

```
REVOKE DELETE ON CRUISE FROM 'PETER';
REVOKE DELETE ON CRUISE FROM 'PETER' RESTRICT; [ANSI-specific method]
```

This has the effect of removing the DELETE privilege from Peter, but will still leave him with the SELECT, INSERT and UPDATE privileges for this table.

Simple revocation (no cascading):

Tim has given Peter ALL privileges including the "WITH GRANT OPTION" on table CRUISE. Peter then gives Anne the privileges to SELECT and DELETE on table CRUISE. Tim then decides to revoke the DELETE privilege from Peter.

Tim:

```
REVOKE DELETE ON CRUISE FROM 'PETER';
```

Tim:

```
REVOKE DELETE ON CRUISE FROM 'PETER' RESTRICT; [ANSI-specific method]
```

This will fail and result in the error message that there are still dependent privileges. First, Peter has to revoke the privileges SELECT and DELETE from Anne:

Peter:

```
REVOKE SELECT, DELETE ON CRUISE FROM 'ANNE';
```

Peter:

```
REVOKE SELECT, DELETE ON CRUISE FROM 'ANNE' RESTRICT;[ANSI-specific
   method]
```

After that, Tim can revoke the DELETE privilege from Peter.

Assume that Tim has also given Peter the UPDATE table privilege. Now he wants to revoke the UPDATE privilege on column xx from Peter.

Tim:

```
REVOKE UPDATE ( XX ) ON CRUISE FROM 'PETER';
```

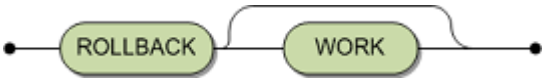
The result will be that the UPDATE table privilege still exists: only the column privilege for column xx is destroyed. If Peter then tries to grant the UPDATE privilege to Gary, this will have the effect that Gary also gets UPDATE column privileges for all columns of table CRUISE with the exception of column xx. That means Gary is allowed to update all columns in CRUISE except xx.

ROLLBACK

Function

The ROLLBACK statement terminates a transaction and removes all changes to the database that were made during the current transaction.

Syntax



Description

The ROLLBACK statement terminates the current transaction and starts a new transaction. All changes to the database that have been made during the transaction are not applied and the database is as it existed at the time the transaction was started. All cursors that have been opened during the current transaction are closed.

Limitations

None.

ANSI Specifics

The keyword WORK is mandatory.

CONNX Specifics

The keyword WORK is optional.

Example

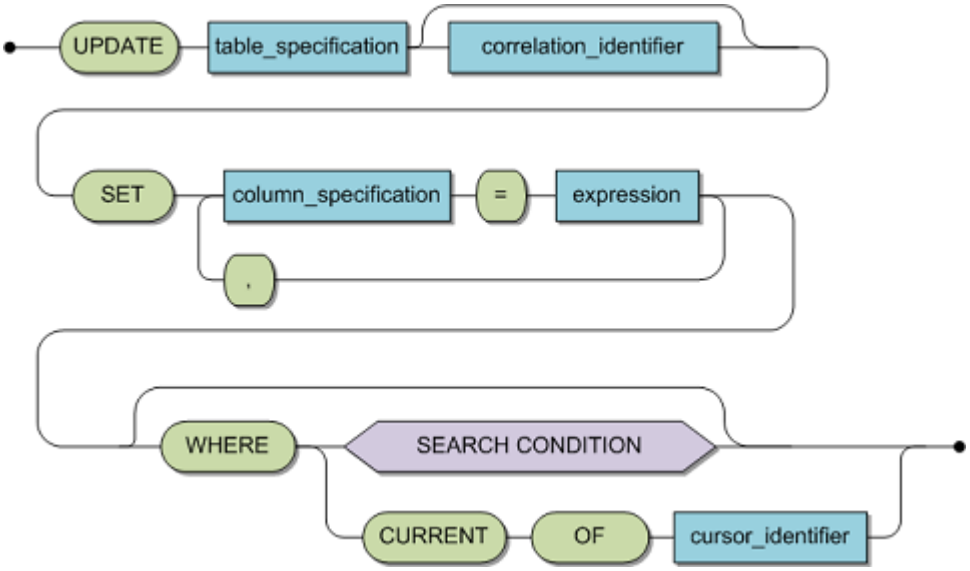
```
ROLLBACK WORK ;
```

UPDATE

Function

The UPDATE statement modifies the data contained in a particular row or set of rows. There are two forms, positioned UPDATE and searched UPDATE.

Syntax



table_specification	A qualified or unqualified identifier which refers to the table to be amended. The table must be defined at this statement's compilation time. If the table specification is a view reference this view must be updatable. See Common Elements, section Table Specification for more details.
correlation_identifier	Allows the table to be referenced by another identifier. See Common Elements , section Correlation Identifier for more details.
expression	Specifies the new values to which the columns in the row(s) under consideration will be assigned. See Common Elements, section Row Amendment Expression for more details.
WHERE CURRENT OF cursor identifier	Indicates that the UPDATE is positioned. The cursor identifier refers to a cursor which is currently open and pointing to a row.
WHERE_search_condition	Indicates that the UPDATE statement is searched. Omission of the WHERE clause equates to a special case of a searched UPDATE statement. See Search Condition.

Description

- An UPDATE statement modifies the columns of the rows identified in the WHERE clause with the values specified in the row amendment expression.

- Updates of key column values in the master table will be cascaded to the related subtables. All other columns of a subtable can be updated with new values as usual, provided that the values of foreign keys and SEQNOs remain the same as already stored.
- If the UPDATE statement is positioned, then the UPDATE is only applied to the row to which the cursor is currently pointing. The cursor must be open and pointing to a row otherwise a runtime error will occur. In addition, the cursor must be updatable. See DECLARE CURSOR for further details. Updating does not alter the position of the cursor. In addition, any locks on the row are not released until either a COMMIT or a ROLLBACK statement is executed.
- Alternatively, in case of a searched UPDATE statement, a resultant table is established at execution time in a similar manor to a query specification. The UPDATE, then occurs for each row in the resultant table as specified by the row amendment expression . All the rows of the resultant table are locked and are not released until either a COMMIT or a ROLLBACK is executed. If no rows are identified for updating, then the field SQLCODE of the SQLCA will be set to +100.
- An UPDATE statement without a WHERE clause is really a special case of the searched alternative as a resultant table is established which contains all the rows of the target table.

Limitations

- If the table referenced is a view, then this view must be updatable.
- (Adabas only) For reasons of enforcing referential integrity it is not possible to change the value of foreign key columns in level 1 or level 2 tables. In a clustered environment this would require to physically move a row to a new location.
- Restrictions which apply when updating views can be found in the Limitation section of the CREATE VIEW statement description.
- (Adabas only) A SEQNO column is not updatable. The SEQNO columns map to the information that is used for internal Adabas addressing, and no rows will be moved to a new location using an UPDATE statement.
- (Adabas only) An empty string or zero value can not be inserted into columns which have been defined with SUPPRESSION (i.e. the Adabas NU option) and with the NULL capability, as these two values are actually not representable under this condition. Same applies to columns with just the NULL capability, as the empty string or zero value represent the NULL value.

ANSI Specifics

- The use of the VALUES format in the row amendment expression is not permitted.
- A positioned UPDATE statement must appear in the same compilation unit as the associated DECLARE and OPEN statements and must appear physically after the DECLARE statement.
- The use of correlation identifiers in this context is not supported in ANSI compatibility mode.

CONNX Specifics

- The use of correlation identifiers is permitted.

Example

The following example updates all prices in the cruise table by adding 100 to the original cost.

```
UPDATE cruise
   SET cruise_price = cruise_price + 100 ;
```

The following example decreases customer 816 amount to pay by 100.

```
UPDATE contract
   SET amount_payment = amount_payment - 100
      WHERE id_customer = 816;
```

SQL Aggregate Functions

AVG

Returns the average of a column in the resultset/group.

Example:

With the sample data set:

Field1	Field2	Field3
23	10	17
17	10	9
5	12	15

```
SELECT AVG(Field1) returns 15.
```

COUNT

Returns the number of records in the resultset/group. When used with the optional DISTINCT keyword, returns the unique number of records in the resultset/group.

This parameter can be either an asterisk (*) or an expression. If an asterisk is used then all rows of the resultset/group are counted regardless of NULL status. If the expression for a row evaluated to NULL, it will not be counted.

Example:

With the sample data set:

Field1	Field2	Field3
23	10	17
17	10	9
5	12	15

```
SELECT COUNT(Field2) returns 3.

SELECT COUNT(Distinct Field2) returns 2.
```

MAX

Returns the largest value of the field in the resultset/group.

Example:

With the sample data set:

Field1	Field2	Field3
23	10	17
17	10	9
5	12	15

SELECT MAX(Field1) returns 23.

MIN

Returns the smallest value of the field in the resultset/group.

Example:

With the sample data set:

Field1	Field2	Field3
23	10	17
17	10	9
5	12	15

SELECT MIN(Field1) returns 5.

SUM

Returns the sum total of the field in the resultset/group.

Example:

With the sample data set:

Field1	Field2	Field3
23	10	17
17	10	9
5	12	15

SELECT SUM(Field1) returns 45.

ASCII(string_exp)

Returns the ASCII code value of the leftmost character of string_exp as an integer.

Example:

```
SELECT ASCII('CONNX') returns 67 (ASCII code for 'C')
```

BIT_LENGTH(string_exp)

Returns the length in bits of the string expression.

Example:

```
SELECT BIT_LENGTH('ABCDEFG') returns 56 (length in bits of a 7-
   character string expression)
```

CHAR_LENGTH(string_exp)

Returns the length in characters of the string expression if the string expression is of a Character data type; otherwise, returns the length in bytes of the string expression.

Example:

```
SELECT CHAR_LENGTH('ABCDEFG') returns 7 (number of characters in string
   expression)
```

CHARACTER_LENGTH(string_exp)

Returns the length in characters of the string expression if the string expression is of a Character data type; otherwise, returns the length in bytes of the string expression.

Example:

```
SELECT CHARACTER_LENGTH('ABCDEFG') returns 7 (length of string
   expression)
```

CHR(code)

Returns the character that has the ASCII code value specified by code. The value of code should be between 0 and 255.

Example:

```
SELECT CHR(100) returns d (ASCII code for '100')
```

CONCAT(string_exp1, string_exp2)

Returns a character string that is the result of concatenating string_exp2 to string_exp1.
Note - the || operator may also be used to concatenate two values.

Example:

```
SELECT CONCAT('ABCDEFG','HIJKLMN') returns ABCDEFGHIJKLMN
SELECT 'ABCDEFG' || 'HIJKLMN' returns ABCDEFGHIJKLMN
```

DIFFERENCE(string_exp1, string_exp2)

Returns an integer value that indicates the difference between the values returned by the SOUNDEX function for string_exp1 and string_exp2.

Example:

```
SELECT DIFFERENCE('ABC','DEF') returns -2026 (the difference between
   the values returned by the SOUNDEX function)
```

HEX(numeric_expr)

Returns the hexadecimal representation of the numeric value in character form.

Example:

```
SELECT HEX(15) returns F (hexadecimal value of 15)
```

INSERT(string_exp1, start, length, string_exp2)

Returns a character string where length characters have been deleted from string_exp1 beginning at start and where string_exp2 has been inserted into string_exp, beginning at start.

Example:

```
SELECT INSERT('ABCDEFGHIJ',3,4,'MMMMM') returns ABMMMMMGHIJ
```

LCASE(string_exp)

Returns a string equal to that in string_exp with all uppercase characters converted to lowercase.

Example:

```
SELECT LCASE('ABCDEFG') returns abcdefg
```

LEFT(string_exp, count_exp)

Returns the leftmost count characters of string_exp.

Example:

```
SELECT LEFT('ABCDEFG',4) returns ABCD
```

LENGTH(string_exp)

Returns the number of characters in string_exp, excluding trailing blanks.

Example:

```
SELECT LENGTH('ABCEDFG') returns 7
```

LOCATE(string_exp1, string_exp2[, start])

Returns the starting position (one based) of the first occurrence of string_exp2 within string_exp1. The search for the first occurrence of string_exp2 begins with the first character position in string_exp1 starting at the specified position. If the start position is omitted, then the search is performed from the beginning of the string.

Example:

```
SELECT LOCATE('ABCDEFG','DEFG') returns 4
```

LTRIM(string_exp)

Returns the characters of string_exp, with leading blanks removed.

Example:

```
SELECT LTRIM(' ABCDEFG') returns ABCDEFG
```

LTRIMWHITESPACE(string_exp)

Returns the characters of string_exp, with leading white space characters (space, tab, new line, carriage return) removed.

Example:

```
SELECT LTRIMWHITESPACE(' ABCDEFG') returns ABCDEFG
```

OCTET_LENGTH(string_exp)

Returns the length in bytes of the string expression. The result is the smallest integer not less than the number of bits divided by 8.

Example:

```
SELECT OCTET_LENGTH('ABCDEFG') returns 7
```

POSITION(string_exp IN string_exp)

Returns the starting position (one based) of the first character expression in the second character expression.

Example:

```
SELECT POSITION('CDE' IN 'ABCDEFG') returns 3
```

REMOVENULL(string_exp)

Replaces all embedded binary zeros in a string with spaces.

Example:

```
removenull('abcde'+chr(0)+'moretext') returns 'abcde moretext' - where
  the binary zero has been replaced with a space
```

REPEAT(string_exp, count)

Returns a character string composed of string_exp repeated count times.

Example:

```
SELECT REPEAT('ABC',3) returns ABCABCABC
```

REPLACE(string_exp1, string_exp2, string_exp3)

Search string_exp1 for occurrences of string_exp2 and replace with string_exp3.

Example:

```
SELECT REPLACE('ABCDE','CDE','FG') returns ABFG
```

RIGHT(string_exp, count)

Returns the rightmost count characters of string_exp.

Example:

```
SELECT RIGHT('ABCDE',2) returns DE
```

RTRIM(string_exp)

Returns the characters of string_exp with trailing blanks removed.

Example:

```
SELECT RTRIM('ABCDE ') returns ABCDE
```

RTRIMWHITESPACE(string_exp)

Returns the characters of string_exp with trailing white space characters (space, tab, new line, carriage return) removed.

Example:

```
SELECT RTRIMWHITESPACE('ABCDE ') returns ABCDE
```

SUBSTRING(string_exp, start, length)

Returns a character string that is derived from string_exp beginning at the character position specified by start for length characters.

Example:

```
SELECT Substring('ABCDE', 2, 1)
```

Results: 'B'

Example:

```
SELECT Substring('ABCDE', 4, 2)
```

Results: 'DE'

SOUNDEX(string_exp)

Returns a data source-dependent character string representing the sound of the words in string_exp. A SOUNDEX code is returned.

Example:

```
SELECT SOUNDEX('SMITH') returns S53 (the SOUNDEX code for words that
   sound like "Smith")
```

SPACE(count)

Returns a character string consisting of count spaces.

Example:

```
SELECT SPACE(12) returns a character string of 12 count spaces
```

TRIM(string_exp)

Returns the characters of string_exp with both leading and trailing blanks removed.

Example:

```
SELECT TRIM(' ABCDE ') returns ABCDE
```

TRIMWHITESPACE(string_exp)

Returns the characters of string_exp with both leading and trailing white space characters (space, tab, new line, carriage return) removed.

Example:

```
SELECT TRIMWHITESPACE(' ABCDE ') returns ABCDE
```

UCASE(string_exp)

Returns a string equal to that in string_exp with all lowercase characters converted to uppercase.

Example:

```
SELECT UCASE('abcde') returns ABCDE
```

SQL Date Functions

CURRENT_DATE()

Returns the current date.

Example:

```
SELECT CURRENT_DATE() returns 9/13/01 (for example)
```

CURRENT_TIME()

Returns the current local time.

Example:

```
SELECT CURRENT_TIME() returns 9/13/01 10:35:45 AM
```

CURRENT_TIMESTAMP()

Returns the current local date and local time as a timestamp value. CONNX returns a value that is accurate to the millisecond.

Example:

```
SELECT CURRENT_TIMESTAMP() returns 9/13/01 10:35:45 AM
```

CURDATE()

Returns the current date.

Example:

```
SELECT CURDATE() returns 9/13/01
```

CURTIME()

Returns the current local time.

Example:

```
SELECT CURTIME() returns 9/13/01 12:08:08 AM
```

DAYNAME(date_exp)

Returns a character string containing the data source/specific name of the day (for example, Sunday through Saturday).

Example:

```
SELECT DAYNAME({d '2001-09-31'}) returns Thursday
```

DAYOFMONTH(date_exp)

Returns the day of the month based on the month field in date_exp as an integer value in the range of 1-31.

Example:

```
SELECT DAYOFMONTH({d '2001-09-13'}) returns 13
```

DAYOFWEEK(date_exp)

Returns the day of the week based on the week field in date_exp as an integer value in the range of 1-7, where 1 represents Sunday.

Example:

```
SELECT DAYOFWEEK({d '2001-09-13'}) returns 5
```

(Thursday as the fifth day of the week)

DAYOFYEAR(date_exp)

Returns the day of the year based on the year field in date_exp as an integer value in the range of 1-366.

Example:

```
SELECT DAYOFYEAR({d '2001-09-13'}) returns 256
```

EXTRACT(extract-field FROM extract-source)

Returns the extract-field portion of the extract-source. The extract-source argument is a date or timestamp expression.

The extract-field argument can be one of the following keywords:

YEAR
MONTH
DAY
HOUR
MINUTE
SECOND

Example:

```
SELECT EXTRACT(YEAR FROM {d '2001-09-13'}) returns 2001
```

HOUR(time_exp)

Returns the hour based on the hour field in time_exp as an integer value in the range of 0-23.

Example:

```
SELECT HOUR({t '10:00:00'}) returns 10 (ten o'clock is the tenth hour
   of the day)
```

MINUTE(time_exp)

Returns the minute based on the minute field in time_exp as an integer value in the range of 0-59.

Example:

```
SELECT MINUTE({t '10:33:00'}) returns 33
```

MONTH(date_exp)

Returns the month based on the month field in date_exp as an integer value in the range of 1-12.

Example:

```
SELECT MONTH({d '2000-09-13'}) returns 9
```

MONTHNAME(date_exp)

Returns a character string containing the data source/specific name of the month (for example, January through December). Currently only supports English locale.

Example:

```
SELECT MONTHNAME({d '2000-09-13'}) returns
   September
```

NOW()

Returns current date and time as a timestamp value. CONNX returns a value that is accurate to the millisecond.

Example:

```
SELECT NOW() returns 9/14/01 10:33:00 AM (for example)
```

QUARTER(date_exp)

Returns the quarter in date_exp as an integer value in the range of 1-4, where 1 represents January 1 through March 31.

Example:

```
SELECT QUARTER({d '2001-09-13'}) returns 3
```

SECOND(time_exp)

Returns the second based on the second field in time_exp as an integer value in the range of 0-59.

Example:

```
SELECT SECOND({t '10:00:03'}) returns 3
```

TIMESTAMPADD(interval, integer_exp, timestamp_exp)

Returns a timestamp offset by the specified interval.
Interval can be one of the following values
SQL_TSI_FRAC_SECOND (in billionths of a second)
SQL_TSI_SECOND
SQL_TSI_MINUTE
SQL_TSI_HOUR
SQL_TSI_DAY
SQL_TSI_WEEK
SQL_TSI_MONTH
SQL_TSI_QUARTER
SQL_TSI_YEAR

Example:

```
SELECT timestampadd(SQL_TSI_HOUR, 3, {ts '1998-01-11 10:00:00'})
```

Returns: 1998-01-11 13:00:00

Example:

```
SELECT timestampadd(SQL_TSI_DAY, -3, {ts '1998-01-11 10:00:00'})
```

Returns: 1998-01-08 10:00:00

TIMESTAMPDIFF(interval, timestamp_exp1, timestamp_exp2)

Returns the integer number of intervals of type interval by which timestamp_exp2 is greater than timestamp_exp1. Valid Intervals are described in the TIMESTAMPADD function.

Example:

```
SELECT timestampdiff(SQL_TSI_MONTH, {ts '1998-01-11 10:00:00'}, {ts
   '1998-03-11 10:00:00'})
```

Returns: 2

Example:

```
SELECT timestampdiff( SQL_TSI_YEAR, {ts '2003-01-11 10:00:00'}, {ts
   '1998-01-11 10:00:00'})
```

Returns: -5

WEEK functions

WEEK functions-summary

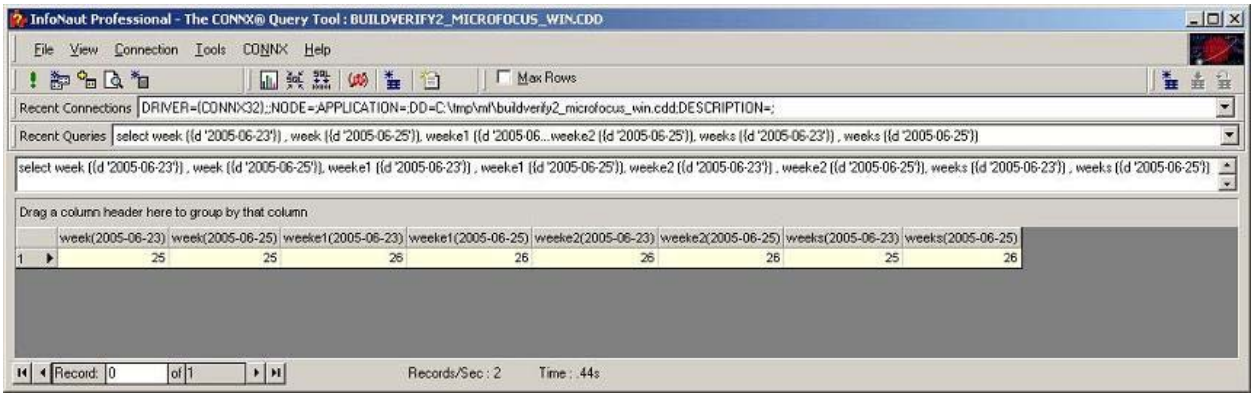
CONNX has four functions to determine which week a date occurs in:

- WEEK(timestamp) for ISO week
- WEEKE1(timestamp) for Excel WEEKNUM(CELL,1)
- WEEKE2(timestamp) for Excel WEEKNUM(CELL,2)
- WEEKS(timestamp) for simple week numbers.

Depending on how you want to calculate which week a date occurs in, one of these functions should satisfy your needs.

Microsoft has an excellent article explaining the differences between the four functions.

Here is an example of how the four functions calculate specific week numbers:



WEEK(date_exp)

Returns the ISO week number (ISO8601:2000 Standard). All weeks begin on a Monday. Week 1 starts on Monday of the first week of the calendar year with a Thursday.

The International Organization for Standardization issued *Standard 8601 -- Representation Of Dates And Times*, in 1988. This provides some standardization for "week numbers." Since compliance with these standards is entirely voluntary, your business may or may not use the ISO definitions.

Under the ISO standard, a week always begins on a Monday, and ends on a Sunday. The first week of a year is that week which contains the first Thursday of the year, or, equivalently, contains January 4th.

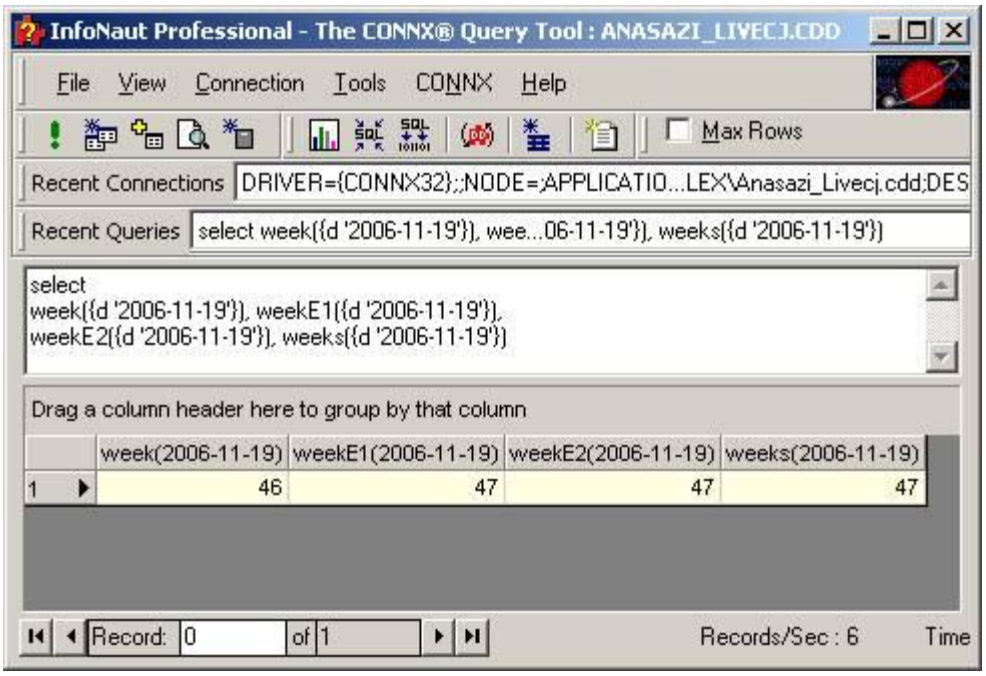
While this provides some standardization, it can lead to unexpected results - namely that the first few days of a year may not be in week 1 at all. Instead, they will be in week 52 of the preceding year! For example, the year 2000 began on Saturday. Under the ISO standard, weeks always begin on a Monday. In 2000, the first Thursday was January 6th, so week 1 begins the preceding Monday, or January 3rd. Therefore, the first two days of 2000, January 1st and January 2nd, fall into week 52 of 1999.

An ISO week number may be between 1 and 53. Under the ISO standard, week 1 will always have at least 4 days. If January 1st falls on a Friday, Saturday, or Sunday, the first few days of the year are defined as being in the last (52nd or 53rd) week of the previous year.

Example:

```
SELECT WEEK({d '2006-11-19'}) returns 46
```

The results of the SQL expression are shown in the first column.



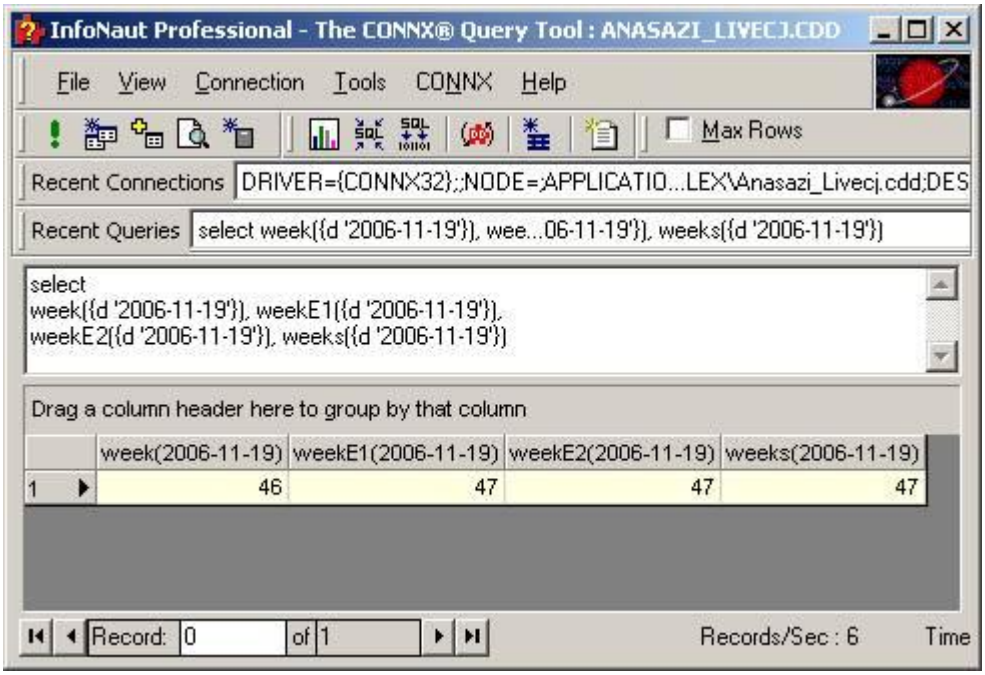
WEEKE1(date_exp)

Excel WEEKNUM function with an optional second argument of 1 (default). Week 1 begins on January 1st; week two begins on the following Sunday.

Example:

```
SELECT WEEKE1({d '2006-11-19'}) returns 47
```

The results of the SQL expression are shown in the second column.



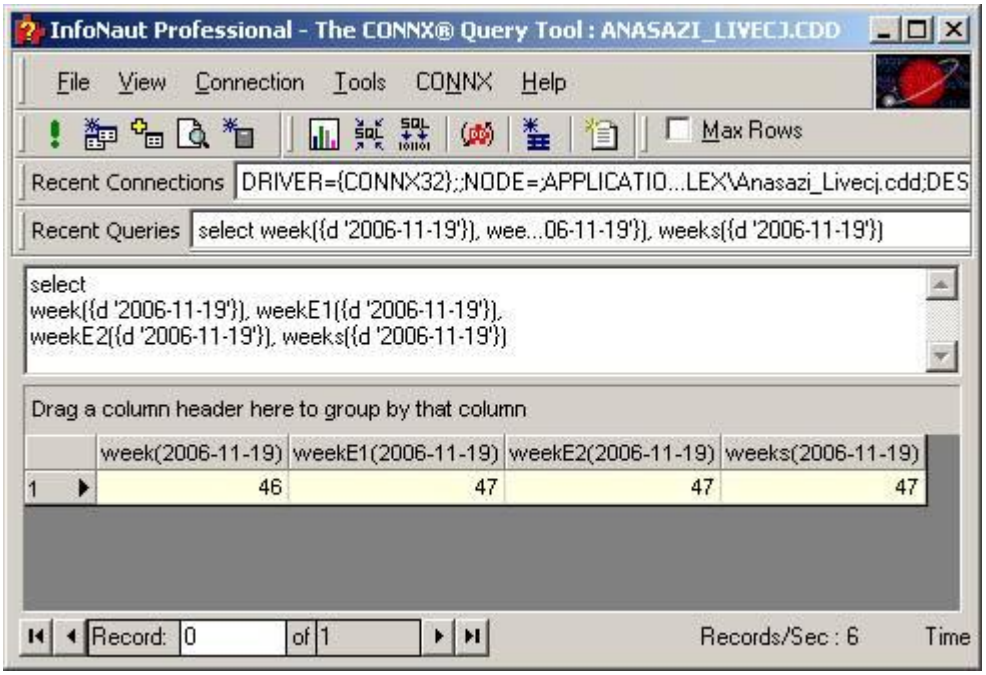
WEEKE2(date_exp)

Excel WEEKNUM function with an optional second argument of 2. Week 1 begins on January 1st; week two begins on the following Monday.

Example:

```
SELECT WEEKE2({d '2006-11-19'}) returns 47
```

The results of the SQL expression are shown in the third column.



WEEKS(date_exp)

Absolute Week Numbers. Returns the week of the year based on the week field in date_exp as an integer value in the range of 1-53.

An absolute week number is the seven day period that a date falls within, based solely on the first day of the year, regardless of the day of the week.

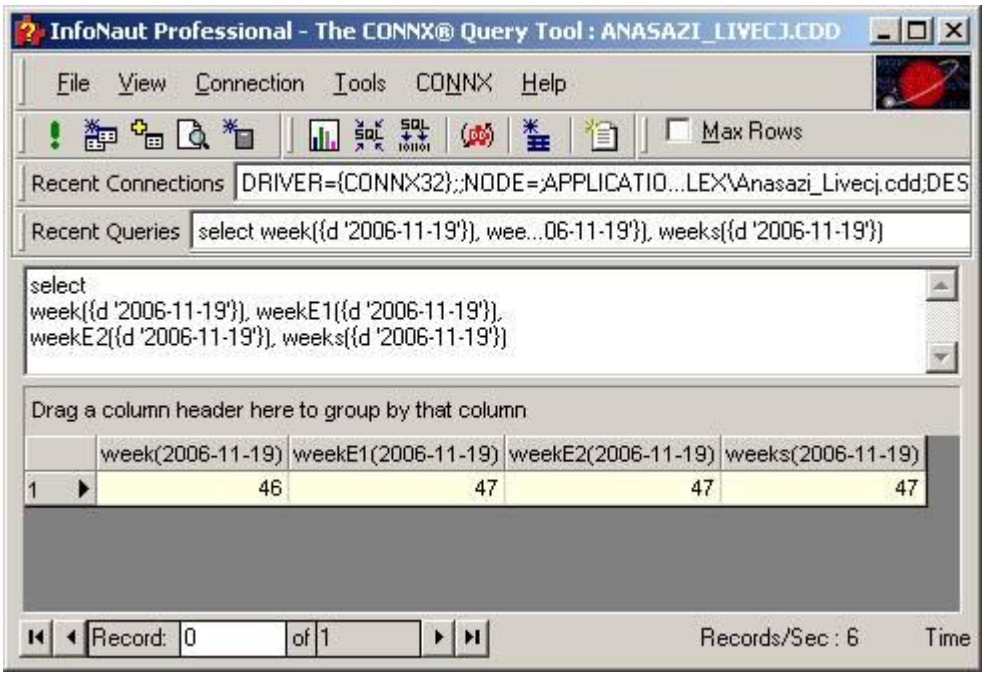
Week 1 is always January 1st to January 7th; week 2 is always January 8th to January 14th, and so on. If the year begins on a Thursday, then each "week" is from Thursday to the following Wednesday.

The absolute week number will always be between 1 and 53. Week 53 will have either one or two days, depending on whether the year is a leap year. If the year is a not a leap year, week 53 will consist of one day: December 31st. If the year is a leap year, week 53 will consist of two days: December 30th and December 31st.

Example:

```
SELECT WEEKS({d '2006-11-19'}) returns 47
```

The results of the SQL expression are shown in the fourth column.



YEAR(date_exp)

Returns the year based on the year field in date_exp as an integer value. The range is data source-dependent.

Example:

```
SELECT YEAR({d '2001-09-13'}) returns 2001
```

CONNXCDD()

This function is used to retrieve the current CONNX data dictionary name.

Example:

```
SELECT connxcdd()
```

CONNX_API_VERSION([catalog_exp])

This function is used to retrieve the version of the API protocol used to communicate between the client and the server. catalog_exp can optionally refer to the catalog name of the connection for which the build information will be returned. Information about the protocol API version can be useful during calls to CONNX Technical Support.

Example:

```
SELECT connx_api_version( 'VSAM_SERVER' )
```

CONNX_SERVER_VERSION([catalog_exp])

This function is used to retrieve build number information of the specified CONNX Server. catalog_exp can optionally refer to the catalog name of the connection for which the build information will be returned. Information about build numbers can be useful during calls to CONNX Technical Support.

Example:

```
SELECT connx_server_version( 'VSAM_SERVER' )
```

CONNX_VERSION()

This function is used to retrieve build number information of the CONNX Client. Information about build numbers can be useful during calls to CONNX Technical Support.

Example:

```
SELECT connx_version()
```

DATABASE()

Returns the name of the current ODBC/JDBC data source. If a DSN-less connection is used, it returns an empty string.

Example:

```
SELECT DATABASE() returns CONNX9SAMPLES32
```

GETCURSORNAME()

This function returns the SQL cursor name.

Example:

```
SELECT GETCURSORNAME()
```

To embed it in a **SELECT ... FOR UPDATE** SQL statement, use the following syntax:

```
SELECT GETCURSORNAME(), CUSTOMERNAME FROM CUSTOMERS_TABLE WHERE CUSTOMERID
= '12345' FOR UPDATE OF CUSTOMERNAME
```

NTUSERNAME()

This function is used to retrieve the current user name.

Example:

```
SELECT ntusername()
```

USER()

Returns the current CONNX user name.

Example:

```
SELECT USER() returns NICKD (current user name)
```

XPUSERNAME()

This function is used to retrieve the fully qualified Windows user name for machines running Windows XP SP2 and above.

Example:

```
SELECT xpusername()
```

SQL Conversion Functions

CASTASCONNXTYPE(CONNXTypeName, ColumnName, [[Precision] [,Scale]])

Converts the specified column using the supplied CONNX Data type, with an optional precision and scale for those CONNX data type that required it.

Example:

```
select
    CastAsCONNXType('PACKED Decimal -> Decimal', numcol, 4, 1) ,
    CastAsCONNXType('Text Date (YYYYMMDD)', stringdate)
from trixie.dbo.test2
```

CNXRawConvert(CONNXTypeName, ColumnName [[,Offset] [,Length] [,Precision] [,Scale] [,Codepage]])

Converts the specified char or binary expression using the supplied CONNX Data type, with an optional offset, length, codpage, precision and scale for those CONNX data type that required it.

Example:

```
select
    CNXRawConvert('PACKED Decimal -> Decimal', binarycol, 4, 3) ,
    CNXRawConvert('Text Date (YYYYMMDD)', stringvalue)
from trixie.dbo.test2
```

CONVERT(exp,datatype)

Converts the expression exp to the specified data type. Conversion can potentially cause truncation as shown in the examples below.

If this size parameter is omitted for the following data types, it defaults to a size of one: Char, Varchar, Binary, Varbinary, Nchar, and Nvarchar.

Valid data types are shown in the following table:

CONVERT Data Types

Data Type	Description
Bigint	8-byte integer
Integer	4-byte integer
Smallint	2-byte integer
Tinyint	1-byte integer
Char(size)	Fixed length character string
Varchar(size)	Variable length character string
Binary(size)	Fixed length binary string
Varbinary(size)	Variable length binary string
Date	ODBC date
Time	ODBC time
Timestamp	ODBC date and time
Double	8-byte double
Float	4-byte real
Bit	bit
Decimal(precision, scale)	Decimal
Numeric(precision, scale)	Numeric
Nchar(size)	Unicode
Nvarchar(size)	Variable length Unicode

Example:

```
CONVERT("123", integer)
```

Returns: 123

```
CONVERT(123, char(3))
```

Returns: "123"

```
CONVERT(123, char(2))
```

Returns: "12"

```
CONVERT(123, char)
```

Returns: "1"

SQL Numeric Functions

ABS(numeric_exp)

Returns the absolute value of numeric_exp.

Example:

```
SELECT ABS(-123456) returns 123456
```

ACOS(float_exp)

Returns the arccosine of float_exp as an angle, expressed in radians.

Example:

```
SELECT ACOS(.75) returns 0.722734247813416
```

ASIN(float_exp)

Returns the arcsine of float_exp as an angle, expressed in radians.

Example:

```
SELECT ASIN(.75) returns 0.848062078981481
```

ATAN(float_exp)

Returns the arctangent of float_exp as an angle, expressed in radians.

Example:

```
SELECT ATAN(.75) returns 0.643501108793284
```

ATAN2(float_exp1, float_exp2)

Returns the arctangent of the x and y coordinates, specified by float_exp1 and float_exp2, respectively, as an angle, expressed in radians.

In most specifications (like POSIX or IEEE) the y coordinate is specified first, followed by the x coordinate. However, the ATAN2 ODBC specification specifies x, then y.

Example:

```
SELECT ATAN2(.75, .50) returns 0.588002603547568
```

CEILING(numeric_exp)

Returns the smallest integer greater than or equal to numeric_exp.

Example:

```
SELECT CEILING(.75) returns 1
```

COS(float_exp)

Returns the cosine of float_exp, where float_exp is an angle expressed in radians.

Example:

```
SELECT COS(.75) returns 0.731688868873821
```

COT(float_exp)

Returns the cotangent of float_exp, where float_exp is an angle expressed in radians.

Example:

```
SELECT COT(.75) returns 1.07342614854938
```

DEGREES(numeric_exp)

Returns the number of degrees converted from numeric_exp radians.

Example:

```
SELECT DEGREES(.785398163397448) returns 45
```

EXP(float_exp)

Returns the exponential value of float_exp.

Example:

```
SELECT EXP(.75) returns 2.11700001661267
```

FLOOR(numeric_exp)

Returns the largest integer less than or equal to numeric_exp.

Example:

```
SELECT FLOOR(75.5) returns 75
```

LOG(float_exp)

Returns the natural logarithm of float_exp.

Example:

```
SELECT LOG(.75) returns -0.287682072451781
```

LOG10(float_exp)

Returns the base 10 logarithm of float_exp.

Example:

```
SELECT LOG10(.75) returns -0.1249387366083
```

MOD(integer_exp1, integer_exp2)

Returns the remainder (modulus) of integer_exp1 divided by integer_exp2.

Example:

```
SELECT MOD(75, 2) returns 1
```

PI()

Returns the constant value of pi as a floating point value.

Example:

```
SELECT PI() returns 3.14159265358979
```

POWER(numeric_exp, integer_exp)

Returns the value of numeric_exp to the power of integer_exp.

Example:

```
SELECT POWER(75, 2) returns 5625
```

RADIANS(numeric_exp)

Returns the number of radians converted from numeric_exp degrees.

Example:

```
SELECT RADIANS(75) returns 1.30899693899575
```

RAND(integer_exp)

Returns a random floating point value between 0 and 1 using integer_exp as the optional seed value.

Example:

```
SELECT RAND(75) returns 0.915692611577849
```

ROUND(numeric_exp, integer_exp)

Returns numeric_exp rounded to integer_exp places right of the decimal point. If integer_exp is negative, numeric_exp is rounded to |integer_exp| places to the left of the decimal point.

Example:

```
SELECT ROUND(75.12345678,3) returns 75.123
```

Example:

```
SELECT ROUND(75.12345676,7) returns 75.1234568
```

SIGN(numeric_exp)

Returns an indicator of the sign of numeric_exp. If numeric_exp is less than zero, -1 is returned. If numeric_exp equals zero, 0 is returned. If numeric_exp is greater than zero, 1 is returned.

Example:

```
SELECT SIGN(.75) returns 1
```

SIN(float_exp)

Returns the sine of float_exp, where float_exp is an angle expressed in radians.

Example:

```
SELECT SIN(.75) returns 0.681638760023334
```

SQRT(float_exp)

Returns the square root of float_exp.

Example:

```
SELECT SQRT(.75) returns 0.866025403784439
```

TAN(float_exp)

Returns the tangent of float_exp, where float_exp is an angle expressed in radians.

Example:

```
SELECT TAN(.75) returns 0.931596459944072
```

TRUNCATE(numeric_exp, integer_exp)

Returns numeric_exp truncated to integer_exp places right of the decimal point. If integer_exp is negative, numeric_exp is truncated to |integer_exp| places to the left of the decimal point.

Example:

```
SELECT TRUNCATE(-75.123456, 3) returns -75.123
```

XOR(numeric_exp, numeric_exp, ...)

Takes parameters and converts them to 64 bit integers, then returns the unsigned 64 bit XOR of those two values. If either value is NULL then the result is NULL.

Example:

```
SELECT xor('1234567890', '9876543210', 9876543210)
returns 1234567890

SELECT xor('1234567890', '9876543210')
returns 8676316216

SELECT xor('1234567890')
returns 0

SELECT xor()
returns SYNTAX ERROR
```

XOR64(numeric_exp, numeric_exp)

XOR64(a,b) which takes exactly two parameters and converts them to 64 bit integers. It then returns the unsigned 64 bit XOR of those two values. If either value is NULL then the result is NULL. If there are more or less than two parameters, then an error is returned.

Example:

```
SELECT xor64('1234567890', '9876543210')
returns 8676316216

SELECT xor64('1234567890')
returns SYNTAX ERROR
```

SQL Decision Tree Functions

CNXNAME (string_exp, nameformat_exp, nameoutput_exp)

The CNXNAME function is used to parse **string_exp** data fields so that they may be reconstituted into a more desired format.

The **string_exp** parameter is the field that contains the data.

The **nameformat_exp** parameter is a string value that is derived from the format of your data. Each part of the name is assigned a character (Last = 'L', First = 'F', Middle = 'M', Suffix (Jr., Sr., III, etc.) = 'S'). If there is a comma separating portions of the name, that is denoted with a 'C'. It is assumed that there will always be a space between portions of the name.

The **nameoutput_exp** parameter is a string value created from the same characters as the nameformat, in addition to a space (' ') character.

Example:

If the data field is in the format "Last<comma><space>First<space>Middle", the nameformat parameter would be 'LCFM'. If you wanted the output to be in the format "First<space>Last", the nameoutput parameter would be 'F L'. Using these values, the final function would resemble the following:

```
SELECT CNXName(NAME, 'LCFM', 'F L') from MyTable
```

CNXPREFERENCE(likeclause_count, like_clause_1, like_clause_2, ... , criteria1_exp, value1_exp, criteria2_exp, value2_exp, ...)

The CNXPreference function accepts an ordered preference list of like clauses and a list of value/output pairs. The function uses the LIKE operator to compare each of the like_clause items to each of the criteria values in the criteria/value pair. The output value is the first value from the criteria/value pairs where the criteria matches any of the like clauses. All of the pairs are first compared against like clause #1, then the pairs are then compared against like clause #2, etc, until a match is found. If no match is found, NULL is returned.

The following example will first attempt to return a local address. If a local address cannot be found, it returns a permanent address.

Example:

File	Example
Contacts	Company
	Contact
	Title
	Phone_Type_1
	Phone_Number_1
	Phone_Type_2
	Phone_Number_2
	Phone_Type_3
	Phone_Number_3
	Address_Type_1
	Address_1
	Address_Type_2
	Address_2
	Address_Type_3
	Address_3

Return the contacts address and phone information. The address should be the contacts mailing address (type 'M'); if a mailing address does not exist, use the permanent address (type 'P'). The phone number should be their office phone number (type 'O'); and if an office phone number does not exist, use the assistant's phone number (type 'A').

The following SQL Statement, using the CNXPreference, returns five columns (Company, Contact, Title, Address, Phone).

```
SELECT
contacts.Company,
contacts.Contact,
contacts.Title,

cnxpreference(2, "%M%", "%P%",
contacts.Address_Type_1,contacts.Address_1,
```

```
contacts.Address_Type_2,contacts.Address_2,
contacts.Address_Type_3,contacts.Address_3) as Address,

cnxpreference(2, "%O%", "%A%",
contacts.Phone_Type_1,Phone_Number_1,
contacts.Phone_Type_2,Phone_Number_2,
contacts.Phone_Type_3,Phone_Number_3) as Phone

FROM

Contacts
```

IF(criteria_exp,then_exp,else_exp)

Returns then_exp if criteria_exp evaluates to a non-zero value.
If criteria_exp evalulates to zero, then the function returns else_exp.

Example:

```
Select If(1, 'YES', 'NO')
```

Returns: YES

Example:

```
Select If(0, 'YES', 'NO')
```

Returns: NO

IFEMPTY (exp1, exp2)

If exp1 is empty (white space) or null, exp2 is returned, otherwise exp1 is returned. The possible data type or type of exp2 must be compatible with the data type of exp1.

Example:

```
SELECT IFEMPTY( NULL , 456) return 456
or
SELECT IFEMPTY(123, 456) return 123
or
SELECT IFEMPTY('', 'ABC') return ABC
```

IFNULL(exp1, exp2)

If exp1 is null, exp2 is returned, otherwise exp1 is returned. The possible data type or types of exp1 must be compatible with the data type of exp2.

Example:

```
SELECT IFNULL(123, 456) returns 123
```

NULLIF(exp1, exp2)

If exp1 equals exp2, then NULL is returned, otherwise exp1 is returned.

Example:

```
SELECT NULLIF(123, 123) returns NULL
SELECT NULLIF(123, 234) returns 123
```

SWITCH(criteria_exp, compare1_exp, return1_exp, compare2_exp, return2_exp,...,[default_return_exp])

Returns return#_exp if criteria_exp evaluates to compare#_exp, or possibly default_return_exp if there are no matches. If the criteria_exp does not match any of the compare expressions, then the default_return_exp will be return if supplied. If the criteria_exp does not match any of the compare expressions and the default_return_exp was not specified, NULL is returned.

Note: DECODE and SWITCH are synonyms; you can use either function and get the same results.

Example:

```
Select Switch(2, 1, 'YES', 2, 'NO', 3, 'POSSIBLY')
```

Returns: NO

Example:

```
Select Switch(3, 1, 'YES', 2, 'NO', 3, 'POSSIBLY')
```

Returns: POSSIBLY

Example:

```
Select Switch(20, 1, 'YES', 2, 'NO', 3, 'POSSIBLY', 'UNKNOWN' )
```

Returns: UNKNOWN

DECODE(criteria_exp, compare1_exp, return1_exp, compare2_exp, return2_exp,...,[default_return_exp])

Returns return#_exp if criteria_exp evaluates to compare#_exp, or possibly default_return_exp if there are no matches. If the criteria_exp does not match any of the compare expressions, then the default_return_exp will be return if supplied. If the criteria_exp does not match any of the compare expressions and the default_return_exp was not specified, NULL is returned.

Note: DECODE and SWITCH are synonyms; you can use either function and get the same results.

Example:

```
Select Decode(2, 1, 'YES', 2, 'NO', 3, 'POSSIBLY')
```

Returns: NO

Example:

```
Select Decode(3, 1, 'YES', 2, 'NO', 3, 'POSSIBLY')
```

Returns: POSSIBLY

Example:

```
Select Decode(20, 1, 'YES', 2, 'NO', 3, 'POSSIBLY', 'UNKNOWN' )
```

Returns: UNKNOWN

COALESCE(exp1, exp2, ...)

Returns the first non-null expression from the list of supplied expressions. If all expressions are NULL, returns NULL. The data types of the supplied expressions must be compatible.

Example:

```
SELECT COALESCE(NULL, NULL, 123, 456) returns 123
```

SQL Join Syntax

Inner Join

CONNX supports two types of inner join syntax.

Syntax 1. <table_1.column1> = <table_2.column2>

Example:

SELECT * FROM table_1, table_2 WHERE Table_1.key = table_2.key

Syntax 2. <table_1> INNER JOIN <table_2> ON <table_1.column1> = <table_2.column2> [AND <table_1.column2 = table_2.column3> ...]

Example:

SELECT * FROM Table_1 INNER JOIN Table_2 ON Table_1.key = Table_2.key

Outer Join

CONNX supports three types of outer join syntax.

Syntax 1. {oj <table_1> LEFT|RIGHT OUTER JOIN <table_2> ON <table_1.column1> = <table_2.column2> [AND <table_1.column2 = table_2.column3> ...] }

Example:

SELECT * FROM {oj Table_1 LEFT OUTER JOIN Table_2 ON Table_1.key = Table_2.key}

or

SELECT * FROM {oj Table_1 RIGHT OUTER JOIN Table_2 ON Table_1.key = Table_2.key}

Syntax 2. <table_1.column1> *= <table_2.column2>

Example:

SELECT * FROM table_1, table_2 WHERE Table_1.key *= table_2.key

Note: Do not use this syntax for Adabas tables. Instead, use the OUTER JOIN syntax in the example below.

Syntax 3. <table_1> LEFT|RIGHT OUTER JOIN <table_2> ON <table_1.column1> = <table_2.column2> [AND <table_1.column2 = table_2.column3> ...]

Example:

SELECT * FROM Table_1 LEFT OUTER JOIN Table_2 ON Table_1.key = Table_2.key

or

SELECT * FROM Table_1 RIGHT OUTER JOIN Table_2 ON Table_1.key = Table_2.key

SQL Extended Functions

{adabasfdtfname <adabas_fdt_file_name>} - Adabas only (optional)

The CONNX extended functionality Is used in conjunction with CREATE TABLE to name the (optional) FDT file name used to create the table.

Example:

```
{adabasfdtfname 'C:\Program Files\Software
   AG\Adabas\V331\Example\FDT\emp.fdt'}
```

If no FDT file name is supplied using adabasfdtfname, CONNX creates an FDT file in memory using the SQL data types that have been appropriately transformed.

{adabasfilename <adabas_file_name>} - Adabas only (optional)

The CONNX extended functionality Is used in conjunction with CREATE TABLE to name the (optional) target file name on the Adabas server.

Example:

```
{adabasfilename 'inv.dat'}
```

If adabasfilename is not used, CONNX attempts to formulate a file name that can be used.

RESTRICTION:

The maximum allowed file name length is 16 characters; 'FNAMECNX01' fulfills the requirement, while 'FNAMECNX01_CNX02_CNX03_CNX04' does not.

{bsearchtempkey}

The bsearchtempkey function causes the method chosen for joining data to be a binary search of an ordered list. This is the fastest method for small to medium sets of data. This method is the default for joins that do not have an index, or where the index use is discarded with forcetempkey.

Example:

```
SELECT a.*, b.*
FROM db.dbo.tab a, db.dbo.tab b
WHERE
a.col_name = b.col_name {bsearchtempkey}
```

Currently, bsearchtempkey is the default method used to process joins, and so the above query would be equivalent to:

```
SELECT a.*, b.*
FROM db.dbo.tab a, db.dbo.tab b
WHERE
a.col_name = b.col_name
```

However, at some time in the future, the default may change or CONNX may heuristically switch between methods if no function is specified.

{autobulkinserton}

This function enables automatic bulk mode processing for standard insert statement. This enables the feature at the statement level.
See AUTOBULKINSERT for more information about this feature.

{autobulkinsertoff}

This function disables automatic bulk mode processing for standard insert statement. This does not override the global configuration setting AUTOBULKINSERT.
This disables the feature at the statement level. See AUTOBULKINSERT for more information about this feature.

{caseinsensitive}

This option overrides the default string comparison behavior of the core SQL engine. By default, the core SQL engine makes case insensitive comparisons. This default behavior can be changed globally with a configuration setting. Additionally the behavior can be changed at a statement level with this escape clause. This setting, nor the global setting will change the case comparison behavior of the target database. If an index is used by the database, the databases comparison normal behavior will occur. This setting only applies to string comparisons done within the CONNX core, outside of the database. CONNX metadata system tables are always case insensitive regardless of these settings. When used with a query that invloves a single table, this function will have no affect on the case-sensitivity of the results. Once this is used it remains in effect as long as the process that is being used to run the query is running. Breaking the connection and then reconnecting does not terminate their effect.

Example:

```
SELECT a.*, b.*
FROM db.dbo.tab a, db.dbo.tab b
WHERE
a.unique_key = b.unique_key
{caseinsensitive} /* force case-insensitive comparisons for non-key
   comparisons */
```

{casesensitive}

This option overrides the default string comparison behavior of the core SQL engine. By default, the core SQL engine makes case insensitive comparisons. This default behavior can be changed globally with a configuration setting. Additionally the behavior can be changed at a statement level with this escape clause. This setting, nor the global setting will change the case comparison behavior of the target database. If an index is used by the database, the databases comparison normal behavior will occur. This setting only applies to string comparisons done within the CONNX core, outside of the database. CONNX metadata system tables are always case insensitive regardless of these settings. When used with a query that invloves a single table, this function will have no affect on the case-sensitivity of the results. Once this is used it remains in effect as long as the process that is being used to run the query is running. Breaking the connection and then reconnecting does not terminate their effect.

Example:

```
SELECT a.*, b.*
FROM db.dbo.tab a, db.dbo.tab b
WHERE
a.unique_key = b.unique_key
{casesensitive} /* force case-sensitive comparisons for non-key
   comparison */
```

{fn enableservertrace}

By executing this extended SQL statement, CONNX dynamically enables server side tracing for the currently active database connections.

Example:

select 1 {fn enableservertrace}

{fn disableservertrace}

By executing this extended SQL statement, CONNX dynamically disables server side tracing for the currently active database connections.

Example:

```
select 1 {fn disableservertrace}
```

{fn flushopenfilecache <tablename>}

By executing this extended SQL statement, CONNX closes all open tables and flushes any open data caches. Table name is an optional parameter. If omitted all tables in the cache are flushed.

Example:

SELECT * FROM customers {fn flushopenfilecache}
to flush all entries in the cache

or

SELECT * FROM customers {fn flushopenfilecache 'customers_rms'}
to flush an individual table

{fn refreshcdd}

By executing this extended SQL statement, CONNX retrieves a new instance of the CDD file before executing the query.

Example:

```
SELECT * FROM customers {fn refreshcdd}
```

{fn updatestatistics}

Updates the index and cardinality information for all table for all connected databases in the data dictionary. Linked databases from linked CDDs will not be updated.

Example:

```
{fn updatestatistics}
```

{fn setfilename <SQL Table Name>, <New File Name>}

By executing the above extended SQL statement, the physical file name can be remapped dynamically. This requires that the record structure of the new file be identical to that of the previous file. Works with any file-based data source, for example, VSAM, C-ISAM, Microfocus, POWER*flex*, DataFlex, and RMS. When using {fn setfilename} multiple times in a query for the same table name such as the union example below, a table alias must be used.

Example1:

```
SELECT * FROM customer {fn setfilename customer,
   'c:\customers\customer.dat'}
```

Example2:

```
SELECT * FROM RMS.dbo.customers_rms a {fn setfilename a ,'orders.dat'}
union all
SELECT * FROM RMS.dbo.customers_rms b {fn setfilename b
   ,'cnxdir:customer.dat'}
union all
SELECT * FROM RMS.dbo.customers_rms c {fn setfilename c
   ,'products.dat'}
union all
SELECT * FROM RMS.dbo.customers_rms d {fn setfilename d
   ,'customers.dat'}
```

{forceadanonukey}

To disable keyed searches on super descriptors that contain NC or NU constituent fields when criteria is not supplied for all columns of the super descriptor.

Example

```
Select * from localhost.dbo.adabas_file_15 where AA='1234'
   {forceadanonukey}
```

{forceadanonukey} only applies to NU or NC compound keys (super descriptors)..

For more information on how and when to use {forceadanonukey}, see Sub / Super Descriptor Handling.

{forceadanukey}

To enable keyed searches on super descriptors that contain NC or NU constituent fields when criteria is not supplied for all columns of the super descriptor.

Example

```
Select * from localhost.dbo.adabas_file_15 where AA='1234'
   {forceadanukey}
```

{forceadanukey} only applies to NU or NC compound keys (super descriptors).

For more information on how and when to use {forceadanukey}, see Sub / Super Descriptor Handling.

{forcetempkey}

The forcetempkey function is used to abandon the database keys in a lookup and force the creation of a temporary key. This technique is valuable when the connection to the database is slow, and is enhanced when an equal join on an unfiltered column is requested. Under such circumstances, the joins are extremely fast. You do not have to specify the forcetempkey escape for equal and outer joins on columns without indexes. A temporary key join is performed automatically.

Example:

```
SELECT a.*, b.*
FROM db.dbo.tab a, db.dbo.tab b
WHERE
a.unique_key = b.unique_key
{forcetempkey} /* force the use of a temporary key (binary key is the
default) */
```

{hashtempkey}

The hashtempkey function causes the method chosen for joining data to use hashing. This is the fastest method for joining large sets of data.

The following example, which assumes there is no index on the column col_name, uses binary search of an ordered table to reduce work in performing the join.

Example:

```
SELECT a.*, b.*
FROM db.dbo.tab a, db.dbo.tab b
WHERE
a.col_name = b.col_name
{hashtempkey}
```

{ignoretimestampfraction}

The ignoretimestampfraction function is useful for comparison of timestamp data across different database systems. If one database system does not support nanoseconds in the timestamp (Oracle is one example of such a database), then we can compare all of the other fields for equality by using this example.

Example:

```
SELECT a.*, b.*
FROM db.dbo.tab a, db.dbo.tab b
WHERE
a.unique_key = b.unique_key
{ignoretimestampfraction} /* do not check fractional seconds */
```

{killstatement <statementID>}

Stops a currently running CONNX query from within the current process. The statement IDs of all running queries within the current process can be viewed with the {showsessions} command.

Example:

{killstatement 27B663C8}

{maxrows #[, offset]}

This keyword limits the number of rows that are returned. An optional offset can be specified as a second parameter. The offset is one based.

Example:

SELECT * FROM customers {maxrows 10}

This will return a maximum of 10 rows of data.

SELECT * FROM customers {maxrows 10,4}

This will return a maximum of 10 rows of data, starting at the 4th row of output.

{nativesql}

This keyword passes a SQL statement to a backend database in its native format.

Example:

```
SELECT * FROM customers {nativesql}
```

{nativetypemode}

This keyword returns the CONNX native type, length, precision, and scale in the description field of the metadata functions that describe the columns. This metadata is available from four different sources:

1. JDBC metadata calls
2. ODBC metadata calls (SQL columns)
3. ADO metadata calls OpenSchema (ADODB.SchemaEnum.adSchemaColumns)
4. CONNX metadata tables (syscnxColumns)

The format of the description field is as follows:

NativeType = ### :NativeSize = ##### : NativePrecision = ##### : NativeScale ###

Example:

```
NativeType = 1 :NativeSize = 20 : NativePrecision = 0 : NativeScale 0
```

{nomaxrows}

This keyword switches off {maxrows #} and reestablishes unlimited row count return.

Example:

SELECT * FROM customers {nomaxrows}

{nopassthrough}

By executing this extended SQL statement, CONNX disallows passthrough of a SQL statement to SQL Server.

Example:

```
SELECT * FROM customers {nopassthrough}
```

To use the DBKEY value for subsequent queries in Rdb databases, use the above defined CONNX SQL extended function "{nopassthrough}" on all queries that reference the DBKEY, or you may receive or transmit the passthrough type DBKEY (which is binary in format). In general, it is easier to manipulate DBKEY values in character mode.

To use the Oid value for subsequent queries in PostgreSQL databases, use the above defined CONNX SQL extended function "{nopassthrough}" on all queries that reference the Oid, or you may receive or transmit the passthrough type Oid (which is binary in format). In general, it is easier to manipulate Oid values in character mode.

{nosqloptimize}

This keyword turns off the CONNX Advanced SQL Optimization for the duration of this single query. This keyword can be appended to any existing SQL statement that CONNX supports.

Example:

SELECT * FROM customers {nosqloptimize}

{nooptimizeoperator}

The {nooptimizeoperator} escape can be used to tell CONNX not to use the prior operator in query optimization. CONNX continues to evaluate the expression, but it is done on the client, rather than potentially using an index, and the associated criteria. For example, using the CONNX sample CUSTOMER table, the following query will be optimized to use the index on CUSTOMERID.

Select * from customers where customerID = 'ALWAO'

To tell CONNX not to use the index (to perform a table scan), use the following SQL statement:

Select * from customers where customerID = {nooptimizeoperator} 'ALWAO'

The {nooptimizeperator} is very useful if you have multiple criterion that have usable indexes, and you want to force a particular index to be used. If there was an index both on the customername and the customerid field, you can force the customerid index to be used as follows:

Select * from customers where customerID = 'TEST3' and customername = {nooptimizeoperator} 'Test name 3'

Conversely if there was an index both on the customername and the customerid field, you can force the customername index to be used as follows:

Select * from customers where customerID = {nooptimizeoperator} and 'TEST3' customername = 'Test name 3'

{notempkey}

This keyword tells CONNX not to create a temp key, and to use the Cartesian product instead. Without the use of the {notempkey} escape clause, CONNX creates a temporary key to perform the join. With the {notempkey} escape clause, CONNX forms a brute force nested loop to perform the join.

Example:

```
SELECT * FROM customers a, customers b WHERE a.customername = b.customername and
a.customeraddress = b.customeraddress  {notempkey}
```

{passthrough <database name>}

By executing this extended SQL statement, CONNX enables passthrough of a SQL statement to a database.

Example:

SELECT * FROM customers {passthrough <database name>}

{recordmismatch}

The recordmismatch function is used to locate differences between two tables in an equal join. The joining columns must contain all of the columns of a unique index. When the recordmismatch function isused, records only appear in the query where mismatches occur. Therefore, if a database has ten thousand records and exactly two records are different, then the query returns two records in the results set.

Example:

```
SELECT a.*, b.*
FROM rms_db.dbo.tab a, oracle_db.dbo.tab b
WHERE
a.unique_key = b.unique_key
{recordmismatch} /* check for differences between the records*/
```

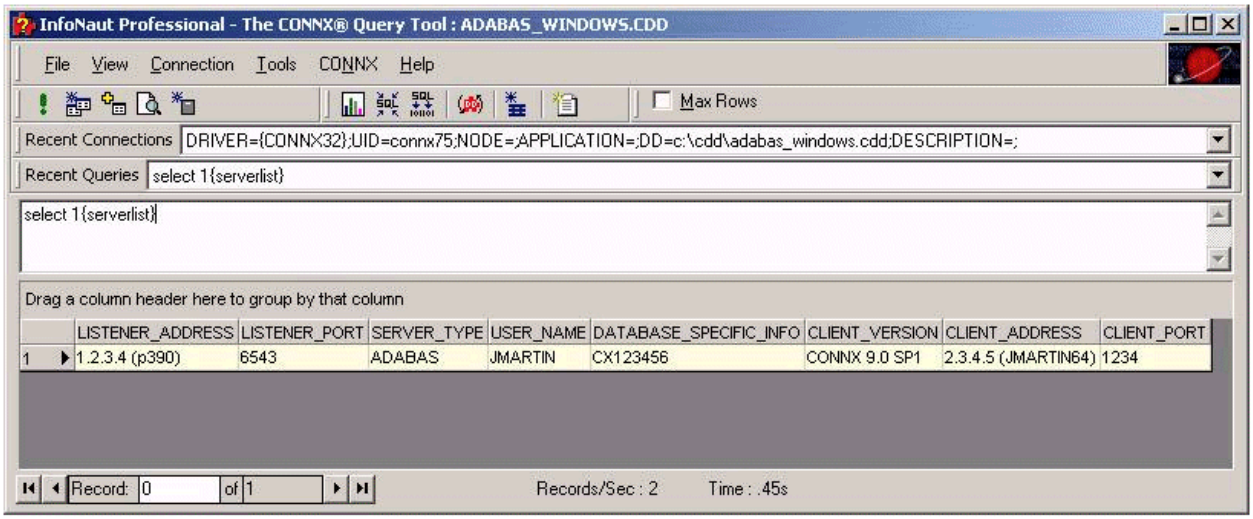
{serverlist} (Adabas only)

By executing this extended SQL statement in the InfoNaut querying tool, session management data for Adabas and the Adabas SQL Gateway (CONNX for Adabas) can be viewed.

Example:

{serverlist}

Note: To view results in InfoNaut, preface the command with "Select 1" as shown below.



{setadapassword}

At runtime, you can provide a password for any Adabas file using the following extended CONNX syntax:

```
{fn setadapassword <table alias> , <password>}
```

Example: To specify a password for the CUSTOMERS_ADABAS Table, issue the following SQL statement:

```
SELECT * FROM adabas_windows.dbo.CUSTOMERS_ADABAS {fn setadapassword
CUSTOMERS_ADABAS, PASSWORD}
```

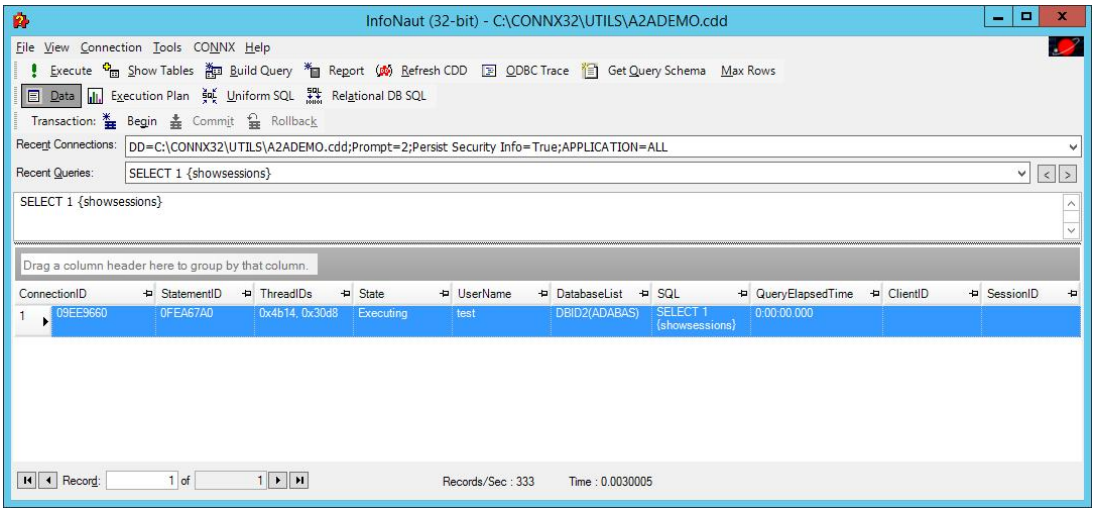
If you have defined a global password in the data dictionary, the password specified with the SQL statement will override the global password.

{showsessions}

This keyword will return a list of active CONNX connections & statements within the current process. This is useful for diagnostics of applications and servers that manage multiple CONNX connections, such as the JDBC server, or a web server. Individual statements returns from {showsessions} can be killed with the {killstatement} command.

Example:

select 1 {showsessions}



{statistics}

By adding the expression noted above to the end of an SQL statement, CONNX will return the query execution plan instead of the requested data.

Example:

SELECT * FROM customers, orders WHERE customers.name = "Meyer" and customers {statistics}

{statistics} SQL Extended Function

Step	Table	Action
1	Customers	CONNX performs a keyed lookup using index #2.
2	Orders	CONNX performs a crosstable lookup using data retrieved from table "Customers" and using index #40.

{startconnectionpooling}

This keyword enables connection pooling from the current point forward if connection pooling is disabled in the CONNX Registry file (see CONNX Registry File Settings). All future disconnects are pooled. This command has no effect if connection pooling is already enabled.

Example:

{startconnectionpooling}

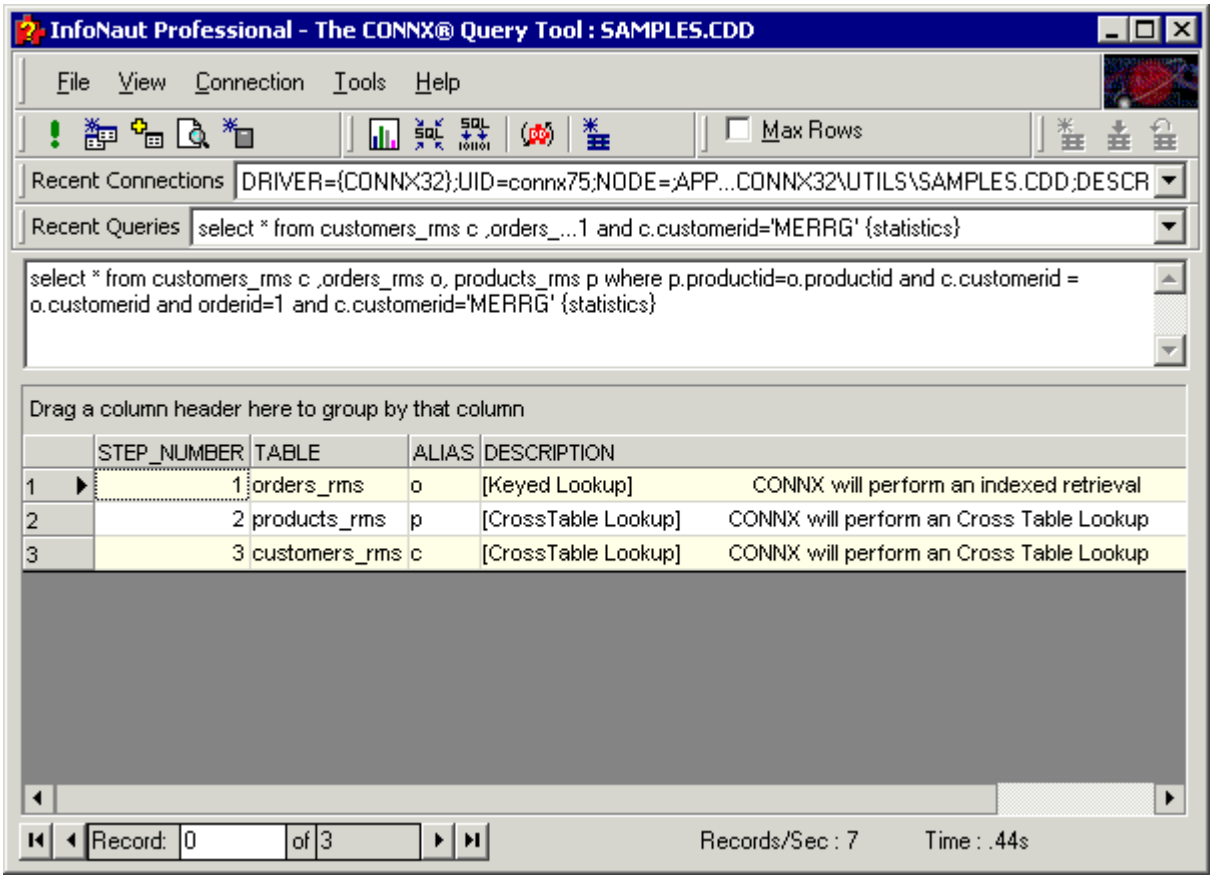
{startwiththistable}

This keyword provides a hint CONNX to start with a specific table in a query plan. The hint may not be honored, depending on available query plans.

Example:

```
select * from customers_rms c, orders_rms {startwiththistable} o,
   products_rms p where p.productid = o.productid and c.customerid =
   o.customerid
```

CONNX determines an execution plan based on the various tables/fields that are used. To view the execution plan for a query, the extended CONNX function {statistics} can be placed after the query. The results that are returned are the query plan.



In this example, two indexes are used. CONNX determines that ORDERS_RMS is to be used as the starting table because of the use of an index in the query (ordered=1).

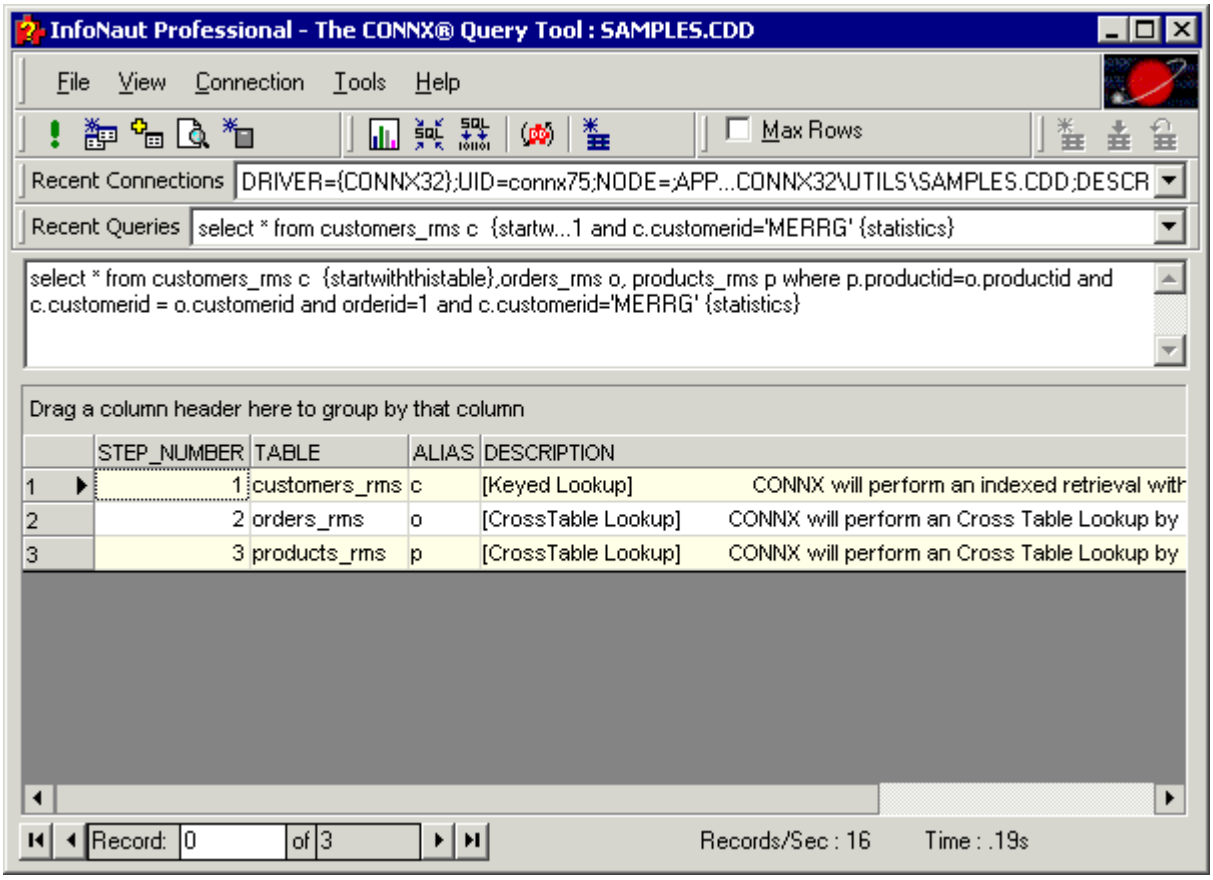
In most cases, CONNX selects the most efficient plan. However, the user can force the choice of the starting table by using the extended function {startwiththistable}.

Example:

```
select * from customers_rms c  {startwiththistable} , orders_rms  o ,
   products_rms p   where
```

```
p.productid = o.productid and c.customerid = o.customerid  and
   orderid=1 and c.customerid='MERRG' {statistics}
```

This query produces the following query plan:



{stopconnectionpooling}

This keyword turns off connection pooling as established in the CONNX Registry file (see CONNX Registry File Settings) and disconnects any pooled connections. All current connections remain open.

Example:

{stopconnectionpooling}

Note: {stopconnectionpooling} should be used to shut down connection pooling if an application calls the Windows API "TerminateProcess," which causes unpredictable behavior in the CONNX connection pooling shutdown process.

{transactmode readonly} (Rdb only)

Execute the above extended SQL statement to commit the transaction and change the transaction mode to read only.

{transactmode readwrite}

Execute the above extended SQL statement to commit the transaction and change the transaction mode to read/write.

{usekey}

When there are multiple choices for keys within a table, CONNX selects the optimal key choice.

There are two ways to manually override the selected key.

- Place {usekey <key no.> } directly after the table name. CONNX will use the specified key, when possible. However, the usage of the key is not guaranteed.
- CONNX will accept a string that contains all of the key field names, separated by commas, instead of the key number. Place {Usekey '<keycolum#1>[,<keycolum#2>...]'} directly after the table name.

Example:

Key number

```
SELECT * FROM equipmnt_rms {usekey 2} where equipmnt_rms.location='MIS
   DEPT.' and equipmnt_rms.description='TRAILBLAZER MODEM'
```

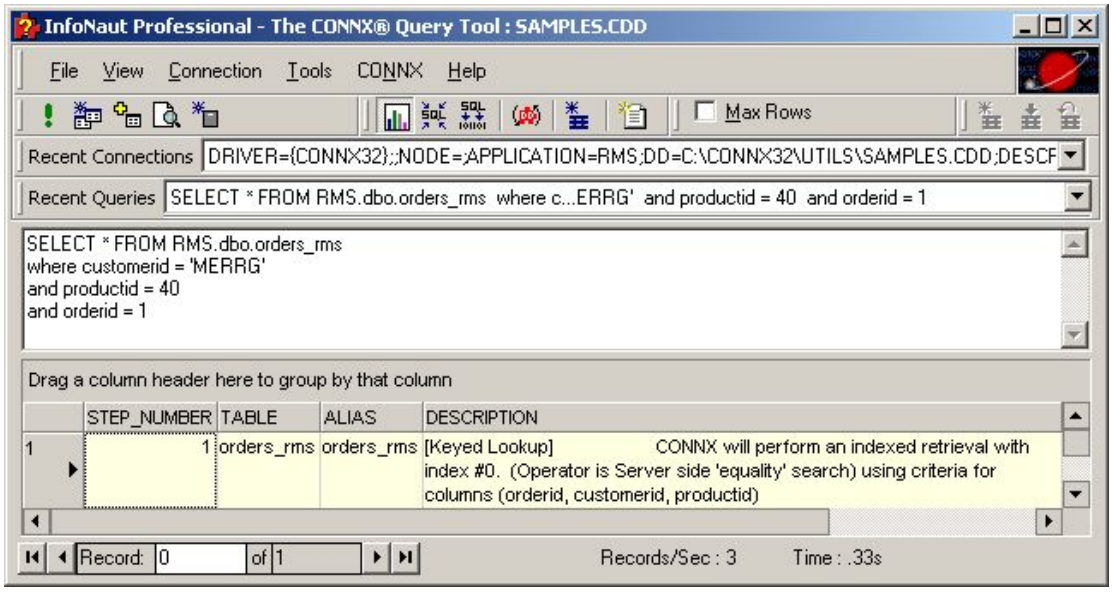
Key field names

The sample PRODUCT table that comes with CONNX has three keys on it:

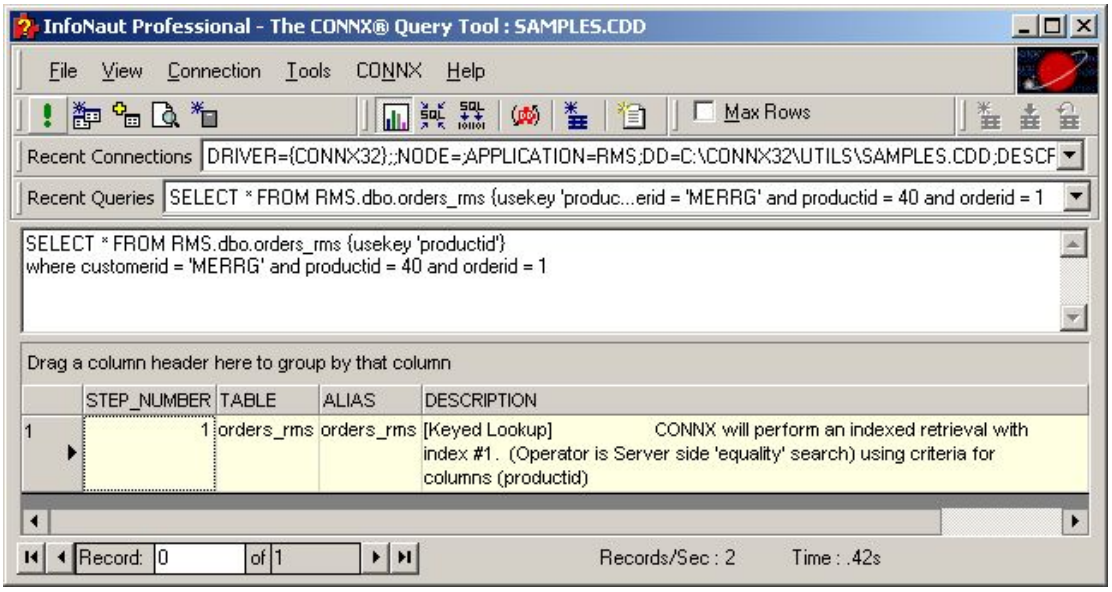
Key #0 = a composite key of orderid, customerid, and productid
Key #1 = customerid
Key #2 = productid

When issuing the following SQL statement, CONNX will by default select key #0 as the best key candidate.

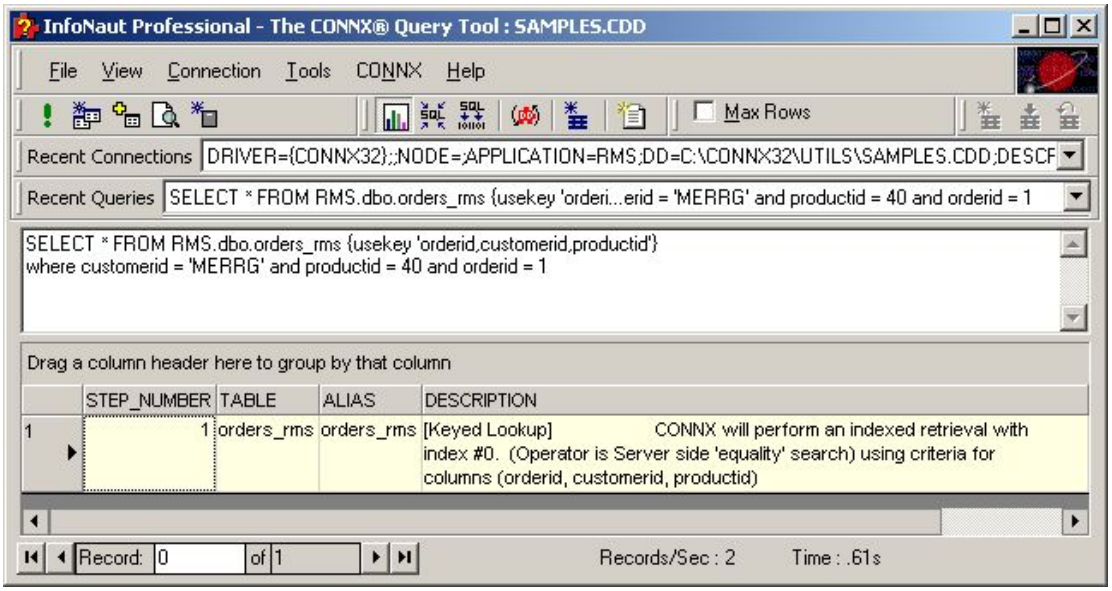
```
SELECT * FROM RMS.dbo.orders_rms
where customerid = 'MERRG'
and productid = 40
and orderid = 1
```



The following example uses tells CONNX to use the productid key.



The following example uses tells CONNX to use the orderid, customerid, productid composite key.



autocounter()

This expression is used to create ascending lists of numbers. It can be used to create mock data or simple, unique ascending values, although it is not guaranteed to be serial in nature or to start at any particular value.

Example:

```
SELECT autocounter() from customers
```

Another example would be:

Example:

```
SELECT 1e10 -autocounter() * 100 from customers
```

Since autocounter() returns a number, it can be used in math expressions like any other function. Here we have selected 10 billion and subtracted autocounter() times 100 to create a descending value.

BIBXREF(field, prefix, index)

This is a function for a specific customer that scans the data in the record area defined by field looking for the n-th instance, as identified by index, of the field prefixed by the special character prefix. Note that the range of possible prefixes is preset by the customer specific tables for which this function was implemented.

Example:

```
SELECT BIBXREF(data, 241, 2) FROM MyTable
```

cnxcurrentconnections()

Returns the current number of connections to data servers (includes JDBC server).

Example:

```
SELECT cnxcurrentconnections()
```

cnxcurrentstatements()

Returns the current number of active SQL Statements on the data server (includes JDBC Server).

Example:

```
SELECT cnxcurrentstatements()
```

cnxpeakconnections()

Returns the peak number of connections (high water mark) to data servers (includes JDBC server).

Example:

```
SELECT cnxpeakconnections()
```

cnxpeakstatements()

returns to peak number of simultaneous SQL Statements (high water mark) active on the data server (includes JDBC server).

Example:

```
SELECT cnxpeakstatements()
```

cnxtotalconnections()

Returns the total number of connections to data servers over the life of the data servers (includes JDBC server).

Example:

```
SELECT cnxtotalconnections()
```

cnxtotalstatements()

Returns the total number of SQL Statements created over the life of the data server (includes JDBC server).

Example:

```
SELECT cnxtotalstatements()
```

CNXFORCECHAR (binary_exp)

This expression is useful in the manipulation of the BINARY data type as it forces the data type of the expression to convert to CHAR. No additional data type checking or data conversion is performed.

Example:

SELECT CNXFORCECHAR (BinaryField) from MyTable

The above syntax converts the column BinaryField to character. It does not attempt to convert the binary field to a character hexadecimal representation.

CNXFORCEBINARY (string_exp)

This expression is useful in the manipulation of the BINARY data type as it forces the data type of the expression to convert to BINARY. No additional data type checking or data conversion is performed.

Example:

SELECT CNXFORCEBINARY (CharacterField) from MyTable

The above syntax converts the column CharacterField to binary. It does not assume that the character field is a hexadecimal representation of a binary field.

CNXSLEEP(numeric_exp)

This function is used to create a delay, where numeric_exp represents the number of milliseconds to wait.

Example:

```
SELECT cnxsleep(5000) from customers
```

This example creates a pause of 5 seconds for every record in the customers_rms table.

ILIKE

The SQL primary command ILIKE, also known as the ILIKE operator, can be used for wildcard comparison against String values to filter specific data. The ILIKE operator has three special characters, each with a different usage: the percent sign (%), the underscore (_), and the caret (^).

Important: *The "I" in ILIKE represents an abbreviated form of the word "insensitive"; therefore, the ILIKE operator is NEVER case-sensitive (even if the database tables are case-sensitive).*

% multi-character wildcard

The use of the percent sign (%) in a ILIKE clause retrieves matches of zero or more characters.

Example:

If the column TEST contains 'ABCDEF' the expression TEST ILIKE '%F' returns TRUE, the expression TEST ILIKE '%C%' returns TRUE, and the expression TEST ILIKE '%G' returns TRUE.

Likewise, the expression TEST ILIKE 'f%' returns TRUE.

_ single-character wildcard

The use of an underscore (_) in a ILIKE clause retrieves matches of a single character.

Example:

If the column TEST contains 'GHIJKLM' the expression TEST ILIKE 'G_IJKLM' returns TRUE, and the expression TEST ILIKE 'A_IJKLM' returns TRUE.

Likewise, the expression TEST ILIKE 'A_ijkLM' returns TRUE.

^ literal character identifier

The use of the caret (^) in a ILIKE cause permits the use of any of the wildcard characters %, _ or ^. If the ^ character is placed in front of either of the other two wildcard characters, the characters following are treated as normal literals.

Example:

If the column TEST contains '10% of the data' the expression TEST ILIKE '10^% of the data' returns TRUE, the expression TEST ILIKE '10%data' returns TRUE, and the expression TEST ILIKE '10data' returns FALSE.

Important: The Microsoft Jet Engine for Microsoft Access uses an asterisk (*) as the wildcard character instead of the percent sign (%). Replace the percent sign (%) wildcard with an asterisk (*) when using Microsoft Access, or when using other DAO-compliant applications.

SLIKE

The SQL primary command SLIKE, also known as the SLIKE operator, can be used for wildcard comparison against String values to filter specific data. The SLIKE operator has three special characters, each with a different usage: the percent sign (%), the underscore (_), and the caret (^).

Important: *The "S" in SLIKE represents an abbreviated form of the word "sensitive"; therefore, the SLIKE operator is ALWAYS case-sensitive (even if the database tables are NOT case-sensitive).*

% multi-character wildcard

The use of the percent sign (%) in a SLIKE clause retrieves matches of zero or more characters.

Example:

If the column TEST contains 'ABCDEF' the expression TEST SLIKE '%F' returns TRUE, the expression TEST SLIKE '%C%' returns TRUE, and the expression TEST SLIKE '%G' returns FALSE.

Likewise, the expression TEST SLIKE 'f%' returns FALSE.

_ single-character wildcard

The use of an underscore (_) in a SLIKE clause retrieves matches of a single character.

Example:

If the column TEST contains 'GHIJKLM' the expression TEST SLIKE 'G_IJKLM' returns TRUE, and the expression TEST SLIKE 'A_IJKLM' returns FALSE.

Likewise, the expression TEST SLIKE 'A_ijkLM' returns FALSE.

^ literal character identifier

The use of the caret (^) in a SLIKE cause permits the use of any of the wildcard characters %, _ or ^. If the ^ character is placed in front of either of the other two wildcard characters, the characters following are treated as normal literals.

Example:

If the column TEST contains '10% of the data' the expression TEST SLIKE '10^% of the data' returns TRUE, the expression TEST SLIKE '10%data' returns TRUE, and the expression TEST SLIKE '10data' returns FALSE.

Important: The Microsoft Jet Engine for Microsoft Access uses an asterisk (*) as the wildcard character instead of the percent sign (%). Replace the percent sign (%) wildcard with an asterisk (*) when using Microsoft Access, or when using other DAO-compliant applications.

cnxclientdefaultcodepage()

This returns the default codepage of the client operating system as detected by CONNX.

Example:

```
SELECT cnxclientdefaultcodepage()
```

cnxclientcodepage()

This returns the CONNX codepage of the client operating system, including any override value in the sqlregisty or CONNXRegistry.

Example:

```
SELECT cnxclientcodepage()
```

SQL Statistical Functions

Introduction: Statistical Functions

All of the CONNX Statistical Functions are aggregate functions. They can be divided roughly into three classes: calculation of central tendency, calculation of dispersion, and calculation of shape.

Calculation of central tendency means finding out what observations are likely.

Calculation of dispersion shows how scattered the data is. One example of dispersion is range, which is the difference between the biggest and smallest value in the set.

Calculation of shape defines the shape of the curve of our observations. Unusually shaped populations can be detected through examination of skew and kurtosis. Skew shows whether there is an abundance of small observations or an abundance of large observations. Kurtosis shows whether distribution is flat with tiny tails, or sharply peaked with large tails. The skewness and kurtosis of the normal distribution are zero.

Many of the statistical functions come in two flavors:

- For the population (the result is a parameter)
- For the sample (the result is a statistic)

Important: Use the population-specific function only if your data set contains a measurement for each and every member of the complete population of objects and all of them are included in the query.

AVEDEVMEAN(numeric_exp)

Returns the Average Deviation from the Mean for the Population of numeric_exp.

Description

To discern how scattered observations are about some central value, choose either the median or the mean for the central value. If you should choose the mean, average deviation from the mean value is one measure of dispersion. The *average deviation* (also known as the *mean deviation*) is the average absolute difference between the observed values and the arithmetic mean (average) for all values in the data set. Sometimes, the calculation is performed using distance from the median instead of the mean (See AVEDEVMEDIAN). The term *average deviation* is something of a misnomer, since by definition of the mean the sum of all deviations about the mean are zero except for possible rounding errors. The true *average deviation* cannot be used since that sum is always zero, which says nothing about how far the average observation is from the mean. Use the absolute value of the difference between each observation and the mean to find a correct answer. While the mean deviation is sometimes called the *mean absolute deviation*, this usage is not strictly correct unless the data is categorized into bins first. For estimating population standard deviation in a normal population, the mean deviation is not as efficient as the sample standard deviation.

$$AveDevMean = \frac{1}{N}\sum_{i=1}^{N} | x_i - \bar{x} |$$

In plain English, we take the sum of the absolute value of all observations minus the mean and divide that sum by the number of observations (N).

Parameters

numeric_exp must be a number or a numeric expression.

Comments

Flaws exist in using this calculation. If a sample is taken and the accuracy of a process using our sample is estimated, a different estimate if the result if the sample is cut into two smaller samples and the calculation is performed on the subsamples. The amount of underestimation is not only a function of the sample size, but also a function of the probability of the distribution of the errors in measurement. This strongly indicates against the use of small samples for performing this calculation.

There are also some special merits in this calculation. It is not unheard of to be dealing with a distribution whose variance does not exist. In this case, all higher moments and derivative measures such as the standard deviation are useless as a measure of the data's width around its mean. Attempted calculations of the statistics using higher moments produce random results. The average deviation does not suffer from this defect but is a good measure for estimation for broad distributions with a significant number of outlier points. Higher order moments or statistics involving higher powers of the input data are less robust than lower moments or statistics that involve only linear sums or counting.

AVEDEVMEDIAN(numeric_exp)

Returns the Average Deviation from the Median for the Population of numeric_exp.

Description

To discern how scattered observations are about some central value, choose either the median or the mean for the central value. If you should choose the median, average deviation from the median value is one such measure of dispersion. The *average deviation* (also known as the *mean deviation*) about the median is the average absolute difference between the observed values and the median (central value in an ordered set) for all values in the data set. For any fixed sample, choosing the median rather than some other measure of central tendency minimizes the mean deviation. Sometimes, the calculation is performed using distance from the mean instead of the median (see AVEDEVMEAN). To calculate the average deviation of the median, use the absolute value of the difference between each observation and the median. While the mean deviation is sometimes called the *mean absolute deviation*, this is not strictly correct unless the data is categorized into bins first. For estimating population standard deviation in a normal population, the mean deviation is not as efficient as the sample standard deviation.

$$AveDevMedian = \frac{1}{N}\sum_{i=1}^{N} | x_i - x_{median} |$$

In other words, we take the sum of the absolute value of all observations minus the median and divide that sum by the number of observations (N).

Parameters

numeric_exp must be a number, or a numeric expression.

Comments

Flaws exist in using this calculation. If a sample is taken and the accuracy of a process using a sample is estimated, the result is a different estimate if the sample is divided into two smaller samples and the calculation is performed on the subsamples. The amount of underestimation is not only a function of the sample size, but also a function of the probability of the distribution of the errors in measurement.

There are also some special merits in this calculation. It is not unheard of to be dealing with a distribution whose variance does not exist. In this case, all higher moments and derivative measures such as the standard deviation are useless as a measure of the data's width around its mean. Attempted calculations of the statistics using higher moments produce random results. The average deviation does not suffer from this defect but is a good measure for estimation for broad distributions with a significant number of outlier points. Higher order moments or statistics involving higher powers of the input data are less robust than lower moments or statistics that involve only linear sums or counting.

COEFVARPCT(numeric_exp)

Returns the Coefficient of Variation in Percentage for the Population of numeric_exp.

Description

When the averages for two or more different samples are not equal you need a measure of relative dispersion. The Coefficient of Variation Percentage for the sample is such a measure. The Coefficient of Variation Percentage yields information about the sample standard deviation relative to the mean. Thus it could be thought of as the magnitude of the sample standard deviation. This quantity is dimensionless, and therefore has the advantage of being independent of the units of measurement. It is calculated as follows:

$$CV\% = \left(\frac{s}{\bar{x}}\right)100$$

In other words, the population coefficient of variation percentage is the standard deviation of the sample divided by the mean, expressed as a percentage.

Parameters

numeric_exp must be a number, or a numeric expression.

Comments

See also COEFVARPCTP

COEFVARPCTP(numeric_exp)

Returns the Coefficient of Variation in Percentage for the Population of numeric_exp.

Description

When the averages for two or more different samples are not equal you need a measure of relative dispersion. The Coefficient of Variation Percentage for the Population is such a measure. The Coefficient of Variation Percentage yields information about the Standard Deviation relative to the mean. Thus it could be thought of as the magnitude of the Standard Deviation. This quantity is dimensionless, and therefore has the advantage of being independent of the units of measurement. It is calculated as follows:

$$CV\% = \left(\frac{\sigma}{\bar{x}}\right)100$$

In other words, the population coefficient of variation percentage is the standard deviation of the population divided by the mean, expressed as a percentage.

Parameters

numeric_exp must be a number, or a numeric expression.

Comments

COEFVARPCTP should be used only if we use observations for the entire population. For a sample from the population, use COEFVARPCT instead.

FIRST(value_exp)

Returns the first value in the unsorted population of value_exp.

Description

This function returns the 1st value of the unsorted population.

Example

V_1	V_2	V_3	V_4	V_5	V_6
5	25	2	25	25	30

The above example would return 5, because it is the 1st value in the population.

Parameters

value_exp can be an expression of any SQL data type.

KTHLARGEST(numeric_exp, K_numeric_exp)

Returns the Kth largest item indicated by k_numeric_exp in the population of numeric_exp.

For any K supplied, there will be at most K-1 items that are larger. If you ask for K=1, then zero elements are larger; therefore you must have the largest item. If you ask for K=20, then up to 19 elements can be larger.

For the numeric column L with values:

1,2,3,2,5,6,7,8,9,4,2,22

KTHLARGEST(L, 1) /* Ask for the 1st largest element from column L */

returns 22

KTHLARGEST(L, 2) /* Ask for the 2nd largest element from column L */

returns 9

The result is the same as sorting the list (descending, ascending for KTHSMALLEST) and then picking the Kth element from the list. But it is much faster than sorting.

Parameters:

numeric_exp must be a number, or a numeric expression.

k_numeric_exp must be a number, or a numeric expression.

KTHSMALLEST(numeric_exp, K_numeric_exp)

Returns the Kth smallest item indicated by k_numeric_exp in the population of numeric_exp.

For any K supplied, there will be at most K-1 items that are smaller. If you ask for K=N (where N is the total number of things in the list), then zero elements are larger; therefore you must have the smallest item. If you ask for K=20, then up to 19 elements can be smaller.

For the numeric column L with values:

1,2,3,2,5,6,7,8,9,4,2,22

KTHSMALLEST(L, 1) /* Ask for the 1st smallest element from column L */

returns 1

KTHSMALLEST(L, 2) /* Ask for the 2nd smallest element from column L */

returns 2

KTHSMALLEST(L, 3) /* Ask for the 3rd smallest element from column L */

returns 2

This may seem surprising, but if you sorted the list it would look like this: 1,2,2,2,3,4,5,6,7,8,9,22 and so the 2nd, 3rd and 4th smallest elements of the list are all 2.

The result is the same as sorting the list (ascending, descending for KTHLARGEST) and then picking the Kth element from the list. But it is much faster than sorting.

Parameters:

numeric_exp must be a number, or a numeric expression.

k_numeric_exp must be a number, or a numeric expression.

KURTOSIS(numeric_exp)

Returns the Kurtosis of the population of numeric_exp, where numeric_exp is a sampling of a larger population..

Description

Kurtosis is used in distribution analysis to describe how big the tails are for a distribution. Kurtosis indicates the likelihood of an event far away from the average. Kurtosis is based on the size of a distribution's tails. A distribution with approximately the same kurtosis as the normal distribution is called mesokurtic (meaning medium-curved). The mesokurtic kurtosis of a normal distribution is 0. If a distribution is short and flat with small tails it is described as platykurtic (flat-curved). For a platykurtic distribution, individual observations are spread out fairly uniformly across their range. If a distribution is tall and slender with relatively large tails it is spoken of as leptokurtic (slender-curved). For a leptokurtic distribution, observations tend to cluster densely about some particular point in the range.

The following formula can be used to calculate kurtosis of the sample from the raw moments:

$$Kurtosis_{sample} = \frac{\left(\frac{\sum_{i=1}^{N} x_i^4}{N}\right) - 4\left(\frac{\sum_{i=1}^{N} x_i}{N}\right)\left(\frac{\sum_{i=1}^{N} x_i^3}{N}\right) + 6\left(\frac{\sum_{i=1}^{N} x_i}{N}\right)^2\left(\frac{\sum_{i=1}^{N} x_i^2}{N}\right) - 3\left(\frac{\sum_{i=1}^{N} x_i}{N}\right)^3}{s^4} - 3$$

where *s* is the standard deviation of the sample. Subtract 3 because the kurtosis of the normal distribution is 3 (without the subtraction). By subtracting 3, the direction of the kurtosis can be seen by examining its sign. Kurtosis ranges from -2 (highly platykurtic) to 0 (mesokurtic) to +infinity (highly leptokurtic).

Parameters

numeric_exp must be a number, or a numeric expression.

KURTOSISP(numeric_exp)

Returns the Kurtosis of the Population of numeric_exp.

Description

Kurtosis is used in distribution analysis to describe how big the tails are for a distribution. Kurtosis indicates the likelihood of an event far away from the average. Kurtosis is based on the size of a distribution's tails. A distribution with approximately the same kurtosis as the normal distribution is called mesokurtic (meaning medium-curved). The mesokurtic kurtosis of a normal distribution is 0. If a distribution is short and flat with small tails it is described as platykurtic (flat-curved). For a platykurtic distribution, individual observations are spread out fairly uniformly across their range. If a distribution is tall and slender with relatively large tails it is spoken of as leptokurtic (slender-curved). For a leptokurtic distribution, observations tend to cluster densely about some particular point in the range.

The following formula can be used to calculate kurtosis of the population from the raw moments:

$$Kurtosis_{population} = \frac{\left(\frac{\sum_{i=1}^{N} x_i^4}{N}\right) - 4\left(\frac{\sum_{i=1}^{N} x_i}{N}\right)\left(\frac{\sum_{i=1}^{N} x_i^3}{N}\right) + 6\left(\frac{\sum_{i=1}^{N} x_i}{N}\right)^2\left(\frac{\sum_{i=1}^{N} x_i^2}{N}\right) - 3\left(\frac{\sum_{i=1}^{N} x_i}{N}\right)^3}{\sigma^4} - 3$$

where *sigma* is the standard deviation of the population. Subtract 3 from kurtosis because the kurtosis of the normal distribution is 3 (without the subtraction). By subtracting 3, the direction of the kurtosis can be seen by examining its sign.

Parameters

numeric_exp must be a number, or a numeric expression.

LAST(value_exp)

Returns Last Value in the unsorted population of value_exp.

Description

This function returns the nth value of the unsorted population of n elements.

Example

V_1	V_2	V_3	V_4	V_5	V_6
5	25	2	25	25	30

The above example would return 30, because it is the 6th value in the population.

Parameters

value_exp can be an expression of any SQL data type.

MEDIAN(numeric_exp)

Returns the Median Value of the population of numeric_exp.

Description

The median is used to indicate the center of a given population at the point where the distribution of scores is divided in half. Given a population of "n":

If n is **odd**, the median is the middle data elements of the sorted list of values.

If n is **even**, the median is the average of the data element between the middle data elements.

Example:

With a resultset of the following four values (v_1, v_2, v_3, v_4), the median would be the average (or mean) of values v_2 and v_3.

v_1	v_2	v_3	v_4
5	20	30	50

The median would be 25.

Parameters

numeric_exp must be a number, or a numeric expression.

Note: If the number of data points is less than QUANTILE_ESTIMATION_THRESHHOLD, then the calculation for a median or quantile will proceed as follows:

1. If the data is not sorted on any columns for which calculation is requested, the data will be sorted.
2. The quantile position is calculated by linear interpolation.

If the number of data points is greater than or equal to QUANTILE_ESTIMATION_THRESHHOLD, then the calculation for a median or quantile will proceed as follows:

The quantile position will be estimated by the linear time selection algorithm "Randomized-Select" as described in Introduction to Algorithms by Cormen, Leiserson, and Rivest, p. 187.

MIDDLE(value_exp)

Returns the Middle Value of the unsorted population of value_exp.

Description

This function returns the middle value of the unsorted population of n elements.
Middle will return the (n+1)/2'th value, performing integer division.

Example:

V_1	V_2	V_3	V_4	V_5	V_6
5	25	2	25	25	30

The above example would return 2, because it is the 3rd value in the population ((6+1)/2 = 3.5 ~=3).

V_1	V_2	V_3	V_4	V_5
5	25	2	25	25

The above example would return 2, because it is the 3rd value in the population ((5+1)/2 = 3).

Parameters

value_exp can be an expression of any SQL data type.

MODE(value_exp)

Returns the mode of the population of value_exp.

Description

The mode is used to indicate the location of the center of a population. The mode is not an indicator of the frequency of the score that occurs most often, but rather the actual value of the score.

Example:

V_1	V_2	V_3	V_4	V_5	V_6
5	25	2	7	25	30

With a resultset of the following six values (v_1, v_2, v_3, v_4, v_5, v_6), the mode would be the 25, because it occurs the most number of times.

V_1	V_2	V_3	V_4	V_5	V_6
5	25	8	7	25	8

In the above example, the Mode would be NULL, because there is more than one value that repeats at the highest frequency (both 25 and 8 repeat twice). This is called a "multi-modal" population.

Parameters

value_exp can be an expression of any SQL data type.

MULTIMODALCOUNT(value_exp)

Returns the count of unique values, if any, that makes the Population of value_exp multi modal.

Description

This function returns the number of unique values that cause a population to be multimodal. In other words, this function only returns results if the population is multimodal. If the population is not multimodal, this function returns 0.

Example:

V_1	V_2	V_3	V_4	V_5	V_6
5	25	2	25	25	30

With a resultset of the following six values (v_1, v_2, v_3, v_4, v_5, v_6), the multimodalcount would be the 0, because the population is not multimodal (there is a single mode, 25).

V_1	V_2	V_3	V_4	V_5	V_6
5	25	8	7	25	8

In the above example, the multimodalcount would be 2, because the population is multimodal, and the number of unique values that make the population multimodal is two (8 and 25).

Parameters

value_exp can be an expression of any SQL data type.

MULTIMODALOCCUR(value_exp)

Returns the number of times the mode value occurs in the population of value_exp.

Description

This function returns the number of times the mode occurs (even in multi-modal populations).

Example:

V_1	V_2	V_3	V_4	V_5	V_6
5	25	2	25	25	30

With a resultset of the following six values (v_1, v_2, v_3, v_4, v_5, v_6), the multimodaloccur would be the 3, because the mode, 25, occurs three times.

V_1	V_2	V_3	V_4	V_5	V_6
5	25	8	7	25	8

In the above example, the multimodaloccur would be 2, because there is more than one value that repeats at the highest frequency (both 25 and 8), and they both repeat twice.

Parameters

value_exp can be an expression of any SQL data type.

QUANTILE(numeric_exp, Q_numeric_exp)

Returns the quantile specified by Q_numeric_exp for the population of numeric_exp.

Description

The quantile of a distribution of values is a number x_p such that a given proportion of the population values are less than or equal to x_p. For example, the 0.75 quantile (also referred to as the 75th percentile or upper quartile) of a variable is a value x_p such that 75% of the values of the variable fall below that value. The 0.5 quantile is the median.

Parameters

numeric_exp must be a number, or a numeric expression.

Q_numeric_exp (the quantile to compute) must be a floating point number strictly between 0 and 1.

Note: If the number of data points is less than QUANTILE_ESTIMATION_THRESHHOLD, then the calculation for a median or quantile will proceed as follows:

- If the data is not sorted on any columns for which calculation is requested, the data will be sorted.
- The quantile position is calculated by linear interpolation.

If the number of data points is greater than or equal to QUANTILE_ESTIMATION_THRESHHOLD, then the calculation for a median or quantile will proceed as follows:

The quantile position will be estimated by the linear time selection algorithm "Randomized-Select" as described in Introduction to Algorithms by Cormen, Leiserson, and Rivest, p. 187.

QUARTILE1(numeric_exp)

Returns the First Quartile for the population of numeric_exp.

Description

QUARTILE1 is the 0.25 quantile. It is used frequently in other calculations such as the inter-quartile range, the quartile deviation, and the quartile variation coefficient.

Parameters

numeric_exp must be a number, or a numeric expression.

QUARTILE3(numeric_exp)

Returns the Third Quartile for the population of numeric_exp.

Description

QUARTILE3 is the 0.75 quantile. It is used frequently in other calculations such as the inter-quartile range, the quartile deviation, and the quartile variation coefficient.

Parameters

numeric_exp must be a number, or a numeric expression.

RANGE(numeric_exp)

Returns the Range of the population of numeric_exp.

Description

The range is the simplest measure of the spread or dispersion of a data set. The range is the difference between the highest and lowest values in a Set. The range reflects information about extreme values but not necessarily about typical values. Only when the range is narrow (meaning that there are no outliers) does it tell us about typical values in the data. The higher the range, the greater the amount of variation in a data set.

Range = Max - Min

Where:

Max = Highest observed value in the data set

Min = Lowest observed value in the data set

Parameters

numeric_exp must be a number, or a numeric expression.

RMSERROR(numeric_exp)

Returns the RMS Error of the population of numeric_exp, where numeric_exp is a sampling of a larger population.

Description

To calculate the root mean squared error, the individual errors are squared, added together, dividing by the number of individual errors, and then the square root is taken. This gives a single number that summarizes the overall error.

$$RMS\,Error_{sample} = \sqrt{\frac{\sum_{i=1}^{n}(x-\bar{x})^2}{n-1}}$$

Parameters

numeric_exp must be a number, or a numeric expression.

RMSERRORP(numeric_exp)

Returns the RMS Error of the Population of numeric_exp.

Description

To calculate the root mean squared error, the individual errors are squared, added together, dividing by the number of individual errors, and then the square root is taken. This gives a single number that summarizes the overall error.

$$RMS\,Error_{population} = \sqrt{\frac{\sum_{i=1}^{N}(x-\bar{x})^2}{N}}$$

Parameters

numeric_exp must be a number, or a numeric expression.

SKEWNESS(numeric_exp)

Returns the Skewness of the Population of the numeric_exp, where numeric_exp is a sampling of a larger population.

Description

A set is skewed if one of its tails is longer than the other. A set has a positive skew if it has a long tail in the positive direction. A set has a negative skew if it has a long tail in the negative direction. A set is perfectly symmetrical if it has no skew. Though negatively skewed sets do occur, sets with positive skews are more common than sets with negative skews. Skew can be calculated as:

$$Skewness_{sample} = \frac{2\left(\frac{\sum_{i=1}^{n} x_i}{n}\right)^3 - 3\left(\frac{\sum_{i=1}^{n} x_i}{n}\right)\left(\frac{\sum_{i=1}^{n} x_i^2}{n}\right) + \left(\frac{\sum_{i=1}^{n} x_i^3}{n}\right)}{s^3}$$

As a general rule, the mean is larger than the median in positively skewed sets and less than the median in negatively skewed sets. The standard deviation is not a good measure of spread in highly skewed distributions and should be augmented in those cases by the semi-interquartile range. Skew is sometimes called the third moment of a set. A set with a skew of zero is one that is not lopsided in either direction. A set with a skewness of 1 or more is highly skewed. A set with a skewness between 0 and 1/2 is considered moderately skewed. If the skewness is less than 1/2 then the distribution is fairly symmetrical.

Parameters

numeric_exp should be a numeric column or a numeric expression.

SKEWNESSP(numeric_exp)

Returns the Skewness of the Population of numeric_exp.

Description

A set is skewed if one of its tails is longer than the other. A set has a positive skew if it has a long tail in the positive direction. A set has a negative skew if it has a long tail in the negative direction. A set is perfectly symmetrical if it has no skew. Though negatively skewed sets do occur, sets with positive skews are more common than sets with negative skews. Skew can be calculated as:

$$Skewness_{population} = \frac{2\left(\frac{\sum_{i=1}^{N} x_i}{N}\right)^3 - 3\left(\frac{\sum_{i=1}^{N} x_i}{N}\right)\left(\frac{\sum_{i=1}^{N} x_i^2}{N}\right) + \left(\frac{\sum_{i=1}^{N} x_i^3}{N}\right)}{\sigma^3}$$

As a general rule, the mean is larger than the median in positively skewed sets and less than the median in negatively skewed sets. The standard deviation is not a good measure of spread in highly skewed distributions and should be augmented in those cases by the semi-interquartile range. Skew is sometimes called the third moment of a set. A set with a skew of zero is one that is not lopsided in either direction. A set with a skewness of 1 or more is highly skewed. A set with a skewness between 0 and 1/2 is considered moderately skewed. If the skewness is less than 1/2 then the distribution is fairly symmetrical.

Parameters

numeric_exp should be a numeric column or a numeric expression.

SORTFIRST(value_exp)

Returns the First Item in a Sorted Population of value_exp.

Description

This function returns the 1st value of the sorted population.

Example:

V_1	V_2	V_3	V_4	V_5	V_6
5	25	2	25	25	30

The above example would return 2, because it is the 1st value of the sorted population.

Parameters

value_exp can be an expression of any SQL data type.

SORTLAST(value_exp)

Returns the Last Item in the Sorted Population of value_exp.

Description

This function returns the nth value of the sorted population of n elements.

Example:

V_1	V_2	V_3	V_4	V_5	V_6
5	25	30	25	25	2

The above example would return 30, because it is the 6th value in the sorted population.

Parameters

value_exp can be an expression of any SQL data type.

SORTMIDDLE(value_exp)

Returns the Middle Item in the Sorted Population of value_exp.

Description

This function returns the middle value of the sorted population of n elements.

SortMiddle will return the (n+1)/2'th value, performing integer division.

Example

V_1	V_2	V_3	V_4	V_5	V_6
5	25	2	1	0	30

The above example would return 2, because it is the 3th value in the sorted population ((6+1)/2 = 3.5 ~=3).

V_1	V_2	V_3	V_4	V_5
5	25	2	25	25

The above example would return 25, because it is the 3rd value in the population ((5+1)/2 = 3).

Parameters

value_exp can be an expression of any SQL data type.

STDDEV(numeric_exp)

Returns the Standard Deviation of the Population of numeric_exp, where numeric_exp is a sampling of a larger population.

Description

The standard deviation statistical function is used to depict dispersion among the measures of a given population.

To find the standard deviation of a population, the variance must first be defined, since the square root of the variance equals the standard deviation of the population.

To calculate the standard deviation of a sample of a population, you must first calculate the variance of the sample.

$$Stddev_{sample} = \sqrt{Variance}_{sample} = \sqrt{\frac{n\sum_{i-1}^{n} x_i^2 - \left(\sum_{i-1}^{n} x_i\right)^2}{n(n-1)}}$$

Parameters

numeric_exp must be a number, or a numeric expression.

STDDEVP(numeric_exp)

Returns the Standard Deviation of the Population of numeric_exp.

Description

The standard deviation statistical function is used to depict dispersion among the measures of a given population.

To find the standard deviation of a population, the variance must first be defined, since the square root of the variance equals the standard deviation of the population.

$$Stddev_{population} = \sqrt{Variance} = \frac{\sqrt{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2}}{N}$$

Parameters

numeric_exp must be a number, or a numeric expression.

TRIMEAN(numeric_exp)

Returns the Common Trimean of the Population of numeric_exp.

Description

The common method for calculation of the trimean is adding the 25th percentile plus twice the 50th percentile plus the 75th percentile and dividing the sum by four. The trimean is almost as resistant to extreme scores as the median and does not wobble as much from sample to sample as does the average in a skewed distribution. The trimean is a good measure of central tendency. The trimean requires more information than the median because it includes the upper and lower quartiles.

trimean = (quartile1 + 2*median + quartile3) / 4

Tukey's method (which used upper and lower hinge instead of the 25th percentile and the 75th percentile) produces a very similar answer. The lower hinge is the median of the lower half of the data up to and including the median. The upper hinge is the median of the upper half of the data up to and including the median. The hinges are the same as the quartiles unless the remainder when dividing the sample size by four is three.

Parameters

numeric_exp must be a number, or a numeric expression.

VARIANCE(numeric_exp)

Returns the Variance of the population of numeric_exp, where numeric_exp is a sampling of a larger population.

Description

The variance is used to depict the dispersion among the measures of a given population. To find the variance, you must first find the mean of the scores, find the measurement by which each score differs from the mean, find the square root of the difference, then divide the number by (n-1).

In a list of 'n' numbers, the Variance is the sum of the square of the differences between each number and the mean of the sample of the population.

$$Variance_{sample} = s^2 = \frac{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2}{n(n-1)}$$

Parameters

numeric_exp must be a number, or a numeric expression.

VARIANCEP(numeric_exp)

Returns the Variance of the Population of numeric_exp.

Description

The variance is used to depict the dispersion among the measures of a given population. To find the variance, you must first find the mean of the scores, find the measurement by which each score differs from the mean, then find the square root of the difference.

In a list of 'n' numbers, the variance is the sum of the square of the differences between each number and the mean of the population.

$$Variance_{population} = \sigma^2 = \frac{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2}{N^2}$$

Parameters

numeric_exp must be a number, or a numeric expression.

SQL Relational Operators

ALL

Retrieves all rows. ALL is the default for a SELECT statement. Can be used with operators <, >, = to collect specific rows.

Example:

```
Select * from employees where customer_id >= all (select customer_id from
customers where customer_id = 50005800 or customer_id = 50005600)
```

AND

Retrieves rows that contain all the same values in the specified data. Can be used with operators <, >, = to collect specific rows.

Example:

```
SELECT * FROM Customers_dataflex WHERE customercountry = 'USA' AND
   customerstate = 'WA'
```

ANY

Retrieves any rows. Can be used with operators <, >, and = to collect specific rows.

Example:

```
Select * from employees where customer_id <= any (select customer_id from
customers where customer_id = 50005800 or customer_id = 50005600)
```

BETWEEN

Finds values for *fieldname* in the range of *minimumvalue* to *maximumvalue,* inclusive. Equivalent to *fieldname* >= *minimumvalue* and *fieldname* <= *maximumvalue*. Using the BETWEEN operator results in better overall performance than using >= or <=.

Example:

```
SELECT productid FROM products WHERE productid BETWEEN 1 and 100
```

= (equal)

Retrieves rows that contain the same values as the specified data.

Example:

```
SELECT * FROM Customers_dataflex WHERE customername = 'Always Open
   Quick Mart'
```

> (greater than)

Retrieves rows whose values are greater than the specified data.

Example:

```
SELECT * FROM Customer_dataflex WHERE recordnumber > 33
```

>= (greater than or equal)

Retrieves rows whose values are greater than or equal to the specified data.

Example:

```
SELECT * FROM Customer_dataflex WHERE recordnumber >= 10
```

IN

Finds values for *fieldname* that match the list of values inside the parenthesis. This is equivalent to using the OR operator (such as *fieldname* = *value1* OR *fieldname* = *value2* OR ...). Using the IN() operator results in better overall performance.

Example:

```
SELECT productid FROM products WHERE productid IN(1,3,5,7,11)
```

IS NOT NULL

Retrieves rows where information is not null or does exist.

Example:

```
Select customer_address from customers where customer_address is not null
```

IS NULL

Retrieves rows where information is null or does not exist.

Example:

```
Select customer_address from customers where customer_address is null
```

< (less than)

Retrieves rows whose values are less than the specified data.

Example:

```
SELECT * FROM Customer_dataflex WHERE recordnumber < 10
```

<= (less than or equal)

Retrieves rows whose values are less than or equal to the specified data.

Example:

```
SELECT * FROM Customer_dataflex WHERE recordnumber <= 10
```

LIKE

The SQL primary command LIKE, also known as the LIKE operator, can be used for wildcard comparison against String values to filter specific data. The LIKE operator has three special characters, each with a different usage: the percent sign (%), the underscore (_), and the caret (^).

% multi-character wildcard

The use of the percent sign (%) in a LIKE clause retrieves matches of zero or more characters.

Example:

If the column TEST contains "ABCDEF" the expression TEST LIKE "%F" returns TRUE, the expression TEST LIKE "%C%" returns TRUE, and the expression TEST LIKE "%G" returns FALSE.

_ single-character wildcard

The use of an underscore (_) in a LIKE clause retrieves matches of a single character.

Example:

If the column TEST contains "GHIJKLM" the expression TEST LIKE "G_IJKLM" returns TRUE, and the expression TEST LIKE "A_IJKLM" returns FALSE.

^ literal character identifier

The use of the caret (^) in a LIKE cause permits the use of any of the wildcard characters %, _ or ^. If the ^ character is placed in front of either of the other two wildcard characters, the characters following are treated as normal literals.

Example:

If the column TEST contains "10% of the data" the expression TEST LIKE "10^% of the data" returns TRUE, the expression TEST LIKE "10%data" returns TRUE, and the expression TEST LIKE "10data" returns FALSE.

Important: The Microsoft Jet Engine for Microsoft Access uses an asterisk (*) as the wildcard character instead of the percent sign (%). Replace the percent sign (%) wildcard with an asterisk (*) when using Microsoft Access, or when using other DAO-compliant applications.

NOT

Retrieves rows that contain different values from the specified data. Can be used with operators <, >, = to collect specific rows.

Example:

```
SELECT * FROM Customers_dataflex WHERE customerstate is NOT = 'UT'
```

<> (not equal)

Retrieves rows that contain different values from the specified data.

Example:

```
SELECT * FROM Customers_dataflex WHERE customername <> 'Always Open
   Quick Mart'
```

OR

Retrieves rows that contain any or all the values in the specified data. Can be used with operators <, >, = to collect specific rows.

Example:

```
SELECT * FROM Customers_dataflex WHERE customerstate = 'UT' OR
   customerID = 'anthb'
```

UNION

Using the Union relational operator enables you to join information from two or more tables that have the same structure. Duplicate rows are eliminated.

Example:

```
SELECT * FROM testtable
UNION
SELECT * FROM testtable1
```

UNION ALL

Using the Union All relational operator enables you to join information from two or more tables that have the same structure. Duplicate rows are preserved.

Example:

```
SELECT * FROM testtable
UNION ALL
SELECT * FROM testtable1
```

REVERSE FETCH

It is possible to perform a reverse fetch in DBMS by using a SQL ORDER BY statement using the DESC keyword. This feature takes advantage of the natural indexes in DBMS.

Example:

```
SELECT * FROM dbms_table
ORDER BY cnx_dbms_table desc
```

Chapter 12 - 64bit Support

Native 64bit Application Support

CONNX supports 64bit clients for the windows platform. 64bit applications have the advantage of a large virtual process area, which enables completely new ways of problem solving. Under windows, 32bit applications have a maximum virtual process area of 4GB. The 32bit windows operating system reserves 2GB for private use, leaving a usable areas of 2G for application code and data. The 64bit windows operating system provides a 16 TB (Terabyte) virtual process area, where 8 TB is reserved for the operating system, and 8TB is usable for application code and data.

Architectural component	64-bit Windows	32-bit Windows
Total virtual memory (per process)	16 terabytes	4 gigabytes
Reserved virtual memory (per process)	8 terabytes	2 gigabytes
Usable virtual memory (per process)	8 terabytes	2 gigabytes

When CONNX is installed on a 64bit system, both the 32bit and 64bit version of relevant components are installed, allowing existing 32bit applications to continue functioning, and enabling 64bit applications by providing the relevant drivers. The following is a list of CONNX components that have both 32bit and 64bit counterparts.

CONNX component	64-bit Windows	32-bit Windows
ODBC Driver	CONNX32.DLL (in 64bit windows location)	CONNX32.DLL (in 32bit windows location)
OLEDB Provider	CNXOLEDBPROV.DLL (in 64bit windows location)	CNXOLEDBPROV.DLL (in 32bit windows location)
RPC Interface	CNXOLE32.DLL (in 64bit windows location)	CNXOLE32.DLL (in 32bit windows location)
JDBC Server Service	CNXJDBC64.EXE	CNXJDBC.EXE
CONNX Data Dictionary Administrator	CONNXCDD64.EXE	CONNXCDD32.EXE
Enterprise Server Service	CNXREMOTE64.EXE	CNXREMOTE.EXE
JDBC DSN Registry Tool	DSNREGISTRY64.EXE	DSNREGISTRY.EXE
InfoNaut	INFONAUT64.EXE	INFONAUT.EXE

CONNX has been tested and fully supports 64bit SQL Server Linked Server technology, and Oracle Heterogeneous Services.

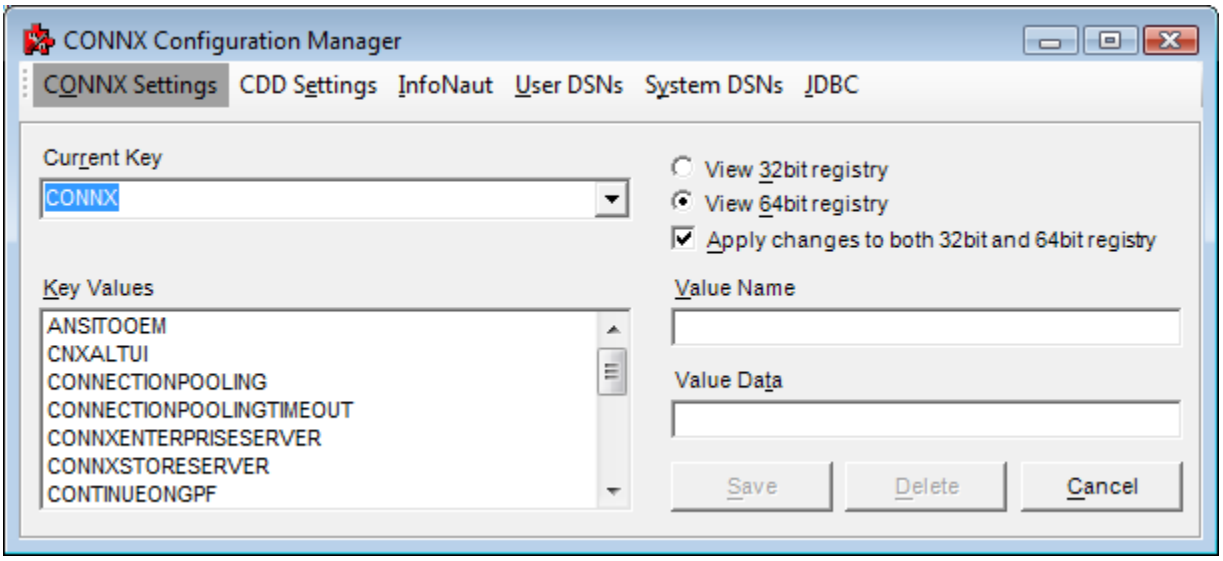
Native 64bit Data Source Support

CONNX provides native 64bit connectivity to the following:

Data Source	64bit Data Access Support
ADABAS	When ADABAS nucleus is in Unix, or Mainframe, or when the Enterprise Server Service is used.
C-ISAM	When CONNX Server component for Unix, or Mainframe, or when the Enterprise Server Service is used.
D-ISAM	When CONNX Server component for Unix, or Mainframe, or when the Enterprise Server Service is used.
DB2	Native 64bit DRDA Driver.
Codasyl DBMS	Native 64bit Access to CONNX Server on VMS.
RMS	Native 64bit Access to CONNX Server on VMS.
RDB	Native 64bit Access to CONNX Server on VMS.
Sybase	Native 64bit ODBC Driver & OLE DB Provider. (Requires 64bit Sybase Driver or Provider)
SQL Server	Native 64bit ODBC Driver & OLE DB Provider. (Requires 64bit Microsoft SQL Server Driver or Provider)
Oracle	Native 64bit OCI Driver. (Requires 64bit Oracle OCI client or Instant client)
MicroFocus and RM Cobol	When CONNX Server component for Unix, or Mainframe, or when the Enterprise Server Service is used.
Informix	Native 64bit ODBC Driver & OLE DB Provider. (Requires 64bit Informix Driver or Provider)
IMS	Native 64bit Access to CONNX Server on the mainframe.
VSAM	Native 64bit Access to CONNX Server on the mainframe.
PostgreSQL & CONNXStore	Native 64bit PostgreSQL Driver.
Any 3rd party 64bit OLE DB or ODBC Driver (CONNX Enterprise Adaptor)	Native 64bit ODBC Driver & OLE DB Provider. (Requires 3rd party 64bit driver)
Any 3rd party 32bit OLE DB or ODBC Driver (CONNX Enterprise Adaptor)	Native 64bit ODBC Driver & OLE DB Provider. (Requires used of the 32bit CONNX Enterprise Server Service)

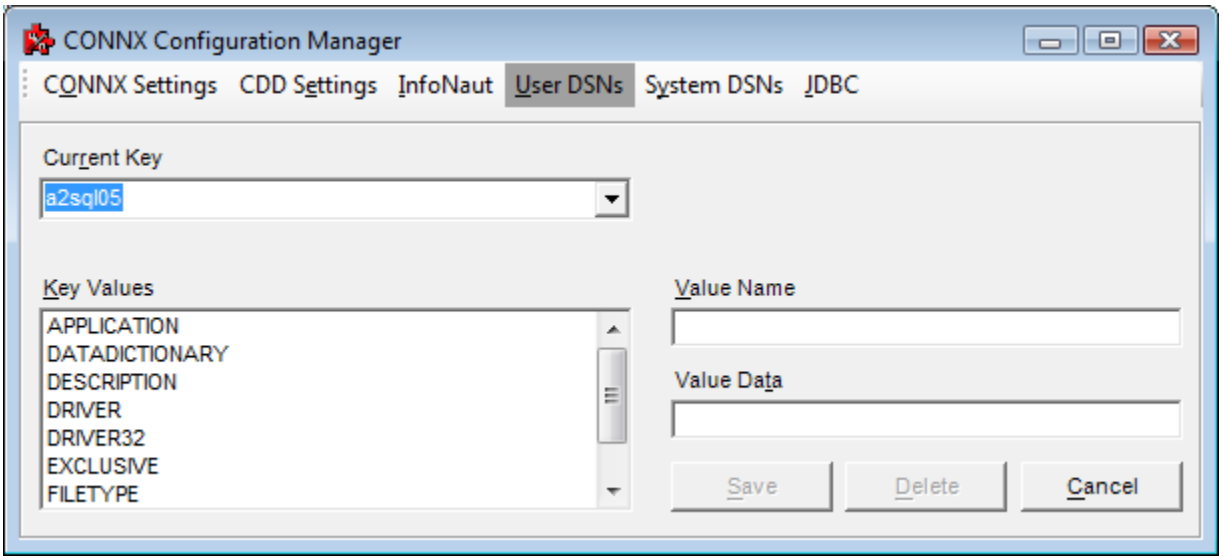
Configuring 64bit and 32bit components

On windows, CONNX is configured with the CONNX Configuration Manager. The CONNX Configuration Manager can be used for managing both the 32 bit and 64 bit components.



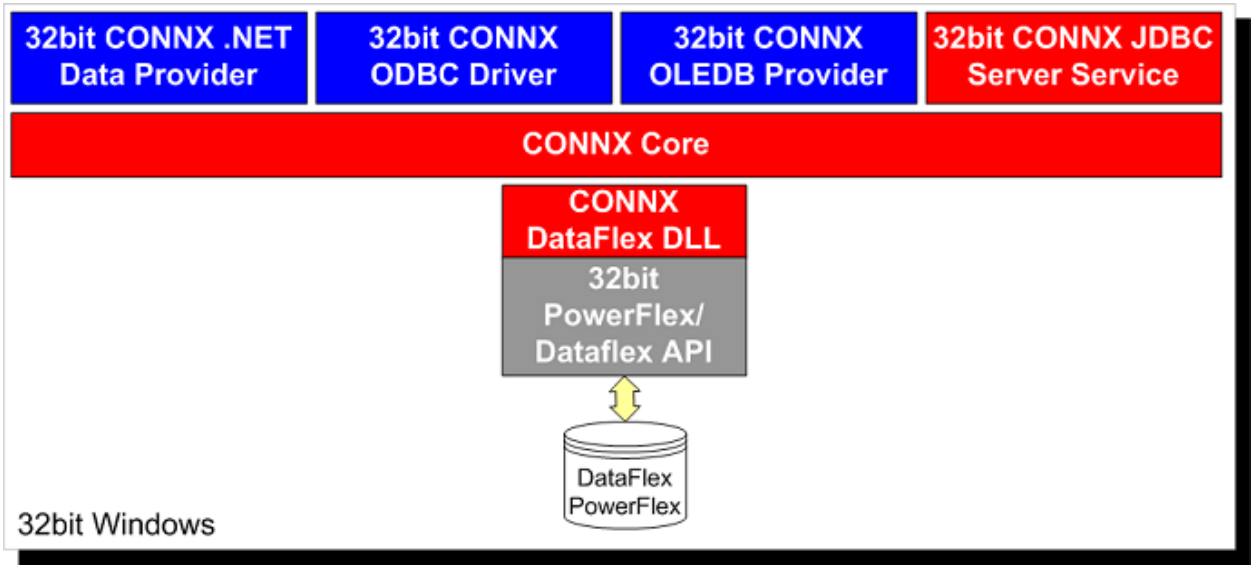
There is a new radio button to select which bit platform you wish to configure. It is likely that you will wish to keep most registry settings the same for both the 32 bit and 64 bit components. In this case, checking the "Apply changes to both 32bit and 64bit registry" checkbox will cause a setting made for one component to be made for the other as well. There are some settings, however, where it may be necessary to maintain different values for the two components. (the port Enterprise Server Service listens on is an example) In this case, this check box needs to be unchecked when changing the value.

On the Infonaut tab as well as the User DSNs tab, the settings are not differentiated between 32 bit and 64 bit. In this case, the selection radio buttons are not displayed and any settings that are made automatically apply to both.

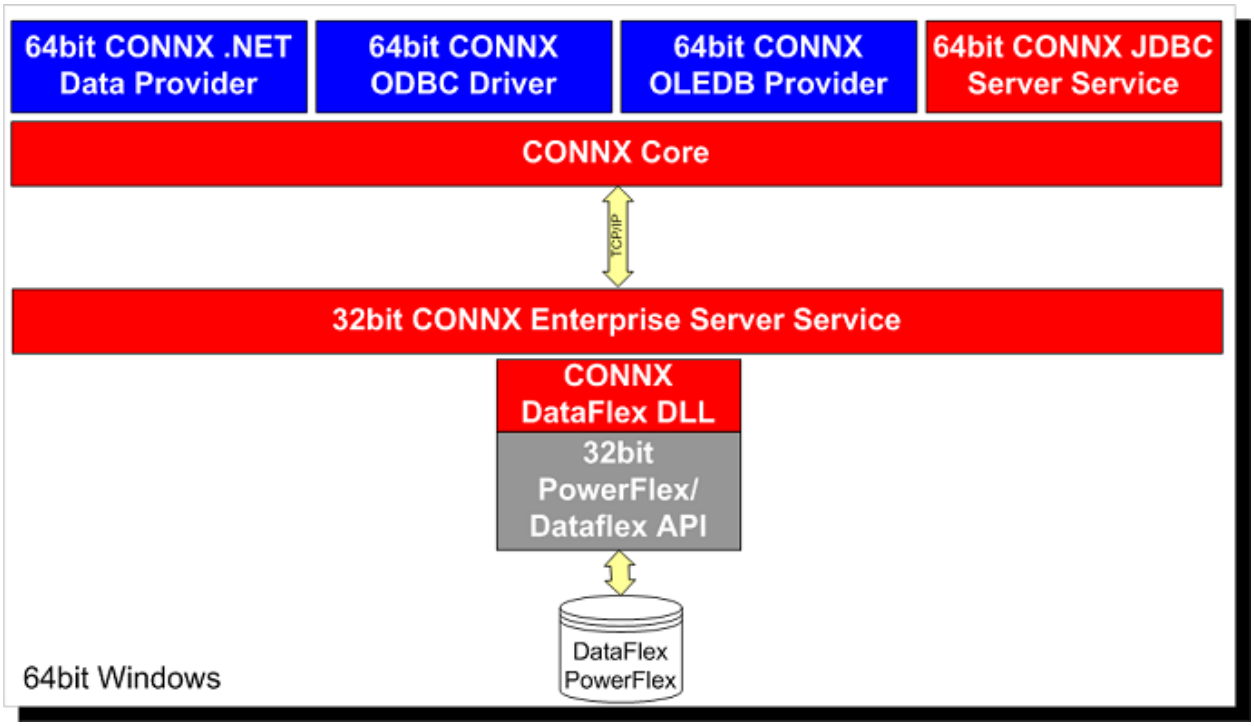


64bit to 32bit ODBC and OLE DB Bridge

The CONNX Enterprise Server Service is a windows component that provides the same "CONNX Server" component architecture that already exists on non-windows platforms. To illustrate how this component works, we will use Dataflex as an example. Currently for windows, only a 32bit Dataflex API exists. Here is a diagram of the CONNX Architecture for DataFlex.



CONNX Enables 64bit applications access to this currently 32bit only data source through a Windows Service component called the CONNX Enterprise Server Service. Using this service, the CONNX Distributed architecture allows the client to be 64bit, and the server to remain 32bit, as shown below. The 64bit CONNX Client uses TCP/IP to communicate to the 32bit Windows service.



This unique architecture enables 64bit applications to communicate to data sources which may only have 32bit drivers, and this feature is unique to CONNX.

Location of DLLs on 64bit windows

In an effort to ease the transition from 32bit to 64bit, Microsoft Windows has two different locations for storing shared 32bit and 64bit DLLs. The CONNX OLE DB Provider, and ODBC Driver are installed into the default location for shared windows DLLs. Under 32bit windows, this location is (typically) C:\WINDOWS\SYSTEM32. Under 64bit windows, it is C:\WINDOWS\SYSWOW64. Also under 64bit windows, the 64bit DLL shared location is C:\WINDOWS\SYSTEM32. At first this may seem backwards, and counter intuitive. The reason Microsoft choose these locations was to provide the maximum about of backward compatibility possible.

Architectural component	64-bit Windows	32-bit Windows
Directory name for shared 32bit DLLs	C:\WINDOWS\SYSWOW64	C:\WINDOWS\SYSTEM32
Directory name for shared 64bit DLLs	C:\WINDOWS\SYSTEM32	Not Applicable

A key to making these locations work seamlessly is something called WOW64 *file redirection.* On a 64bit windows, *w*hen a 32bit process attempts to open a file in C:\WINDOWS\SYSTEM32 - it is automatically redirected to C:\WINDOWS\SYSWOW64.

On 64bit windows, the 32bit CONNX ODBC Driver and OLE DB Provider are located in C:\WINDOWS\SYSWOW64. The 64bit CONNX ODBC Driver and OLE DB Provider are located in C:\WINDOWS\SYSTEM32.

On 32bit windows, the 32bit CONNX ODBC Driver and OLE DB Provider are located in C:\WINDOWS\SYSTEM32.

Limitations of 64bit Data Access on Windows

Some database do not yet have 64bit windows APIs drivers, preventing native 64bit access from windows. The following is a list of datasources with such limitations.

NOTE - this grid only applies if the ***data being accessed resides on windows***. If the data is on another server, VMS for example, the 64bit Client component will use TCP/IP to communicate to the server component on another platform, and results in no limitations.

In other words, if the data being access is on a non-windows system, or the enterprise server service is being used on windows, there is no 64bit limitation.

Data Source	64bit Data Access Limitation
ADABAS	ADABAS not currently available on 64bit Windows. CONNX can provide access to these data sources from 64bit applications through the 32bit CONNX Enterprise Server Service. See 64bit to 32bit ODBC and OLE DB Bridge.
DataFlex/PowerFlex	64bit windows Dataflex/Powerflex API not available. CONNX can provide access to these data sources from 64bit applications through the 32bit CONNX Enterprise Server Service. See 64bit to 32bit ODBC and OLE DB Bridge.
C-ISAM	64bit windows C-ISAM API not available. CONNX can provide access to these data sources from 64bit applications through the 32bit CONNX Enterprise Server Service. See 64bit to 32bit ODBC and OLE DB Bridge.
D-ISAM	64bit windows D-ISAM API not available. CONNX can provide access to these data sources from 64bit applications through the 32bit CONNX Enterprise Server Service. See 64bit to 32bit ODBC and OLE DB Bridge.
Microsoft Access & Excel & Word	64bit ODBC & OLE DB Providers for these data sources not available. CONNX can provide access to these data sources from 64bit applications through the 32bit CONNX Enterprise Server Service. See 64bit to 32bit ODBC and OLE DB Bridge.
Microfocus & RM Cobol	64bit windows server component not available. CONNX can provide access to these data sources from 64bit applications through the 32bit CONNX Enterprise Server Service. See 64bit to 32bit ODBC and OLE DB Bridge.

Enterprise Server Service Component

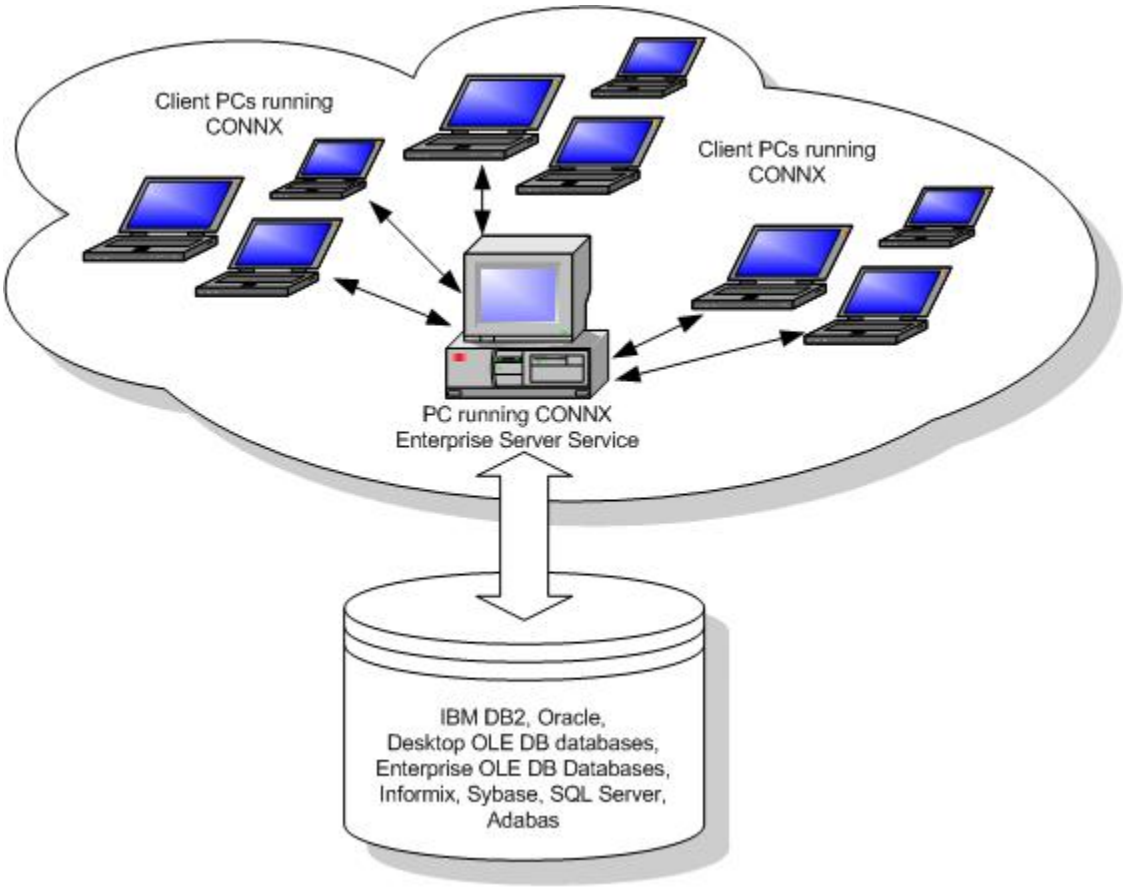
The Enterprise Server Service is essentially the "CONNX Server/Listener" component that normally resides on a non-windows platform, wrapped into a windows service. If the data source being accessed resides on windows, normally the CONNX driver will directly load the DLLs necessary to establish a direct connection to the data source, eliminating the need for the Server/Listener architecture that exists for non-windows datasources like VSAM, Adabas, and RMS for example.

However, there are sometimes advantages of having the "split stack" architecture, even if all components are on windows. Some of the advantages include:

- The ability to access 32bit only data sources from a 64bit client. See 64bit to 32bit ODBC and OLE DB Bridge.
- 3rd party connectivity software (such as the Oracle OCI Client, or the SQL Server Driver) only needs to be installed in a single central location, instead of being installed on every desktop where CONNX resides. In large organizations, this can significantly reduce configuration headaches of trying to keep everyone's tnsnames.ora in sync for example.
- In the case of Adabas, if Entire Network is required to access a version of database that does not have a Native CONNX Data Server, Entire Network only needs to be installed on a single central server instead of every desktop.

To enable the CONNX Enterprise Server Service

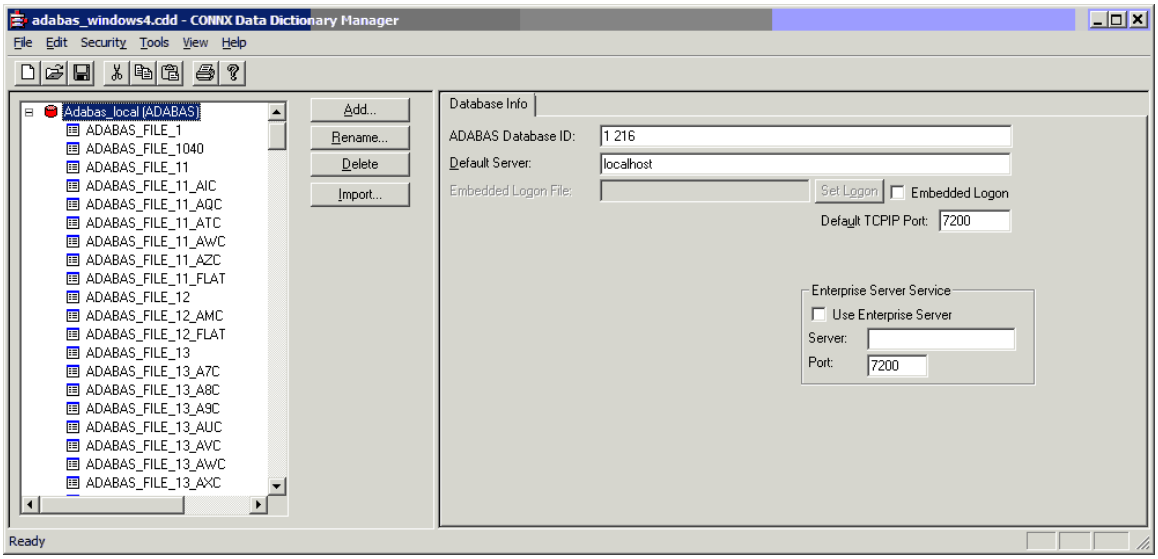
The CONNX Enterprise Server Service technology simplifies the deployment of CONNX in both small and large organizations. In cases where CONNX uses the vendor's native database driver to access data (for example, in Oracle or SQL Server databases), configuration and setup of those drivers takes place on a single middle-tier server, instead of on each client PC. With the Enterprise Server Service, CONNX seamlessly provides a single client-install solution for data access needs throughout the enterprise.



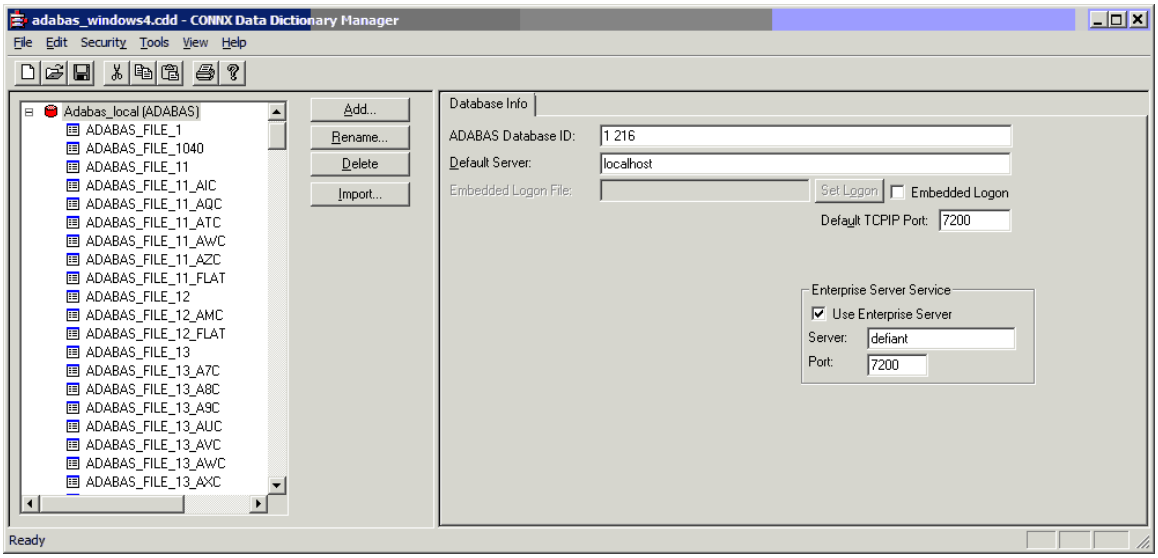
CONNX Enterprise Server Service

Note: *To run the CONNX Enterprise Server Service, you must first have selected the Enterprise Server Service check box in the Select Components window during installation of CONNX.*

1. In the CONNX Data Dictionary Manager window, select a database from the list in the upper pane.
2. The **Database Info** tab appears.



3. Check the **Use Enterprise Server** check box in the **Enterprise Server Service** group box. Enter the name or IP Address of the computer running the CONNX Enterprise Server Service in the **Server** text box. Specify the port number for the server you intend to connect to in the **Port** text box for the CONNX Enterprise Server Service (The default port number for the client to connect to the server is 6500. See CONNX Registry File Settings for information on how to change the server port setting using the CNXRUNPORT variable.).



By selecting the Use Enterprise Server option for a given database, CONNX performs all database access through that remote server. For example, if using CONNX to access SQL Server and Oracle, by selecting Use Remote Server, the SQL Server OLE DB Provider and Oracle Net* Client only need installation on the single PC running the CONNX Enterprise Server Service. This greatly simplifies deployment of CONNX in very large organizations. All other PCs in the organization require the CONNX driver only. Installation of a third-party driver or other network components is not required.

Chapter 14 - CONNX Configuration Settings

What are Configuration Settings?

CONNX uses configuration settings (also called environmental variables and registry settings) to optimize performance and enable addition functionality (like shared connections).

All configuration settings have default values. Some may be modified during installation. Others are changed as needed.

All settings are read once at either listener or server startup.

It always requires either a restart of the listener or connection of the new server for a new setting to take effect.

Note: You usually do not have to change any of the configuration settings. Most settings should be left alone.

Different environments have different was of modifying configuration settings:

Windows	**Mainframe CICS**	**Mainframe Started Task**	**Unix (includes environmental variables)**
Set through Configuration Manager	Set through the NX01 transaction.	Set through member CNXPARMS in the HLQ.CNTL directory.	Set through SQLREGISTRY

Note: For server settings, use the method (listed above) based on the location of Adabas. For example, if the database is on a Windows machine, then server settings are set with the Configuration Manager, if the database is on the Mainframe then server settings are either set in CNXPARMS or with the NX01 transaction (based on if the listener is a Started Task or CICS tranaction), etc.

- **Mainframe**
  - **CICS**
    Mainframe CICS users use the NX01 transaction to change the configuration settings.
  - **Started Task**
    Mainframe started task users use a text file (member) called CNXPARMS in the HLQ.CNTL directory (where HLQ is the higher level qualifier specified in the server component installation) to change the configuration settings.
- **Unix**

  Unix users use the DB file in the CONNX install directory connxfile.db to change the configuration settings. To modify connxfile.db use the SQLREGISTRY program.
- **Windows**

  Windows users use the CONNX Configuration Manager to change the configuration settings.

Alphabetical links to Configuration Settings

Links to all the CONNX Configuration settings topics, in alphabetical order, can be found here.

Configuration Settings beginning with Numbers or Symbols

2DigitYears

"A" Configuration Settings

ADA_DEBUG_TRACE_MASK	ADA_NATURALBYTEASBIT	ADA_TABLENAME	ALLOWMIXEDCASEPASSWORDS
ADA_FIELDNULLASZERO	ADA_NOQUALIFYBINARY	ADA_WAITTIME	ALLOWMIXEDPWD
ADA_ISNNAME	ADA_RESPECT_ISN_ON_INSERT	ADA_WFIELDASBYTES	ALLOWNULLINCHAR
ADA_LOCKDONTWAIT	ADA_SECURITY	ALLOWDATATYPECHANGES	ASYNCACCESS

"B" Configuration Settings

BASE1Index

"C" Configuration Settings

CaseSensitive	CNXLocalIP	CNXPOSTSERVERNAME	COMMITCOUNT
CHARASVARCHAR	CNX_LIBRARY_PATH	CNXRUNPORT	ConnectionPooling
Client_Locale	CNXLISTENER	CNXSELECT	ConnectionPoolingTimeout
CNXBarnard	CNXMUALPHA	CNXSOCKETTIMEOUT	ConnectPort
CNXBATCHBUFFER	CNXNoPreAuthorize	CNXTCPIPBUFFER	ConnectReturnPort
CNXConnectBack	CNXNOPOST	CNXTRUEUCX	ConnectTimeout
CNXDECNETTASK	CNXNOQIO	CNXUSEMBX	ConnectTries
CNXDir	CNX_NO_TIMER	COBOL_COMP_BYTESTEP	CONNXREG_DISPLAY_0PTS
CNXHash	CNXOLDLICENSE	CollatingSequence	
CNXKBAUTOHORI	CNX_PASS_TICKE	Compress	

ZE	TS	

"D" Configuration Settings

DataFlexMode	DebugLevel	DefaultCOBOLDEC	DefaultVMSCreatePath
DataFlexStructureUpdate	DebugLoc	DefaultCOBOLInt	
DB_Locale	DebugVerbose	DefaultOnDemand	
Debug		DefaultPort	

"E" Configuration Settings

EmptyString	ENCRYPT	EntireNetwork	EntireNetworkMultiFetchFixed
ESQNULL			

"F" Configuration Settings

FastMemorySize	ForceADANUKey	FreeformCOBOLFD
FilterDataTypes	ForceTransactions	FujitsuCOBOLFD

No "G" Configuration Settings

"H" Configuration Settings

Hash	Honor DBIDFileID	HuffmanPowerFlex

"I" Configuration Settings

ImportFlatOnly	ImportOverwrite	ImportProcedures	InformixDir
ImportGroupNames	ImportPrefix	ImportViews	IntegratedSecurityMode

"J" Configuration Settings

JoinCacheMultiplier	JoinCacheSize	JoinCount

"K" Configuration Settings

KeepHyphens

"L" Configuration Settings

LogLoc
LowerCaseOnly

"M" Configuration Settings

MapFileIncrementSize	MaxRowsFetched	MisMatchOn	MultiFetch
MaxRowsCompared	MaxRowsFetchedFail	MixedCaseGrammar	MUPESupport

"N" Configuration Settings

NLS_Lang	NoProcedures	NoWait
NoLogo	NoViews	NumberConvert

"O" Configuration Settings

OLEInit	OPMaxHold	OPSearchTime	OracleBulkModeDisabledFlag
OPDontCall	OPMaxISN	OPTimeOut	Oracle_Home
OPMaxCmdID	OPReadOnly	OptimizeCountStar	

"P" Configuration Settings

PacketSize	PerformRangeChecks	PrecisionOverride
PeekMessage	Port	

"Q" Configuration Settings

QuadwordDouble

"R" Configuration Settings

ReadTimeout	REIssueOP	ReplacePrefix	RMSPaddedNoNulls

"S" Configuration Settings

SBCCSID	SetOpen	SocketPause	SuperDescriptorAsField
SCTLogical	ShareConectionCount	StrictConversionRules	
SetEnabled	ShareConections	SubQueryRowEstimate	

"T" Configuration Settings

TableCache	TCPSize	TEXTDBMS	TwoPhaseCommit
TCPIPDebug	TempPath	Timeout	

"U" Configuration Settings

UpperCaseOnly	UseCONNXSchemaForNative	UseOldAdabasDecimalDataType
Use32KRecordBuffer	UseDoubleForNumber	UseSXCall

No "V" "W" "X" Configuration Settings

"Y" Configuration Settings

YearWindow

No "Z" Configuration Settings

2DIGITYEARS

2DIGITYEARS allows DataFlex dates to be interpreted as either two or four digit years .

2DIGITYEARS = 1

If 2DIGITYEARS = 1 and the year is greater than or equal to 1900, DataFlex uses the last two digit of the year.

Example:

If 2DIGITYEARS = 1 and year = 1955, DataFlex uses 55 as the year

If 2DIGITYEARS = 1 and year = 2010, DataFlex uses 10 as the year

If 2DigitYears = 0, DataFlex uses the all four digit of the year.

Example:

If 2DIGITYEARS = 0 and year = 1955, DataFlex uses 1955 as the year

If 2DIGITYEARS = 0 and year = 2010, DataFlex uses 2010 as the year

Default = 1.

Warning: If the DataFlex date is earlier than 1900, set 2DigitYears = 0. Otherwise there will be unpredictable date results.

Environments: Client,Windows, Linux. DataFlex

Configuration Manager: CONNX Settings; Current Key = CONNX\Dataflex; Key Value/Value Name = 2DIGITYEARS

Unix SQL Registry: CONNX.DATAFLEX.2DIGITYEARS

ADA_DEBUG_TRACE_MASK

Specifies which Adabas information is displayed in the debug trace output..

```
ADA_DEBUG_TRACE_MASK = 1
```

1 = Display the hexadecimal dump of the control buffer after each Adabas call.

2 = Display the hexadecimal dump of the format buffer for each Adabas call.

4 = Display the hexadecimal dump of the record buffer for each Adabas call.

8 = Display the hexadecimal dump of the search buffer for each Adabas call.

16 = Display the hexadecimal dump of the value buffer for each Adabas call.

32 = Display the hexadecimal dump of the ISN buffer for each Adabas call.

64 = Display the hexadecimal dump of the control buffer prior to each Adabas call.

128 = Display the hexadecimal dump of the control buffer prior to each Adabas call - only if there is an error. (**Note:** The value of DEBUG must be set to 0 when using this setting)

Caution: Enabling one or more of these debug bitmap trace settings can produce voluminous output. Only do so if requested by tech support.

Examples:

ADA_DEBUG_TRACE_MASK = 1

Display only the hexadecimal dump of the control buffer after each Adabas call.

ADA_DEBUG_TRACE_MASK = 3

Display the hexadecimal dumps of the control and format buffers after each Adabas call.

ADA_DEBUG_TRACE_MASK = 7

Display the hexadecimal dumps of the control, format, and record buffers after each Adabas call.

ADA_DEBUG_TRACE_MASK = 15

Display the hexadecimal dumps of the control, format, record, and search buffers after each Adabas call.

ADA_DEBUG_TRACE_MASK = 31

Display the hexadecimal dumps of the control, format, record, search, and value buffers after each Adabas call.

ADA_DEBUG_TRACE_MASK = 63

Display the hexadecimal dumps of the control, format, and record buffers after each Adabas call.

ADA_DEBUG_TRACE_MASK = 127

Display the hexadecimal dumps the control, format, record, search, value, and ISN buffers before and after each Adabas call.

Default = 127

Environments: Client, Server, Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = ADA_DEBUG_TRACE_MASK

Unix Environment Variable: ADA_DEBUG_TRACE_MASK

Started Task: ADA_DEBUG_TRACE_MASK

CICS: ADA_DEBUG_TRACE_MASK

ADA_FIELDNULLASZERO

ADA_FIELDNULLASZERO = 1

Enables Alpha fields filled with binary 0's, to be treated as a SQL Null field. Natural programs sometimes will fill alpha fields with binary 0's if they are unused. If the table is to be replicated with Open Systems Event Replicator, or using SQL ("insert/select"), this will cause binary 0 filled text fields to be converted to a space padded null text fields for the target table. This potentially means that records might not be available using a superdescriptor, if there was a case where a zero filled field turned into a space padded field. This is due to Adabas not being able to access a record if any of the superdescriptor fields are blank. Therefore **For Replication customers**, it is advised to set this field to 0.

Default = 1.

Environments: Client, Server, Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = ADA_FIELDNULLASZERO

ADA_ISNNAME

Enables the old naming convention of ISN_<DBID>_<FILEID> instead of ISN_<TableName>.

ADA_ISNNAME=0

0 = Use ISN_<DBID>_<FILEID> (where DBID is the Adabas database number and FILEID is the Adabas file number)

1 = Use ISN_<TableName> (where TableName is the Logical Table Name)

Default = 1.

Environments: Client, Windows, Adabas, CDD

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = ADA_ISNNAME

ADA_LOCKDONTWAIT

To modify the outcome when a query tries to issue a Hold Lock on a locked record, set the following value:

```
ADA_LOCKDONTWAIT = 1
```

If ADA_LOCKDONTWAIT = 0, the query will wait until the record is released.

If ADA_LOCKDONTWAIT = 1, the query will generate a 145 nucleus error.

Default = 0.

Environments: Client, Server, Windows, Mainframe, Unix, Adabas
Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = ADA_LOCKDONTWAIT

Unix Environment Variable: ADA_LOCKDONTWAIT
Started Task: ADA_LOCKDONTWAIT
CICS: ADA_LOCKDONTWAIT

ADA_NATURALBYTEASBIT

If you will be importing Natural Logicals, and desire that the logical fields be treated as a Bit field instead of Byte Field, set the following value:

```
ADA_NATURALBYTEASBIT = 1
```

If this value is set, CONNX will treat Natural DDM fields that are logical as Bit fields instead of Byte Fields. This is only used by the Import Tool when using Natural Imports, and thus only needs to be set on Windows.

Default = 0.

Environments: Windows, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = ADA_NATURALBYTEASBIT

ADA_NOQUALIFYBINARY

If you will be manually switching a data type (such as alphanumeric) to binary, set the following value:

```
ADA_NOQUALIFYBINARY = 1
```

If this value is set, CONNX will not do extra qualification on Binary Fields. If it is not set (value of 0), CONNX may (depending on data type) fully qualify the format buffer for any given field.

Note: We do not recommend manually switching data types.

Note2: This value is ignored when using Entire Net-work. In this case, binary fields are always fully qualified.

Default = 1.

Started Task: ADA_NOQUALIFYBINARY

CICS: ADA_NOQUALIFYBINARY

Environments: Server, Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = ADA_NOQUALIFYBINARY

Unix Environment Variable: ADA_NOQUALIFYBINARY

ADA_RESPECT_ISN_ON_INSERT

For Adabas, if this setting is enabled, if an ISN is supplied during an insert statement, the ISN will be treated as a "user isn" and passed to adabas, otherwise an auto-assigned ISN will be requested from adabas.

ADA_RESPECT_ISN_ON_INSERT = 1

Default = 1.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = ADA_RESPECT_ISN_ON_INSERT

Unix Environment Variable: ADA_RESPECT_ISN_ON_INSERT

Started Task: ADA_RESPECT_ISN_ON_INSERT

CICS: ADA_RESPECT_ISN_ON_INSERT

ADA_SECURITY

For Adabas if this setting is enabled, the user id and password defined in the CDD will be passed to Adabas for authentication. If Adabas is also configured for SSX authentication, the user id and password will be authenticated by the database. If either the user id or password is incorrect, the Adabas nucleus will return nucleus response code 200. If the database is configured for SSX authentication and ADA_SECURITY is not set to one in the CONNX server, the Adabas nucleus will return a response code 200 regardless of the user id and password stored in the CDD.

ADA_SECURITY = 1

Default = 0.

Note: This setting applies to LUW only

Environments: Server,Windows, Linux, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = ADA_SECURITY

Unix Environment Variable: ADA_SECURITY

ADA_TABLENAME

Specify Root Tables and Flattened Tables extensions.

ADA_TABLENAME=1

0 = Put a _ROOT extension on the Root table and do not put any extension on the Flat table.
1 = Do not put any extensions on any of the Tables. Do not store the Flat table in the CDD.
2 = Put a _FLAT extension on the Flat table and do not put any extension on the ROOT Table.

Default = 2

Environments: Client, Windows, Adabas, CDD

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = ADA_TABLENAME

ADA_WAITTIME

For Adabas, the number of seconds to wait for a locked record.

ADA_WAITTIME = 5

Default = 0.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = ADA_WAITTIME

Unix Environment Variable: ADA_WAITTIME

Started Task: ADA_WAITTIME

CICS: ADA_WAITTIME

ADA_WFIELDASBYTES

If set, this will honor the length of a field that is an ADABAS W Field. If the W field is defined to be 8 Bytes, then CONNX will register it as 8 bytes. The default is to multiply W fields by 2 to allow for more potential storage in the case they represent languages that use 2 or more bytes per character (example Chinese). This is invoked when CREATE TABLE DESCRIPTION or CREATE TABLE is done using NCHAR fields. An example would be:

```
CREATE TABLE DESCRIPTION L7000   DATABASE NUMBER 2 FILE NUMBER 55
(
   ADA_ISN                    SEQNO(0)  NOT NULL,
   AA nchar (8) SHORTNAME 'AA'
);
```

In this example, AA would be represented as 8 Bytes in the CONNX CDD if ADA_WFIELDASBYTES is set to 1, or it will be 16 bytes (Default) if the Setting is not set, or is set to 0.

```
To invoke set registry in the ADABAS section to:
ADA_WFIELDASBYTES = 1
```

If ADA_WFIELDASBYTES = 0, CONNX will represent the column size for W fields by multiplying the FDT column size by 2. (Default)

If ADA_WFIELDASBYTES = 1, CONNX will represent the column size for W fields by using the actual FDT column size.

Default = 0.

Environments: Client, Windows,Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = ADA_WFIELDASBYTES

ALLOWDATATYPECHANGES

To enable modification of data types in tables for which CONNX does not normally allow modifications, set the following value:

```
ALLOWDATATYPECHANGES=1
```

Enables the modification of data types.

Default = 0.

Environments: Client, Windows CDD

Configuration Manager: CDD Settings; Current Key = CONNXCDD; Key Value/Value Name = ALLOWDATATYPECHANGES

ALLOWMIXEDCASEPASSWORDS

To enable the use of mixed-case passwords during VMS logon procedures, set the following value:

AllowMixedCasePasswords=1

Once AllowMixedCasePasswords is enabled, any combination of uppercase and lowercase characters may be used to log on to VMS databases.

Default = 0 (disable mixed-case passwords)

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = ALLOWMIXEDCASEPASSWORDS

Unix Registry Setting: CONNX.ALLOWMIXEDCASEPASSWORDS

ALLOWMIXEDPWD

Enables the use of mixed-case passwords on zOS 1.7 and higher:

zOS 1.7 and higher allows passwords to be upper case, mixed case or mixed number. If the system is configured to use mixed case or mixed number passwords, the ALLOWMIXEDPWD confguration parameter must be set in the CONNX CNXPARMS file on the server.

To allow mixed case passwords, set:

ALLOWMIXEDPWD=1

To disable mixed-case passwords, set:

ALLOWMIXEDPWD=0

Default = 1 (allow mixed-case passwords)

Environments: Server, zOS only
Configuration: CNXPARMS Settings; ALLOWMIXEDPWD=<0,1>

ALLOWNULLINCHAR

This setting enables a binary zero to be embedded in CHAR fields. Normally zero is used as a null terminator. Enabling this setting will degrade performance.

```
ALLOWNULLINCHAR=1
```

Enables binary zero in char fields.

Default = 0.

Environments: Client, Windows CONNX

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = ALLOWNULLINCHAR

ALLOWTRAILINGSPACEINVARCHAR

This setting enables a non ANSI standard behavior to allow VARCHAR fields to contain trailing spaces.

```
ALLOWTRAILINGSPACEINVARCHAR=1
```

Enables trailing spaces in VARCHAR fields.

Default = 0.

Environments: Client, Windows CONNX

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = ALLOWTRAILINGSPACEINVARCHAR

ANSITOOEM

When not set to 0 or 2 CONNX writes a string it translates the string from the OEM-defined character set into either an ANSI or a wide-character string. On reading a string it translates it from either an ANSI or a wide-character string to OEM-defined character set.

ANSITOOEM=2

Default = 2.

Environments: Client, Windows CONNX

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = ANSITOOEM

ASYNCACCESS

Setting the ASYNCACCESS registry key forces CONNX to run queries asynchronously.

ASYNCACCESS = 1

The default is = 0, which means that CONNX runs queries synchronously. You may also remove the ASYNCACCESS value to restore synchronously run queries.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = ASYNCACCESS

Unix Registry Setting: CONNX.ASYNCACCESS

AUDIT

Enables Auditing for CONNX for ADABAS

```
AUDIT = 1
```

When auditing is enabled, any query access against one of the configured audit files will result in an audit record being generated and placed into the audit table.
If auditing is enabled the following configuration settings should also be configured:

AUDITDBID - the Adabas Database ID that contains the audit file.

AUDITFILECOUNT - the number of files to be audited

AUDITFILEID - the file number of the Audit file

AUDITPASSWORD - the ADASCR password for writing data to the audit file

AUDITFILTER### - where ### is 001 through 999 - the filter specification that will identify which adabas file to be audited, along with the optional key field.

The target audit file has two supported formats - depending on the setting AUDITFORMAT:

If AUDITFORMAT is zero (the default value) then the audit file must have the following structure:

AA - 7,P - Natural timestamp - the time of the audit event

AB - 8.A - UserID - the user identifier

AC - 4,F - The database number for the table that triggered the audit record

AD - 4,F - The file number for the table that triggered the audit record

AE - 4,F - The ISN of the record that triggered the audit.

AF - 100,A - The optional key field from the record that triggered the audit

For example, the audit file can be created using the following SQL statement:

create table AuditFile(AuditTime timestamp NOT NULL, Userid char(8) NOT NULL, databasenumber integer NOT NULL, filenumber integer NOT NULL, ISN integer NOT NULL, keyfield varchar(100) NOT NULL)

If the AUDITFORMAT is 1, then the audit file contains a periodic group and must have the following structure:

1,AA - 7,P - Natural timestamp - the time of the audit event

1,AB - 8.A - UserID - the user identifier

1,AC - 4,F - The database number for the table that triggered the audit record

1,AD - 4,F - The file number for the table that triggered the audit record

1,AE, PE - Periodic group use for audit performance

2,AF - 4,F - The ISN of the record that triggered the audit.

2,AG - 100,A - The optional key field from the record that triggered the audit

For example, the audit file can be created using the following SQL statement:

Create cluster description adalocal.auditfilenewcluster

```
FILE NUMBER <your adabas file number> (
create table AuditFileNew(
internalisn seqno(0) not null,
AuditTime timestamp shortname 'AA' NOT NULL  ,
Userid     shortname 'AB' NOT NULL,
databasenumber integer  NOT NULL  shortname 'AC',
filenumber integer  NOT NULL  shortname 'AD',
primary key (internalisn)),
create table  AuditFileNewValueData(
internalisn seqno(0) not null,
COL_SEQNO_1 SEQNO(1) NOT NULL,
ISN   shortname 'AF' NOT NULL,
keyfield varchar(100)  NOT NULL  shortname 'AG',
foreign key (internalisn) references AuditFileNew,
primary key (internalisn, COL_SEQNO_1))
)
```

AUDITFORMAT = 0 results in a single adabas record for each audit record, which is a simpler format, easier to read, but results in slower performance.

AUDITFORMAT = 1 will store multiple audit records in the periodic group element of a single adabas record. This results in much better performance, but can make the data a little more difficult to read.

Default = 0.

Environments: Server

Configuration Manager: CONNX Settings; Current Key = CONNX/ADABAS; Key Value/Value Name = AUDIT

Unix Environment Variable: AUDIT

VMS Logical: AUDIT

Started Task: AUDIT

CICS: AUDIT

AUDITDBID

Specifies the audit file databsae ID when auditing is enabled for CONNX for ADABAS

```
AUDITDBID = 12
```

Please see the AUDIT configuration setting for more details.

Default = 0.

Environments: Server

Configuration Manager: CONNX Settings; Current Key = CONNX/ADABAS; Key Value/Value Name = AUDITDBID

Unix Environment Variable: AUDITDBID

VMS Logical: AUDITDBID

Started Task: AUDITDBID

CICS: AUDITDBID

AUDITFILEID

Specifies the audit file ID when auditing is enabled for CONNX for ADABAS

```
AUDITFILEID = 12
```

Please see the AUDIT configuration setting for more details.

Default = 0.

Environments: Server

Configuration Manager: CONNX Settings; Current Key = CONNX/ADABAS; Key Value/Value Name = AUDITFILEID

Unix Environment Variable: AUDITFILEID

VMS Logical: AUDITFILEID

Started Task: AUDITFILEID

CICS: AUDITFILEID

AUDITFILECOUNT

Specifies the number of adabas audit filters (the number of files to be audited) when auditing is enabled for CONNX for ADABAS

```
AUDITFILECOUNT = 10
```

Please see the AUDIT configuration setting for more details.

Default = 0.

Environments: Server
Configuration Manager: CONNX Settings; Current Key = CONNX/ADABAS; Key Value/Value Name = AUDITFILECOUNT

Unix Environment Variable: AUDITFILECOUNT
VMS Logical: AUDITFILECOUNT
Started Task: AUDITFILECOUNT
CICS: AUDITFILECOUNT

AUDITFILTER###

Where ### is a number between 000 and 999. (Example, AUDITFILTER001, AUDITFILTER002, ...)

Specifies an Adabas file that will trigger auditing when read, and when Auditing is enabled for CONNX for Adabas.

```
AUDITFILTER001 = 1,11,AA,8,A
AUDITFILTER002 = 1,14,AC,10,A
AUDITFILTER003 = 2,31,AG,20,A
....
```

The format of the audit filter is as follow:

<database ID of file to be audited>,<file id of the file to be audited>,<key field to be placed in audit table>, <length of key field>, <data type of key field>

The key field must be converted to text 'A' Format.

Please see the AUDIT configuration setting for more details.

Default = 0.

Environments: Server

Configuration Manager: CONNX Settings; Current Key = CONNX/ADABAS; Key Value/Value Name = AUDITFILTER###

Unix Environment Variable: AUDITFILTER###

VMS Logical: AUDITFILTER###

Started Task: AUDITFILTER###

CICS: AUDITFILTER###

AUDITFORMAT

Specifies the audit file format when auditing is enabled for CONNX for ADABAS

```
AUDITFORMAT = 1
```

The target audit file has two supported formats - depending on the setting AUDITFORMAT:

If AUDITFORMAT is zero (the default value) then the audit file must have the following structure:

AA - 7,P - Natural timestamp - the time of the audit event

AB - 8.A - UserID - the user identifier

AC - 4,F - The database number for the table that triggered the audit record

AD - 4,F - The file number for the table that triggered the audit record

AE - 4,F - The ISN of the record that triggered the audit.

AF - 100,A - The optional key field from the record that triggered the audit

For example, the audit file can be created using the following SQL statement:

create table AuditFile(AuditTime timestamp NOT NULL, Userid char(8) NOT NULL, databasenumber integer NOT NULL, filenumber integer NOT NULL, ISN integer NOT NULL, keyfield varchar(100) NOT NULL)

If the AUDITFORMAT is 1, then the audit file contains a periodic group and must have the following structure:

1,AA - 7,P - Natural timestamp - the time of the audit event

1,AB - 8.A - UserID - the user identifier

1,AC - 4,F - The database number for the table that triggered the audit record

1,AD - 4,F - The file number for the table that triggered the audit record

1,AE, PE - Periodic group use for audit performance

2,AF - 4,F - The ISN of the record that triggered the audit.

2,AG - 100,A - The optional key field from the record that triggered the audit

For example, the audit file can be created using the following SQL statement:

Create cluster description adalocal.auditfilenewcluster

FILE NUMBER <your adabas file number> (

create table AuditFileNew(

internalisn seqno(0) not null,

AuditTime timestamp shortname 'AA' NOT NULL ,

Userid shortname 'AB' NOT NULL,

databasenumber integer NOT NULL shortname 'AC',

filenumber integer NOT NULL shortname 'AD',

primary key (internalisn)),

create table AuditFileNewValueData(

internalisn seqno(0) not null,

```
COL_SEQNO_1 SEQNO(1) NOT NULL,
ISN   shortname 'AF' NOT NULL,
keyfield varchar(100)  NOT NULL  shortname 'AG',
foreign key (internalisn) references AuditFileNew,
primary key (internalisn, COL_SEQNO_1))
)
```

AUDITFORMAT = 0 results in a single adabas record for each audit record, which is a simpler format, easier to read, but results in slower performance.

AUDITFORMAT = 1 will store multiple audit records in the periodic group element of a single adabas record. This results in much better performance, but can make the data a little more difficult to read.

Please see the AUDIT configuration setting for more details.

Default = 0.

Environments: Server

Configuration Manager: CONNX Settings; Current Key = CONNX/ADABAS; Key Value/Value Name = AUDITFORMAT

Unix Environment Variable: AUDITFORMAT

VMS Logical: AUDITFORMAT

Started Task: AUDITFORMAT

CICS: AUDITFORMAT

AUDITPASSWORD

Specifies the write access ADASCR password for the audit file when auditing is enabled for CONNX for ADABAS

AUDITPASSWORD= ######

Please see the AUDIT configuration setting for more details.

Default = 0.

Environments: Server
Configuration Manager: CONNX Settings; Current Key = CONNX/ADABAS; Key Value/Value Name = AUDITPASSWORD

Unix Environment Variable: AUDITPASSWORD
VMS Logical: AUDITPASSWORD
Started Task: AUDITPASSWORD
CICS: AUDITPASSWORD

AUTOBULKINSERT

This setting will cause all Insert statements to be cached and sent in batch every 1000 records (by default), controlled by BULKOPERATIONCOUNT.
When a statement handle is freed or dropped all remaining cached rows will be inserted using the bulk operation interface applicable to the database.

```
AUTOBULKINSERT=1
```

Enables transparent bulk inserts.

Default = 0.

Environments: Client, Windows CONNX

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = AUTOBULKINSERT

AUTOCONVERTFORLIKE

To enable automatic conversion of any data type into character/Unicode for comparison with SQL LIKE queries, set the following value:

```
AUTOCONVERTFORLIKE=1
```

Enables any type of non-character data (such as Numeric, Float, Integer, Timestamp, etc.) to be treated as a string for the LIKE operator.

Default = 0.

Environments: Client, Windows CDD

Configuration Manager: CDD Settings; Current Key = CONNXCDD; Key Value/Value Name = AUTOCONVERTFORLIKE

BASE1INDEX

If BASE1Index = 1, then the cnxarraycolumn or SEQNO fields will begin with 1, instead of 0 and the first occurrence of a MU, PE, or MUPE field will display 1 in the cnxarraycolumn or SEQNO field.

```
BASE1INDEX=1
```

Default = 0.

Environments: Client, Windows, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = BASE1INDEX

Unix Registry Setting: CONNX.BASE1INDEX

BULKOPERATIONCOUNT

This setting will determines the number of rows to process in batch when performing bulk operations, such as bulk insert.

```
BULKOPERATIONCOUNT=1000
```

Bulk operations row count.

Default = 1000.

Environments: Client, Windows CONNX

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = BULKOPERATIONCOUNT

CASESENSITIVE

To define whether text comparisons are case-sensitive:

```
CASESENSITIVE = 0
```

If CaseSensitive = 0, it will ignore the case during comparisons unless the query is executed in pass-through mode and the target database has a contrary behavior.

If CaseSensitive = 1, it will respect the case during comparisons unless the query is executed in pass-through mode and the target database has a contrary behavior.

Default = 0

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CASESENSITIVE

Unix Registry Setting: CONNX.CASESENSITIVE

CDDPASSWORD

This is a Data server setting which, if set, will only allow connections from CDDs when the CDD password matches the specified value. Only the first 10 characters of the password are used for a case insensitive match to allow connections. This setting is a comma separated list of passwords, enabling the support of multiple CDD passwords simultaneously. This also means that you cannot use a comma in any of the CDD passwords.

```
CDDPASSWORD = <password>[,password][,password ...]
```

Default = not set

Environments: Client, Windows, Unix, VMS

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CDDPASSWORD

Unix Registry Setting: CONNX.CDDPASSWORD

CERTPATH

Used during the Import of a Redshift database to specify the folder with the SSL Certificate. If this is not set then the folder used for the certificates is the root installation folder:

```
CERTPATH = C:\CONNX32
```

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CERTPATH

Unix Registry Setting: CONNX.CERTPATH

CHARASVARCHAR

CHARASVARCHAR = 1

When set to a value of 1, this value allows Adabas Alpha (A) fields to be imported as VARCHAR rather than CHAR. This setting only affects the import process and is only valid for Adabas SYSOBJH Transfer Format (SYSTRANS) imports.

Default = 0.

Environments: Client, Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = CHARASVARCHAR

CLIENT_LOCALE

This Unix-only language setting is for Informix access.

Please consult your Informix documentation for information regarding specific valid values for your operating system.

Environments: Client

Unix Environment Variable: CLIENT_LOCALE

CNXALTUI

This setting is used by the CONNX Installation utility and does not need to be edited. Is specifies whether the installer was the CONNX branded installer, or the Adabas SQL Gateway branded installer.

Environments: Client, Windows CONNX

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CNXALTUI

CNXBARNARD

Enables / Disables the the Barnard TCP/IP stack for the VSAM Server component on VSE/CICS.

```
CNXBARNARD = 1
```

If CNXBARNARD = 0, the Barnard TCP/IP stack is disabled.
If CNXBARNARD = 1, the Barnard TCP/IP stack is enabled.

This only applies on VSE/CICS systems.

Default = 0 disabled.

Environments: Server
CICS: CNXBARNARD

CNXBATCHBUFFER

Determines the size of buffer, in bytes, used for sending records from the data server back to the client in batch.

```
CNXBATCHBUFFER = 1000000
```

The buffer size can have a substantial impact on performance depending on the network latency between the client and the server.

Default = 65535.

Environments: Server
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CNXBATCHBUFFER

Unix Environment Variable: CNXBATCHBUFFER
VMS Logical: CNXBATCHBUFFER
Started Task: CNXBATCHBUFFER
CICS: CNXBATCHBUFFER

CNXCONNECTBACK

Enables / Disables TCP/IP socket sharing in the Listener connection logic.

```
CNXCONNECTBACK = 0
```

The client creates a TCP/IP socket and associate it with the installation port (usually 6500). The installation port will need to be open on the CONNX listener machine. After the initial connection to the listener with this socket:

- If CNXCONNECTBACK = 0, the listener hands the client-created socket to the CONNX server. This socket is used by the rest of the server-client and client-server communications until the client closes the connection. This is the optimal setting.
- If CNXCONNECTBACK = 1, the listener tells the CONNX server the client information and closes the client-created socket. The server creates a new socket and associates it with a port on the client machine. This server-created socket is used by the rest of the server-client and client-server communications until the client closes the connection.

The presence or absence of a firewall between the CONNX client and server will affect the use of CNXCONNECTBACK:

- If there is no firewall between the CONNX client and server the customer does not need to be concerned with the CNXCONNECTBACK setting.
- If there is a firewall between the CONNX client and server, we recommend setting CNXCONNECTBACK to 0 because the only port that needs to be open through the CONNX server machine firewall is the installation port (usually 6500). If CNXCONNECTBACK is 1, a port must be opened in the firewall on each of the client machines connecting to the server because the server is creating a new socket associated with a port on the client.

There is no performance gain with either setting of CNXCONNECTBACK. All communication is done over one socket for each connection; it only changes which part of CONNX creates the socket.

Default = 0 enable.

Environments: Server
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CNXCONNECTBACK

Unix Environment Variable: CNXCONNECTBACK
VMS Logical: CNXCONNECTBACK
Started Task: CNXCONNECTBACK
CICS: CNXCONNECTBACK

CNXDECNETTASK

On VMS, determines whether DECNet Task-to-Task is used to start new server processed. By default CONNX will attempt this method, and if it fails, it will use the CREPRC system call instead. The difference being, with DECNet Task-to-Task, the server process type is "Network", and with CREPRC, the server process type is "Interactive", which may have VMS licensing implications. This setting is the opposite of CNXUSEMBX.

CNXDECNETTASK = 1

Default = 1

Environments: Server, VMS

VMS Logical: CNXDECNETTASK

CNXDIR

VMS logical that points to the location of the CONNX server binaries.

Environments: Server, OpenVMS
VMS Logical: CNXDIR

CNXHASH

Use the standard hashing algorithm on the CONNX client and server in almost all circumstances.

If your older mainframe system uses too many CPU cycles running DataSync, contact Technical Support and if it is appropriate, we will help you change your hashing routine.

Caution: Do not adjust or change this setting unless instructed to by a CONNX Technical Support representative.

To enable the selection of the hash function for incremental updates, enter one of the following value:

0 = fnv hash routing (default)

If CNXHASH is set to 0, or the configuration setting for HASH is 0, then the Fowler / Noll / Vo (FNV) Hash is chosen. Here is a description of the algorithm:

http://www.isthe.com/chongo/tech/comp/fnv/

1 = oy_djb 64bit hash routine

If CNXHASH is set to 1, or the configuration setting for HASH is 1, then the Ozan Yigit (SDBM) Hash combined with the Daniel J. Bernstein Hash is chosen. Here is a description of the algorithm:

http://www.cse.yorku.ca/~oz/hash.html

2 = bj 64bit hash routine

If CNXHASH is set to 2, or the configuration setting for HASH is 2, then the Bob Jenkins Hash is chosen. Here is a description of the algorithm:

http://www.cse.yorku.ca/~oz/hash.html

3 = oy 32bit hash routine

If CNXHASH is set to 3, or the configuration setting for HASH is 3, then the Ozan Yigit (SDBM) Hash is chosen. Here is a description of the algorithm:

http://www.cse.yorku.ca/~oz/hash.html

4 = djb 32bit hash routine

If CNXHASH is set to 4, or the configuration setting for HASH is 4, then the Daniel J. Bernstein Hash is chosen. Here is a description of the algorithm:

http://www.cse.yorku.ca/~oz/hash.html

5 = returns zero

If CNXHASH is set to 5, or the configuration setting for HASH is 5, then the NULL is chosen. The NULL hash does nothing and returns zero. It is for benchmarking purposes only and cannot be used for any other purpose.

6 = oy_djb 32bit + djb 32bit hash routine

If CNXHASH is set to 6, or the configuration setting for HASH is 6, then an alternate form of the Ozan Yigit (SDBM) Hash combined with the Daniel J. Bernstein Hash is chosen. Here is a description of the algorithm:

http://www.cse.yorku.ca/~oz/hash.html

7 = u_mac 64bit hash routine

If CNXHASH is set to 7, or the configuration setting for HASH is 7, then the UMAC Hash is chosen if the server is NOT a VAX. Here is a description of the algorithm:

http://fastcrypto.org/umac/

If the server is a VAX, then the program will fire an assert() and stop if data synchronizations are attempted.

If performing data synchronization using CNXHASH and HASH, the hash algorithm used by both client and server must match, or a full synchronization is performed for every sync operation.

Environments: Server, Windows, Unix, VMS, Mainframe
Windows Environment Variable: CNXHASH
Unix Environment Variable: CNXHASH
VMS Logical: CNXHASH
Started Task: CNXHASH
CICS: CNXHASH

CNXKBAUTHORIZE

On Unix, enables Kerberos authentication instead of standard unix password file authentication.

CNXKBAUTHORIZE = 1

Default = 0

Environments: Server

Unix Environment Variable: CNXKBAUTHORIZE

CNXLOCALIP

CNXLOCALIP can be used to assign an IP address to the data source server.

Environments: Server
Windows Environment Variable: CNXLOCALIP
Unix Environment Variable: CNXLOCALIP
VMS Logical: CNXLOCALIP
Started Task: CNXLOCALIP
CICS: CNXLOCALIP

CNX_LIBRARY_PATH

On Unix, the specified path, if provided, will be used to set the appropriate library path based on the unix operating system.

Environments: Server

Unix Environment Variable: CNX_LIBRARY_PATH

CNXLISTENER

CNXLISTENER points to the name of the listener program. It is used by CNXSTART on the mainframe. This option should not be changed unless instructed by CONNX Technical Support.

Environments: Server
Started Task: CNXLISTENER
CICS: CNXLISTENER

CNXMUALPHA

For Adabas, when enabled, will use an MU instead of an LA field during create table for a char or varchar field between 254 and 16381 bytes.

```
CNXMUALPHA = 1
```

Default = 0.

Environments: Server

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CNXMUALPHA

Unix Environment Variable: CNXMUALPHA

VMS Logical: CNXMUALPHA

Started Task: CNXMUALPHA

CICS: CNXMUALPHA

CNXNOPREAUTHORIZE

Enables / disables the CONNX user ID/password verification logic.

1 = does not validate userid/password.

0 = validates userid/password

Mainframe:

If set to 0, then the CONNX load libraries must be APF-authorized.

Additionally for Adabas: When set to 0, the RACF userid is made available in the attached buffer area of the target nucleus. From Adabas 8.3 onwards, this information is then available in UQ displays, CLOG, PLOG, etc.

This setting affects RACF

Unix/VMS:

If set to 1, the user name/password combination is not validated on the system.

Default = 0

Environments: Server, Mainframe, Unix, VMS
Unix Environment Variable: CNXNOPREAUTHORIZE
VMS Logical: CNXNOPREAUTHORIZE
Started Task: CNXNOPREAUTHORIZE
CICS: CNXNOPREAUTHORIZE

CNXNOPOST

Disables the post back message from the server to the listener. Disabling this postback will cause {serverlist} to no longer function.

```
CNXNOPOST = 1
```

Default = 0.

Environments: Server
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CNXNOPOST

Unix Environment Variable: CNXNOPOST
VMS Logical: CNXNOPOST
Started Task: CNXNOPOST
CICS: CNXNOPOST

CNXNOQIO

Disables the use of QIO for TCP/IP communications on OpenVMS for RMS, RDB and Codasyl DBMS OpenVMS .

1 = do not use QIO

0 = use QIO if possible

Default = 0

Environments: Server, VMS

VMS Logical: CNXNOQIO

CNX_NO_TIMER

Disables the heartbeat timer between the client and the server.

```
CNX_NO_TIMER = 1
```

Do not enable this setting unless instructed to do so by CONNX Technical Support.

Default = 0.

Environments: Server

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CNX_NO_TIMER

Unix Environment Variable: CNX_NO_TIMER

VMS Logical: CNX_NO_TIMER

Started Task: CNX_NO_TIMER

CICS: CNX_NO_TIMER

CNXOLDLICENSE

Enables the pre CONNX 10.5 licensing methodology for RMS, RDB and Codasyl DBMS OpenVMS users, and does not use the current license server for these data sources.

1 = use pre CONNX 10.5 licensing method.

0 = use current license server technology

Default = 0

Environments: Server, VMS

VMS Logical: CNXOLDLICENSE

CNX_PASS_TICKETS

Enables / disables support for RACF PASSTICKETS

1 = enable RACF passticket support

0 = disable RACF passticket support

Not for typical customer usage. Please only set if using RACF passtickets or instructed to by CONNX Technical Support.

Note: This setting applies to Adabas **only**.

Default = 0

Environments: Server, Mainframe (Started Task/Batch only. Does not apply to CICS)

Started Task: CNX_PASS_TICKETS

CNXPOSTSERVERNAME

Specify the TCP/IP name of the server for use when performing the postback (see CNXNOPOST). This setting is typically only required when local loopback is not functioning on the server.

```
CNXPOSTSERVERNAME= <name or IP Address of the server>
```

Default = 0.

Environments: Server
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CNXPOSTSERVERNAME

Unix Environment Variable: CNXPOSTSERVERNAME
VMS Logical: CNXPOSTSERVERNAME
Started Task: CNXPOSTSERVERNAME
CICS: CNXPOSTSERVERNAME

CNXRUNPORT

CNXRUNPORT will update the server port number.

The server port is the TCP/IP port that the CONNX TCP/IP Listener program accepts messages from the CONNX Windows client interfaces (ODBC, OLE DB, and .NET).

To change the server port number of 6500, enter the new port number:

CNXRUNPORT = 6600

This setting may affect ConnectPort.

Default value = 6500.

Environments: Server
Windows Environment Variable: CNXRUNPORT
Unix Environment Variable: CNXRUNPORT
Started Task: CNXRUNPORT
CICS: CNXRUNPORT

CNXSELECT

Determines whether the TCP/IP call "select" is issued before a recv or send to determine if the socket is in a "ready"state.

This option should not be changed unless instructed by CONNX Technical Support.

```
CNXSELECT = 1
```

Default = 1

Environments: Server

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CNXSELECT

Unix Environment Variable: CNXSELECT

VMS Logical: CNXSELECT

Started Task: CNXSELECT

CICS: CNXSELECT

CNXSOCKETTIMEOUT

When CNXCONNECTBACK is enabled, this value is the number of seconds for the server to wait while attempting to connect back to the client.

CNXSOCKETTIMEOUT = 30

Default = 58.

Environments: Server
Windows Environment Variable: CNXSOCKETTIMEOUT
Unix Environment Variable: CNXSOCKETTIMEOUT
VMS Logical: CNXSOCKETTIMEOUT
Started Task: CNXSOCKETTIMEOUT
CICS: CNXSOCKETTIMEOUT

CNXTCPIPBUFFER

Determines the requested send and receive buffer size at the TCP/IP protocol level with the setsockopt call.

This option should not be changed unless instructed by CONNX Technical Support.

```
CNXTCPIPBUFFER = 16384
```

Default = 16384

Environments: Server

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CNXTCPIPBUFFER

Unix Environment Variable: CNXTCPIPBUFFER

VMS Logical: CNXTCPIPBUFFER

Started Task: CNXTCPIPBUFFER

CICS: CNXTCPIPBUFFER

CNXTRUEUCX

Depricated. See CNXNOPREAUTHORIZE.

CNXUSEMBX

On VMS, determines whether DECNet Task-to-Task is used to start new server processed. By default CONNX will attempt DECNet Task-to-Task, and if it fails, it will use the CREPRC system call instead. The difference being, with DECNet Task-to-Task, the server process type is "Network", and with CREPRC, the server process type is "Interactive", which may have VMS licensing implications. This setting is the opposite of CNXDECNETTASK. To disable DECNet Task-to-Task, set CNXUSEMBX=1.

CNXUSEMBX = 1

Default = 0

Environments: Server, VMS

VMS Logical: CNXUSEMBX

COBOL_COMP_BYTESTEP

Normal processing converts COBOL COMP or BINARY field types to a 1, 2, 4 or 8 byte field depending on the precision.

If you want CONNX to treat the COBOL COMP or BINARY field type as a binary field that contains an exact number of bytes to satisfy the decimal precision requirements, enable COBOL_COMP_BYTESTEP:

```
COBOL_COMP_BYTESTEP = nonzero value
```

When COBOL_COMP_BYTESTEP is enabled, the decimal precision of the field will only use the exact number of bytes that are necessary.

Example:

If COBOL_COMP_BYTESTEP is enabled, the field definition

```
PIC S9(05) COMP.
```

would result in a var-binary integer type that has a field byte length of 3 instead of 4.

Environments: Client, Windows,

Configuration Manager: CDD Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = COBOL_COMP_BYTESTEP

COLLATINGSEQUENCE

To use the collating sequence from the mainframe server for sorting, set the COLLATINGSEQUENCE value::

COLLATINGSEQUENCE=1145

If CONNATINGSEQUENCE is set to 1145, the value will represent the EBCDIC code page to be use as the collating sequence when sorting data..

Default = 0 use ASCII collating sequence.

A summary of the limitations of this feature:

1) Only works with non-unicode data
2) Comparisons are done in codepage 1145 (which should be suitable for any EBCDIC platform)
3) There will be a small performance hit due to the extra translations.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = COLLATINGSEQUENCE

Unix Registry Setting: CONNX.COLLATINGSEQUENCE

COMPRESS

To speed up data transmission when data is sent over a network, turn on CONNX data compression:

COMPRESS=1

If Compress = 1, CONNX applies a data compression algorithm that remove repeating characters from the data stream.

Default = 0 (no compression applied).

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = Compress

Unix Registry Setting: CONNX.COMPRESS

COMMITCOUNT

CONNX will automatically issues a commit every specified number of inserts, regardless of transaction mode. Important warning: Setting COMMITCOUNT to a value other that zero can cause transactions to be committed earlier than expected, and subsequent rollbacks will be ineffective or incomplete.

```
COMMITCOUNT=1000
```

Default = 0.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = COMMITCOUNT

Unix Registry Setting: CONNX.COMMITCOUNT

CONNECTIONPOOLING

When Connection Pooling is enabled, at disconnect time, connections are stored in a dynamically sized "pool". The connection remains in the pool until it is either re-used, or the CONNECTIONPOOLINGTIMEOUT number of milliseconds has elapsed.

```
CONNECTIONPOOLING = 1
```

CONNECTIONPOOLING = 0 disables connection pooling;

CONNECTIONPOOLING= 1 enable connection pooling for all applications. This setting is only supported by systems that use TCP/IP.

CONNECTIONPOOLING= 2 signals that connection pooling is enabled, but only for the following specific applications:

- CNXJDBC - Our JDBC Service
- IISADMIN - Internet Information Server
- ASPNET_WP - ASP.NET
- INETINFO - Internet Information Server (used in debugging)
- DLLHOST - Any COM/DCOM DLL based applications
- SVCHOST - Any non-COM/DCOM DLL based applications
- SQLSERVR - SQL Server

Note: When using Connection Pooling with Adabas, this timeout value must be smaller than any of the Adabas inactivity and ET timeout values, otherwise pooled connections could be timed out by Adabas and become invalid.

Default = 2

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = ConnectionPooling

Unix Registry Setting: CONNX.CONNECTIONPOOLING

CONNECTIONPOOLINGTIMEOUT

This setting determines the amount of time a connection remains idle in the connection pool, if CONNECTIONPOOLING is enabled.

CONNECTIONPOOLINGTIMEOUT = 600000

This setting is in milliseconds.

This is only valid when CONNECTIONPOOLING is enabled.

Connections idle for 10 minutes when set to the default (600000).

Note: When using Connection Pooling with Adabas, this timeout value must be smaller than any of the Adabas inactivity and ET timeout values, otherwise pooled connections could be timed out by Adabas and become invalid.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CONNECTIONPOOLINGTIMEOUT

Unix Registry Setting: CONNX.CONNECTIONPOOLINGTIMEOUT

CONNECTPORT

Specify a default port number to use when communicating to the CONNX server component.

CONNECTPORT=nnnnnn

Enter the desired port number where nnnnn represents the default TCP/IP Port number the CONNX server uses to communicate to the computer. The default value is 6500.

This setting may be affected by CNXRUNPORT.

Note: We recommend that you not change the default value.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CONNECTPORT

Unix Registry Setting: CONNX.CONNECTPORT

CONNECTRETURNPORT

When CNXCONNECTBACK is enabled, the specified port number will be reserved on the client, and the server component will connect back to the client on the specified TCP/IP port.

CONNECTRETURNPORT = 0 or VALID PORT NUMBER

This option applies when CONNX makes TCP/IP connections to a CONNX Listener on a remote server (for example when connecting to VSAM files on a IBM Mainframe, or RMS file on a VMS Server, or C-ISAM files on a UNIX server). CONNX initiates a connection to a listener, sends some connection information, and then disconnects. The listener start a new server process, and the server connects back to the client PC on a specified port.

This option controls the port that is used during that connect back process. If this option is zero (default) or not present, CONNX chooses the next available port on the client PC. If a specific port number is specified, then CONNX first attempts to use that port. If the port is unavailable, CONNX increments the port number until it finds an available port.

Default = 0.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CONNECTRETURNPORT

Unix Registry Setting: CONNECTRETURNPORT

CONNECTTIMEOUT

To set the connection timeout, set the following value:

CONNECTTIMEOUT=value

Replace value/Integer with the number of milliseconds to wait for a connection from a server.

Default: 30000 (30 seconds)

Note: The option applies to TCP/IP connections only. The value gives the number of milliseconds CONNX waits for a connection to complete before timing out.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CONNECTTIMEOUT

Unix Registry Setting: CONNECTTIMEOUT

CONNECTTRIES

To determine the number of times CONNX attempts a connection to a host server, set the following value:

CONNECTTRIES=3

By enabling this option with the value of 3, CONNX attempts to connect three times if the host displays a "busy" message or returns the message "Connection refused".

Default value = 2.

Note: The option applies to TCP/IP connections only. ConnectTries is the number of connection attempts (calls to the TCP/IP function "connect") before returning a failure on a connection request. One common cause of "Connection refused" is that the listener process has a backlog of connection requests.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CONNECTTRIES

Unix Registry Setting: CONNECTTRIES

CONNXENTERPRISESERVER

This setting is used by the CONNX Installation utility and does not need to be edited. It specifies whether the Enterprise Service was selected during the last installation.

CONNXREG_DISPLAY_OPTS

To display the current Adabas SQLRegistry key/value pairs output on a single line instead of multiple lines, set the following value:

```
CONNXREG_DISPLAY_OPTS = 1
```

On Unix systems, this allows you to use UNIX tools to operate on the data.

Default = 0.

Environments: Client, Windows, Unix, Adabas
Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = CONNXREG_DISPLAY_OPTS

Unix Environment Variable: CONNXREG_DISPLAY_OPTS

CONNXSTORESERVER

This setting is used by the CONNX Installation utility and does not need to be edited. It specifies whether CONNXStore was selected during the last installation.

CONTINUEONGPF

If set to the default value of 1, CONNX continues to process queries even when it encounters an error.

CONTINUEONGPF=1

Default = 1.

Environments: Client, Windows CONNX

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = CONTINUEONGPF

DATAFLEXMODE

To turn on DataFlex mode, set the following values:

DATAFLEXMODE = 1

Turns on DataFlex access mode. The default is PowerFlex access mode (DataFlex mode=0).
Default = 0.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = DATAFLEXMODE

Unix Registry Setting: DATAFLEXMODE

DataFlexStructureUpdate

To update the complete DataFlex file structure when performing "Update Statistics", set the following values:

```
DATAFLEXSTRUCTUREUPDATE = 0
```

This setting will update the entire DataFlex structure if set to 1.

Default = 0.

Environments: Client, Windows, CDD

Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = DATAFLEXSTRUCTUREUPDATE

Unix Registry Setting: CONNXCDD.OPTIONS.DATAFLEXSTRUCTUREUPDATE

DB_LOCALE

This Unix-only language setting is for Informix access.

Please consult your Informix documentation for information regarding specific valid values for your operating system.

Environments: Client

Unix Environment Variable: DB_LOCALE

DEBUG

Can be a value from 0 to 3. If Debug is greater than zero, enables CONNX debug tracing message logic is. The messages are written to the cnxrun.log

DEBUG = 1 - Minimal logging level

DEBUG = 2 - Moderate logging level

DEBUG = 3 - Full tracing

To begin full trace debugging, set the following values:

DEBUG = 3

Default = 0, which disables the tracing logic.

Environments: Client, Server, Windows, Unix, Mainframe
Configuration Manager: CONNX Settings; Current Key = CONNX, Key Value/Value Name = DEBUG

Unix Client Registry Setting: CONNX.DEBUG
Unix Server Environment Variable: DEBUG
Started Task: DEBUG
CICS: DEBUG

DEBUGLEVEL

To control the CONNX JDBC server debug output that goes to the log file, set the following value:

DEBUGLevel = 1

If DebugLevel = -1, only the startup message and error messages will be logged. All other debug messages will be suppressed.
If DebugLevel = 0, no messages are logged.(Debug output disabled)
If DebugLevel = 1, some messages are logged. This provides a moderate amount of debugging/status information without severely affect performance. (Normal debug output level)
If DebugLevel = 2, all startup, informational and error messages are logged. This diagnostic level will negatively impact performance. (Extreme debug output.)

DebugLevel only applies to the CONNX JDBC server.

Default = -1.

Environments: Client, Windows, Unix, Mainframe

Configuration Manager: CONNX Settings; Current Key = JDBC, Key Value/Value Name = DEBUGLEVEL

Unix Client Registry Setting: JDBC.DEBUGLEVEL

DebugLoc

To assign a specific location to the debugging function, set the following values:

DEBUGLOC = C:\serverlog (for example)

All activity is sent to a file called CNXSERVER.LOG that will exist in DEBUGLOC.

Environments: Client, Windows, Unix Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = DEBUGLOC

Unix Configuration Setting: DEBUGLOC

DebugVerbose

To specify the CONNX JDBC server debug verbosity output level, set the following value:

DebugVerbose = 1

If DebugVerbose = 0, some information is included in the debug messages. This provides a moderate amount of debugging/status information without severely affect performance

If DebugVerbose = 1, all information is included in the debug messages. This information level can negatively impact performance.

DebugVerbose only applies to the CONNX JDBC server.

Default = 0

Environments: Client, Windows, Unix, Mainframe

Configuration Manager: CONNX Settings; Current Key = JDBC, Key Value/Value Name = DEBUGVERBOSE

Unix Client Registry Setting: JDBC.DEBUGVERBOSE

DefaultCOBOLCHAR

To enable the specification of the default signed datatype when importing COBOL FD tables, set the following values:

```
DEFAULTCOBOLCHAR = <CONNX Data type of choice>
```

Supply a CONNX data type number in the Value Data text box.

Default = 1 (Text (Right Space Padded) for ISAM types or Char for relational types)

Environments: Client, Windows

Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = DEFAULTCOBOLCHAR

Unix Environment Variable: CONNXCDD.OPTIONS.DEFAULTCOBOLCHAR

DefaultCOBOLDEC

To enable the specification of the default signed datatype when importing COBOL FD tables, set the following values:

```
DEFAULTCOBOLDEC = <CONNX Data type of choice>
```

Supply a CONNX data type number in the Value Data text box.

Default = 315 (Signed Overpunch -> Decimal)

Environments: Client, Windows

Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = DEFAULTCOBOLDEC

Unix Environment Variable: CONNXCDD.OPTIONS.DEFAULTCOBOLDEC

DefaultCOBOLInt

To enable the specification of the default signed datatype when importing COBOL FD tables, set the following values:

```
DEFAULTCOBOLINT = <CONNX Data type of choice>
```

Supply a CONNX data type number in the Value Data text box. Default = 25 (Signed Overpunch -> Integer)

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = DEFAULTCOBOLINT

Unix Environment Variable: CONNXCDD.OPTIONS.DEFAULTCOBOLINT

DefaultOnDemand

To change the default database connection method to Connect on Demand, set the following value:

DefaultOnDemand=1

If DEFAULTONDEMAND is unset, or set to zero, the default application will be set to Connect to All Database in the CONNX Integrated Logon dialog box.

If DEFAULTONDEMAND is set to one, the default application will be set to Connect on Demand in the CONNX Integrated Logon dialog box.

Default = 0 (Connect to All Database)

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = DefaultOnDemand

Unix Registry Setting: CONNX.DEFAULTONDEMAND

DefaultMUPE

Use DefaultMUPE to change the default number of MUs and PEs on an Adabas import.

DefaultMUPE=5

If DefaultMUPE is unset, or set to a value less than one or a value greater than 191, the default value will be set to 5.

This value determines the default value of the of the Max Repeat field on the ADABAS Count Selection dialog that is displayed during an Adabas file import.

This value is only used at import time and only affects Adabas imports.

Default = 5

Environments: Client, Windows

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = DefaultMUPE

DefaultPort

Use DefaultPort to change the default TCP/IP port used by CONNX to connect to CONNX servers.

To change the default port number of 6500, enter the new port number:

DefaultPort = 6600

Default = 6500.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = DEFAULTPORT

Unix Environment Variable: CONNXCDD.OPTIONS.DEFAULTPORT

DefaultVMSCreatePath

To establish the destination of tables created with the Create Table SQL command with the RMS server component, set the following value:

```
DEFAULTVMSCREATEPATH = DKS600:[MYDIR]
```

Replace MYDIR with the destination folder of tables created with the Create Table SQL command.

Default = CNXDIR:

Environments: Client, Windows

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = DEFAULTVMSCREATEPATH

DOWNGRADELOCK

For Adabas, causes records that are locked when processing an update or delete statement to be immediately unlocked if the client determines the record does not match the where clause criteria. Normally all locks are released when the transaction has been committed..

DOWNGRADELOCK = 1

Default = 0.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = DOWNGRADELOCK

Unix Environment Variable: DOWNGRADELOCK

Started Task: DOWNGRADELOCK

CICS: DOWNGRADELOCK

EmptyString

To treat zero-length strings as null values, set the following value:

```
EMPTYSTRING=1
```

By enabling this option, CONNX treats zero-length strings as NULL. Required for some applications, including Oracle SQL Plus.
Default = 0.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = EMPTYSTRING

Unix Configuration Setting: EMPTYSTRING

ENCRYPT

When set to 1, this setting enables an encrypted connection between the CONNX client and the CONNX server.

ENCRYPT=1

If ENCRYPT = 1, CONNX applies a data encryption algorithm to the data stream between the client and server.

Default = 0 (no encryption applied).

The ENCRYPT setting only applies to databases that use the CONNX Listener/Server component for connections such as RMS, DBMS, RDB, Adabas, VSAM/QSAM, IMS, C-ISAM, D-ISAM, Microfocus COBOL, RM COBOL and Dataflex.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = ENCRYPT

Unix Registry Setting: CONNX.ENCRYPT

ENTIRENETWORK

When set to 1, this setting enables CONNX compliancy with Software AG's Entire Network product.

ENTIRENETWORK=0

Environments: Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = ENTIRENETWORK

Unix Environment Variable: ENTIRENETWORK

Started Task: ENTIRENETWORK

CICS: ENTIRENETWORK

ENTIRENETWORKMULTIFETCHFIXED

When set to 1, a specific Entire Network connection is applied to your EntireNetwork and MULTIFETCH becomes available as a setting.

ENTIRENETWORKMULTIFETCHFIXED=0

Default = 0.

Environments: Windows, Mainframe, Unix, Adabas
Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = ENTIRENETWORKMULTIFETCHFIXED

Unix Environment Variable: ENTIRENETWORKMULTIFETCHFIXED
Started Task: ENTIRENETWORKMULTIFETCHFIXED
CICS: ENTIRENETWORKMULTIFETCHFIXED

ERRORONTRUNCATE

During bulk loading operations such as DataSync syncs, Replication initial states or Insert-Select SQL statements string data is truncated to fit into the target columns. This is done so that the entire operation doesn't stop because one string did not fit into the target, it is just truncated to the correct size and inserted. This setting will cause an error when truncation occurs during one of these bulk loading operations. The error will stop the entire operation from proceeding if any data must be truncated to fit into the target.

ERRORONTRUNCATE = 1

Default = 0.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = ERRORONTRUNCATE

Unix Registry Setting: CONNX.ERRORONTRUNCATE

ESQNULL

During imports, when set to 1, Null Suppressed fields that contain empty values will be treated a SQL NULL. This setting only applies to FDT and Natural DDM imports. This is an import only setting.

ESQNULL=1

Default = 1.

Environments: Client, Windows

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = ESQNULL

FastMemorySize

To control how much physical memory is used to hold temporary results before using a memory mapped file, set the following value:

```
FASTMEMORYSIZE = value
```

Replace value with the amount of memory to spare for holding temporary results.

Default value is 16000000 (16 million).

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = FASTMEMORYSIZE

Unix Registry Setting: CONNX.FASTMEMORYSIZE

FASTPATHMATCH

CONNX for VSAM assumes that alternate index (path) cluster names have the same first two letters as the main cluster name - this assumption significantly speeds up index metadata retrieval.

Default = 1

Environments: Server, Mainframe

Started Task: FASTPATHMATCH

CICS: FASTPATHMATCH

FilterDataTypes

To filter data types by database type, set the following value:

```
FILTERDATATYPES=1
```

Default = 1. (Filter by database types.)

Environment: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = FilterDataTypes
Unix Registry Setting: CONNXCDD.OPTIONS.FILTERDATATYPES

FixMUPEName

This setting corrects the SYSOBJH import naming behavior of MUs and PEs.

FIXMUPENAME=1

Default = 0 (do not use the corrected behavior - for backward compatibility).

Using the Data Dictionary Manager, a SYSOBJH import of the ADABAS "EMPLOYEES" file will generate

.. "BONUS" (Set to 0 - default)

.. "EMPLOYEES_INCOME_BONUS" (Set to 1)

Environments: Client, Window, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = FIXMUPEName

FORCEADANUKEY

This setting enables/disables the use of super descriptors that contain NU or NC constituent fields where no criteria is specified.

FORCEADANUKEY=1

Default = 0 (do not return records with unset key fields).

FORCEADANUKEY only applies to NU / NC compound keys (super descriptors). For more information on how to use this setting, see Sub / Super Descriptor Handling.

Environments: Client, Windows, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = FORCEADANUKEY

Unix Registry Setting: CONNX.ADABAS.FORCEADANUKEY

ForceClientSort

Enabling this option will force all data to be resorted using the client codpage. Without this setting, the results of certain queries will be sorted in the codepage of the dataserver when the index used matches order by clause of the SQL statement. Enabling this option will result in slower performance, but data will be consistently ordered using the client codepage.

```
FORCECLIENTSORT= 1
```

Default = 0.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = FORCECLIENTSORT

Unix Registry Setting: CONNX.FORCECLIENTSORT

ForceTransactions

To force the support of distributed transactions when connecting to two or more data sources, and one of them does not support transactions, set the following value:

```
FORCETRANSACTIONS = 1
```

Default = 0.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = FORCETRANSACTIONS

Unix Registry Setting: CONNX.FORCETRANSACTIONS

FORCEVERSION

Determines whether the version banner is displayed for the CONNX listener.

```
FORCEVERSION = 0
```

Default = 1.

Environments: Server

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = FORCEVERSION

Unix Environment Variable: FORCEVERSION

VMS Logical: FORCEVERSION

Started Task: FORCEVERSION

CICS: FORCEVERSION

FreeformCOBOLFD

FREEFORMCOBOLFD allows you to tell CONNX whether your COBOL statements are in fixed or free format.

FreeformCOBOLFD = 1

COBOL source code can be in one of three formats:

1. COBOL (fixed, columns 7-72). This format skips columns 1-6 and 73- of each line, (the sequence or line number area and the Program Identification Number Area). Only the code in columns 7-72 is used.
2. COBOL (variable, columns 7-). This format skips columns 1-6, that is, the sequence or line number area. All other columns are assumed to contain code. Line length is not limited.
3. COBOL (free format) assumes that all columns are equal and visualizes all the code as if line numbers and the Program Identification Number Area didn't exist. Line length is not limited.

When it encounters COBOL statements, CONNX assumes that you are using the fixed columns 7-72 format (the first format described above) and will ignore anything it finds in columns 1-6 and 73-.

If you are using the free format COBOL format, set FREEFORMCOBOLFD to 1.

Default=0 (fixed, columns 7-72. Skip columns 1-6 and 73-)

Note: FujitsuCOBOLFD checks this setting to determine if there are comments in positions 1-6.

Environments: Client, Windows

Configuration Manager: CDD Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = FREEFORMCOBOLFD

FujitsuCOBOLFD

Use FujitsuCOBOLFD to ensure that mid-line comment information is not treated as part of a COBOL statement or definition.

FujitsuCOBOLFD = 1

Normally, COBOL does not allow inline comments. Comments appear on their own lines, with an * in position (Column) 7.

The Fujitsu COBOL compiler allows inline comments. If the Fujitsu COBOL compiler encounters a *> on a line, it treats the rest of the line as a comment.

If you are using the Fujitsu COBOL compiler and the *> comment operand, set FUJITSUCOBOLFD to 1.

Default=0 (only standard COBOL comment formatting applies).

Note: If there are comments in the left hand margin (first six positions), set FreeformCOBOLFD to 0. Otherwise CONNX will treat the COBOL statements as fixed column, 7-72 format.

Environments: Client, Windows
Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = FUJITSUCOBOLFD

Hash

Use the standard hashing algorithm on the CONNX client and server in almost all circumstances.

If your older mainframe system uses too many CPU cycles running DataSync, contact Technical Support and if it is appropriate, we will help you change your hashing routine.

Caution: Do not adjust or change this setting unless instructed to by a CONNX Technical Support representative.

```
[HKEY_LOCAL_MACHINE\SOFTWARE\CONNX SOLUTIONS\CONNX]
Hash=0
NUMBER (REG_DWORD)
```

Default = 0.

To enable the selection of the hash function for incremental updates, enter one of the following value:

0 = fnv hash routing (default)

If CNXHASH is set to 0, or the configuration setting for HASH is 0, then the Fowler / Noll / Vo (FNV) Hash is chosen. Here is a description of the algorithm:

http://www.isthe.com/chongo/tech/comp/fnv/

1 = oy_djb 64bit hash routine

If CNXHASH is set to 1, or the configuration setting for HASH is 1, then the Ozan Yigit (SDBM) Hash combined with the Daniel J. Bernstein Hash is chosen. Here is a description of the algorithm:

http://www.cse.yorku.ca/~oz/hash.html

2 = bj 64bit hash routine

If CNXHASH is set to 2, or the configuration setting for HASH is 2, then the Bob Jenkins Hash is chosen. Here is a description of the algorithm:

http://www.cse.yorku.ca/~oz/hash.html

3 = oy 32bit hash routine

If CNXHASH is set to 3, or the configuration setting for HASH is 3, then the Ozan Yigit (SDBM) Hash is chosen. Here is a description of the algorithm:

http://www.cse.yorku.ca/~oz/hash.html

4 = djb 32bit hash routine

If CNXHASH is set to 4, or the configuration setting for HASH is 4, then the Daniel J. Bernstein Hash is chosen. Here is a description of the algorithm:

http://www.cse.yorku.ca/~oz/hash.html

5 = returns zero

If CNXHASH is set to 5, or the configuration setting for HASH is 5, then the NULL is chosen. The NULL hash does nothing and returns zero. It is for benchmarking purposes only and cannot be used for any other purpose.

6 = oy_djb 32bit + djb 32bit hash routine

If CNXHASH is set to 6, or the configuration setting for HASH is 6, then an alternate form of the Ozan Yigit (SDBM) Hash combined with the Daniel J. Bernstein Hash is chosen. Here is a description of the algorithm:

http://www.cse.yorku.ca/~oz/hash.html

7 = u_mac 64bit hash routine

If CNXHASH is set to 7, or the configuration setting for HASH is 7, then the UMAC Hash is chosen if the server is NOT a VAX. Here is a description of the algorithm:

http://fastcrypto.org/umac/

If the server is a VAX, then the program will fire an assert() and stop if data synchronizations are attempted.

If performing a data synchronization using CNXHASH and HASH, the hash algorithm used by both client and server must match, or a full synchronization is performed for every sync operation

Environments: Client, Windows, Unix Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = HASH

Unix Registry Setting: CONNX.HASH

HonorDbidFileID

For Adabas, the DBID of the database normally overrides the DBID specified at the table level. If HONORDBIDFILEID = 1, CONNX uses the DBID specified in the CDD at the table level.

`HONORDBIDFILEID=1`

Default = 0.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas
Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = HONORDBIDFILEID

Unix Environment Variable: HONORDBIDFILEID
Started Task: HONORDBIDFILEID
CICS: HONORDBIDFILEID

HuffmanPowerFlex

HuffmanPowerFlex allows you to choose the compression and decompression method for PowerFlex applications that use DataFlex tables.

HuffmanPowerFlex=1

Normally, CONNX uses DataFlex compression and decompression methods for DataFlex data. PowerFlex has an different compression method than the ones used by DataFlex. If HuffmanPowerFlex is non-zero, then we will use the PowerFlex method to compress and decompress DataFlex tables.

Use HuffmanPowerFlex only if your DataFlex compressed files were designed using PowerFlex.

Default = 0 (use DataFlex methods to perform compression and decompression).

Environments: Client,Windows, DataFlex

Configuration Manager: CONNX Settings; Current Key = CONNX\Dataflex; Key Value/Value Name = HuffmanPowerFlex

IGNORECOMMITONREAD

For Adabas, ET commands will be suppressed even if explicitly requested by the client unless insert, update, or delete was performed..

IGNORECOMMITONREAD = 0

Default = 1.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas
Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = IGNORECOMMITONREAD

Unix Environment Variable: IGNORECOMMITONREAD
Started Task:IGNORECOMMITONREAD
CICS: IGNORECOMMITONREAD

ImportFlatOnly

If an Adabas file contains MUs or PEs, the MUs and PEs are normalized into separate tables during the FDT import process. In addition, a table with "_FLAT" appended to the name is create. This "flat" table contains fields for the MUs and PEs within each record. If you don't want the normalized tables imported, you can set the ImportFlatOnly CDD option to 1.

```
ImportFlatOnly=1
```

Default = 0. (Import all tables, not just the flattened tables.)

Note: This setting is only applicable to Adabas and FDT imports.

Environment: Client, Windows, Adabas
Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = ImportFlatOnly

ImportGroupNames

To import the names of groups, set the following value:

```
ImportGroupNames=1
```

Default = 0. (Do not import group names.)

Environment: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = ImportGroupNames
Unix Registry Setting: CONNXCDD.OPTIONS.IMPORTGROUPNAMES

ImportOverwrite

ImportOverwrite is used to determine whether to replace the existing CDD definitions.

```
IMPORTOVERWRITE=0
```

If ImportOverwrite = 0, existing table will not be overwritten when re-importing.

If ImportOverwrite = 1, overwrite existing tables when re-importing.

If ImportOverwrite = 2, preserve Column Long Names.

Default = 1.

Environment: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = ImportOverwrite
Unix Registry Setting: CONNXCDD.OPTIONS.IMPORTOVERWRITE

ImportPrefix

To replace prefixes attached to COBOL FD imported fields, use the following:

```
ImportPrefix="" (enter the prefix string)
```

Use ImportPrefix in conjunction with ReplacePrefix to exchange the ReplacePrefix string with the ImportPrefix string.

If ReplacePrefix does not exist (or is ""), then ImportPrefix prefaces the field name.

Example:

If the field is "BBB_Customer" and

1. ImportPrefix is "AAA_"
2. ReplacePrefix does not exist

the field in the CDD would be "AAA_BBB_Customer".

Environment: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = ImportPrefix
Unix Registry Setting: CONNXCDD.OPTIONS.IMPORTPREFIX

ImportProcedures

During the CONNX Data Dictionary import process, normally all tables, views, and stored procedures can be imported into the CDD. The ImportProcedures registry setting can be changed to remove all the stored procedures from the Import Table Selection window.

To hide stored procedures so they do not show up in the Import Table Selection window, set ImportProcedures to zero:

```
ImportProcedures=0
```

Default = 1. (Display Stored Procedures in Import Table Selection window)

Environment: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = ImportProcedures

Unix Registry Setting: CONNXCDD.OPTIONS.IMPORTPROCEDURES

ImportViews

During the CONNX Data Dictionary import process, normally all tables, views, and stored procedures can be imported into the CDD. The ImportViews registry setting can be changed to remove all the views from the Import Table Selection window.

To hide views so they do not show up in the Import Table Selection window, set ImportViews to zero:

```
ImportViews=0
```

Default = 1. (Display Import Views in Import Table Selection window)

Environment: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = ImportViews
Unix Registry Setting: CONNXCDD.OPTIONS.IMPORTVIEWS

INFORMIXDIR

This Unix-only language setting contains the location of the Informix run-time libraries.

Please consult your Informix documentation for information regarding specific valid values for your operating system.

Environments: Client

Unix Environment Variable: INFORMIXDIR

INICONVERTEDTOREGISTRY

This setting is used by the CONNX Installation utility and does not need to be edited. It specifies whether INI settings from windows 95 and below have been converted to the windows registry.

INST32

This setting is used by the CONNX Installation utility and does not need to be edited. The location of the CONNX installation. When looking at the 32 bit registry it is the 32 bit CONNX file location. When looking at the 64 bit registry it is the 64 bit CONNX file location.

IntegratedSecurityMode

This Windows-only setting allows an administrator to pre-create Windows domain based User ID's in a CDD. This is useful in environments where the administrator wishes to use maximum CDD security. When using NT Integrated security in the default mode, the first time a user connects to a CDD, a special unique NT_INTEGRATED User ID will be created for them in the CDD that is based on their Windows credentials. If an administrator wishes to lock the CDD so no User ID's can be created, setting INTEGRATEDSECURITYMODE to a value of 2 in the CONNX Configuration Manager under the CONNX key will allow this.

To use this feature:

- Set INTEGRATEDSECURITYMODE to a value of 2 using the CONNX Configuration Manager on every Windows machine where a CONNX client will be making a connection.
- In the CDD Administrator, select Tools|Options from the main menu. Select Enable NT Integrated Security from the CONNX Data Dictionary Global Configuration Options dialog.
- From the Security menu, select Users. Select Add User from the CONNX Users dialog.
- In the user name field, enter the username in the following format *UserName*$#NT#$*_DomainName*
    - Please note the underscore '_' between $#NT#$ and the domain name
    - The UserName must be the user's Windows domain User Name
    - DomainName is the name of the Windows Domain the user is logging in from.

`IntegratedSecurityMode=2`

Where:

value = 0. (Default. System generated unique user id)

value = 1 (Reserved. do not use)

value = 2 (Administrator generated user id - must follow format as described in above text)

Environment: Client, Windows

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = INTEGRATEDSECURITYMODE

Unix Registry Setting: N/A

JDBCCONNECTIONPOOLING

Enables total connection pooling (including pooling of the CDD) at the JDBC level:

```
JDBCONNECTIONPOOLING = 1
```

Default = 0.

Environments: Client, Windows, Unix

Configuration Manager: CONNX JDBC Settings; Current Key = JDBC; Key Value/Value Name = JDBCCONNECTIONPOOLING

Unix Registry Setting: CONNX.JDBC.JDBCCONNECTIONPOOLING

JDBCCONNECTIONPOOLINGTIMEOUT

Configures the connection pooling timeout when JDBCCONNECTIONPOOLING is enabled (in milliseconds):

```
JDBCONNECTIONPOOLINGTIMEOUT = 300000
```

Default = 300000.

Environments: Client, Windows, Unix

Configuration Manager: CONNX JDBC Settings; Current Key = JDBC; Key Value/Value Name = JDBCCONNECTIONPOOLINGTIMEOUT

Unix Registry Setting: CONNX.JDBC.JDBCCONNECTIONPOOLINGTIMEOUT

JDBCSERVER

This setting is used by the CONNX Installation utility and does not need to be edited. It specifies whether the JDBC Server Service was selected during the last installation.

JoinCacheMultiplier

To control how many memory map views are created to increase performance, set the following value:

```
JOINCACHEMULTIPLIER = 1
```

The total number of memory map views is log(# of rows) * JOINCACHEMULTIPLIER.

Default = 3.

Environments: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = JOINCACHEMULTIPLIER
Unix Registry Setting: CONNX.JOINCACHEMULTIPLIER

JoinCacheSize

To determine how much memory is used for keeping key data when performing in-memory binary searches.

```
JOINCACHESIZE = 32000000
```

Default = 16000000 (16 million)

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = JOINCACHESIZE

Unix Registry Setting: CONNX.JOINCACHESIZE

JoinCount

To configure the CONNX join count, which determines the number of keys to use in a single pass when performing joins, set the following value:

```
JoinCount=nnn
```

Default = 250 (maximum value).

When using Rdb 6.x, the recommended setting is 8.

Environments: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = JoinCount
Unix Registry Setting: CONNX.JOINCOUNT

JOINMUDESC

This setting determines if descriptors on MU and PE tables will be used in join optimization.

By default they will not be used.

Here is an example - a where clause of "value in ('A00', 'A10')" - the value field is a PE with a descriptor on it.

The first retrieval for 'A00' may bring back ISN 1 2 8

The second retrieval for 'A10' may bring back ISN 3 8 10

ISN #8 has 2 occurrences, one with value 'A00' and one with value 'A10' -

This causes in correct SQL records when the same ISN is returned multiple times for different values.

If it is known for certain that there is a unique set of values in the MU across all ISNs, then this setting can safely be set to 1.

Default = 0.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX/ADABAS; Key Value/Value Name = JOINMUDESC

Unix Registry Setting: CONNX.JOINMUDESC

KEEPHYPHENS

For Adabas, KEEPHYPHENS enables SYSOBJH imports to eliminate the default behavior of having their names import into a CDD with hyphens.

If KEEPHYPHENS = 1, hyphens are retained during the import procedure.

KEEPHYPHENS=1

Default = 0 (hyphens are not retained during import procedures)

Environments: Client, Windows, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = KEEPHYPHENS

Unix Environment Variable: KEEPHYPHENS

KEEPGROUPS

For Adabas, during FTP Import typically the high level group name is not imported, only the fields that make up the group (PE). by enabling this setting, the group fields themselves are also imported.

KEEPGROUPS = 1

Default = 0.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas
Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = KEEPGROUPS

Unix Environment Variable: KEEPGROUPS
Started Task:KEEPGROUPS
CICS: KEEPGROUPS

LICENSESERVER

The name of the server hosting the CONNX License server.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = LICENSESERVER

Unix Registry Setting: CONNX.LICENSESERVER

LICENSESERVERPORT

The port used to connect to the CONNX License server.

Default = 7501

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = LICENSESERVERPORT

Unix Registry Setting: CONNX.LICENSESERVERPORT

LICENSESOURCE

This setting is used by the CONNX Installation utility and does not need to be edited. This specifies the location of the source license files that was specified during the last installation.

LISTENQUOTA

Determines the maximum TCP/IP backlog of sockets for the VSAM server running under VSE/CICS. Used as a parameter to the listen command..

This option should not be changed unless instructed by CONNX Technical Support.

```
LISTENQUOTA = 5
```

This only applies on VSE/CICS systems.

Default = 5

Environments: Server
CICS: CNXBARNARD

LogLoc

Use LogLoc to change the default CONNX JDBC server log file location.

To change the CONNX JDBC server log file location, enter the following value:

LogLoc = C:\MyCONNX\JDBClogs

Default = current working directory (C:\CONNX32\)

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = JDBC; Key Value/Value Name = LogLoc

Unix Configuration Setting: JDBC.LOGLOC

LowerCaseOnly

To force column and table names to appear in lowercase upon import, set the following value:

```
LOWERCASEONLY=1
```

Forces column and table names to appear in lowercase upon import.

Default = 0.

Environment: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = LowerCaseOnly
Unix Registry Setting: CONNXCDD.OPTIONS.LOWERCASEONLY

LSNDEBUG

This setting configures tracing for the listener process only.

It can be a value from 0 to 3. If LSNDebug is greater than zero, enables CONNX debug tracing message for the listener only. The messages are written to the cnxrun.log

LSNDEBUG = 1 - Minimal logging level

LSNDEBUG = 2 - Moderate logging level

LSNDEBUG = 3 - Full tracing

To begin full trace debugging, set the following values:

LSNDEBUG = 3

Default = 0, which disables the tracing logic.

Environments: Client, Server, Windows, Unix, Mainframe

Configuration Manager: CONNX Settings; Current Key = CONNX, Key Value/Value Name = LSNDEBUG

Unix Client Registry Setting: CONNX.LSNDEBUG

Unix Server Environment Variable: LSNDEBUG

Started Task: LSNDEBUG

CICS: LSNDEBUG

MapFileIncrementSize

To control the extension size of a memory mapped file for holding temporary results, set the following value:

```
MAPFILEINCREMENTSIZE = 2000000
```

Default = 16000000

Environment: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = MapFileIncrementSize
Unix Registry Setting: CONNX..MAPFILEINCREMENTSIZE

MaxArrayCount

MaxArrayCount is an ADABAS setting that determines the default maximum number of occurrences for an MU or PE during the import process.

If the MU or PE occurrence count exceeds the old maximum of 192, please increase this value to the new maximum, up to the size of 65535.

If this value is changed, the table must be re-imported in order to see the effect.

Default = 192.

Environments: Windows

Configuration Manager: CONNX Settings; Current Key = CONNX/Adabas; Key Value/Value Name = MAXARRAYCOUNT

MaxRowsCompared

MaxRowsCompared can prevent runaway queries in non-relational databases.

MAXROWSCOMPARED = (ANY VALUE)

CONNX initially searches the number of records specified in MaxRowsCompared. If there is a match (at least one record returned), the entire database is searched. If there no match (no records returned), the search stops and an error is returned.

Default = 0.

Environments: Server, Windows, Unix, Mainframe
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = MAXROWSCOMPARED

Unix Environment Variable: MAXROWSCOMPARED
Started Task: MAXROWSCOMPARED
CICS: MAXROWSCOMPARED

MaxRowsFetched

To define the number of rows that are read by the database table in a single query scan, set a specific value in the following:

```
MAXROWSFETCHED = (ANY VALUE)
```

Default = 0.

See MAXROWSFETCHEDFAIL to determine whether MAXROWSFETCHED queriers should terminate normally or generate an error message.

Environments: Server, Windows, Unix, Mainframe
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = MAXROWSFETCHED

Unix Environment Variable: MAXROWSFETCHED
Started Task: MAXROWSFETCHED
CICS: MAXROWSFETCHED

MaxRowsFetchedFail

To determine whether a MAXROWSFETCHED query will either terminate normally or will generate an error depening on how the query terminates, set MAXROWSFETCHEDFAIL:

```
MAXROWSFETCHEDFAIL = 0
```

If MAXROWSFETCHEDFAIL = 0, once the database has read the number of rows specified by MAXROWSFETCHED the query will terminate normally.

If MAXROWSFETCHEDFAIL =1, once the database has read the number of rows specified by MAXROWSFETCHED, the query will generate an error message.

Default = 1.

Environments: Server, Windows, Unix, Mainframe
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = MAXROWSFETCHEDFAIL

Unix Environment Variable: MAXROWSFETCHEDFAIL
Started Task: MAXROWSFETCHEDFAIL
CICS: MAXROWSFETCHEDfIL

MAXSOCKET

Determines the maximum number of TCP/IP sockets to be used by the data server, and the maximum socket backlog to be used in the listen command..

This option should not be changed unless instructed by CONNX Technical Support.

```
MAXSOCKET = 1024
```

Default = 1024 (but can vary by operating system)

Environments: Server

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = MAXSOCKET

Unix Environment Variable: MAXSOCKET

VMS Logical: MAXSOCKET

Started Task: MAXSOCKET

CICS: MAXSOCKET

MISMATCHON

When performing a SYSOBJH IMPORT, there might be cases where the parser has difficulty with a DDM and translates it as a mismatch, so that "false" error messages are produced.

By setting MISMATCHON to 0, such error messages are eliminated. Use this setting only when such "false" error messages appear.

MISMATCHON=0

Default = 1.

Environments: Client, Windows, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = MISMATCHON

Unix Registry Setting: CONNX.ADABAS.MISMATCHON

MIXEDCASEGRAMMAR

When performing an Adabas import on a file that has a mixed case FDT, set this value to 1 to import the short names the same case as the FDT. Setting the value to 0 will cause the short names to be imported as upper case regardless of the case in the FDT. The default value is 0.

MIXEDCASEGRAMMAR = 1

Default = 0.

Environments: Client, Windows, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = MIXEDCASEGRAMMAR

Unix Registry Setting: CONNX.ADABAS.MIXEDCASEGRAMMAR

MULTIFETCH

If non zero 0, instructs CONNX for Adabas to request that multiple records be returned per read command.

MULTIFETCH = 1

Default = 1.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas
Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = MULTIFETCH

Unix Environment Variable: MULTIFETCH
Started Task: MULTIFETCH
CICS: MULTIFETCH

MUPESUPPORT

If non zero, this setting enables MUs within Periodic Groups to be represented in a new Relational Table. They will have an additional CONNX virtual column ("cnxarraycolumn") that represents the occurrence of the Periodic Group.

```
MUPESUPPORT=0
```

Default = 1.

Environments: Client, Windows, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = MUPESUPPORT

Unix Registry Setting: CONNX.ADABAS.MUPESUPPORT

NLS_LANG

This Unix-only language setting is for Oracle access.

Please consult your Oracle documentation for information regarding specific valid values for your operating system.

Environments: Client

Unix Environment Variable: NLS_LANG

NoLogo

To turn off the CONNX .AVI file, set the following value:

```
NOLOGO=1
```

Default = 0.

Environments: Client, Windows
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = NOLOGO

NoProcedures

To disable the import of stored procedures, use the following value:

```
NOPROCEDURES = 1
```

Default = 0 (Import stored procedures).

Environments: Client, Windows

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = NoProcedures

NoViews

To disable the import of views, use the following value:

```
NOVIEWS = 1
```

Default = 0 (Import views.).

Environments: Client, Windows
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = NoViews

NoWait

To add the suffix the "NO WAIT" clause to the end of the Oracle SELECT ... FOR UPDATE command, use the following value:

```
NOWAIT = 1
```

Default = 0 (do not add suffix).

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX\ORACLE; Key Value/Value Name = NOWAIT

Unix Registry Setting: CONNX.ORACLE.NOWAIT

NumberConvert

For either Oracle or PostgreSQL, to store fractional numbers as well as whole numbers in a number field, add the following registry entry:

```
NUMBERCONVERT=1
```

Enables fractional numbers to be included in a number field in an Oracle or PostgreSQL database.

Environments: Client, Windows

For Oracle:

Configuration Manager: CONNX Settings; Current Key = CONNX\ORACLE; Key Value/Value Name = NumberConvert

Unix Registry Setting: CONNX.ORACLE.NUMBERCONVERT

For PostgreSQL:

Configuration Manager: CONNX Settings; Current Key = CONNX\POSTGRESQL; Key Value/Value Name = NumberConvert

Unix Registry Setting: CONNX.POSTGRESQL.NUMBERCONVERT

OLEDB

This setting is used by the CONNX Installation utility and does not need to be edited. It specifies whether the OLEDB Desktop Adapter was selected during the last installation.

OLEDBE

This setting is used by the CONNX Installation utility and does not need to be edited. It specifies whether the OLEDB Enterprise Adapter was selected during the last installation.

OLEInit

The OLEInit setting controls how CONNX will initialize the OLE Subsystem. Based on what it finds in OLEInt, CONNX initializes the OLE Subsystem as either Apartment-Threaded or Multi-Threaded.

`OLEINIT=2`

- When OLEInit = 2, CONNX initializes the OLE Subsystem as Apartment-Threaded.

Apartment-threading, while allowing for multiple threads of execution, serializes all incoming calls by requiring that calls to methods of objects created by this thread always run on the same thread - the apartment/thread that created them.

- When OLEInit = 0, CONNX initializes the OLE Subsystem as Multi-Threaded.

Multi-threading (also called free-threading) allows calls to methods of objects created by this thread to be run on any thread. There is no serialization of calls - many calls may occur to the same method or to the same object or simultaneously.

Multi-threaded object concurrency offers the highest performance and takes the best advantage of multi-processor hardware for cross-thread, cross-process, and cross-machine calling, since calls to objects are not serialized in any way.

We recommend that OLEInit be set to 0 (Multi-Threaded). This allows the must flexibility and provides the best performance for multi-threaded applications. However, some OLEDB Providers may required their OLE Subsystem to be initialized as Apartment-Threaded. If this is necessary, set OLEINIT to 2.

The default value is 0.

Environments: Client, Windows

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = OLEInit

OLDERVERSION

Not for typical customer usage. Please only set if instructed to by CONNX Technical Support.

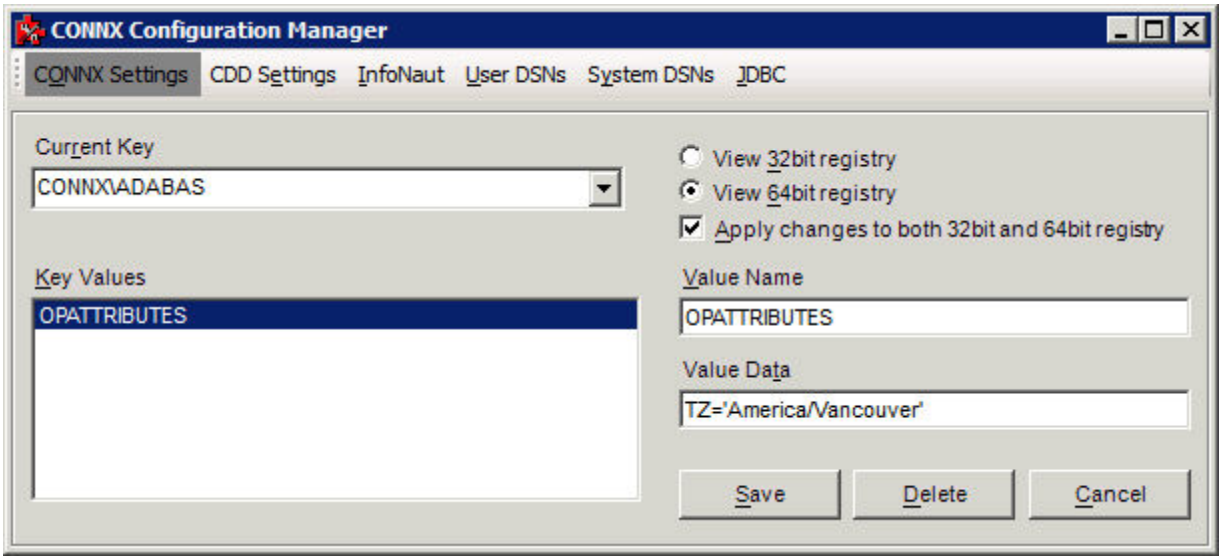
OPATTRIBUTES

OPATTRIBUTES instructs CONNX for Adabas to pass a specific attribute string to Adabas on the OP (open) call.

OPATTRIBUTES=<OP attribute string>

Default = not set.

Example: To set the time zone on the OP call using the TZ attribute to 'America/Vancouver':



This will cause CONNX for Adabas to pass the string TZ='America/Vancouver' as an attribute on the OP call.

NOTE: For the Mainframe, if the the OP attribute string contains an equal sign (=), the equal sign should be replaced with a comma (,) in the CNXPARMS file.

Example:

=OPATTRIBUTES=TZ,'America/Vancouver'

Environments: Client, Server,Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = OPATTRIBUTES

Unix Environment Variable: OPATTRIBUTES

Started Task:OPATTRIBUTES

CICS: OPATTRIBUTES

OPDONTCALL

If > 0, then instructs CONNX for Adabas to bypass the Open User Session call when a client-server connection is established.

OPDONTCALL=1

Default = 0.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = OPDONTCALL

Unix Environment Variable: OPDONTCALL

Started Task:OPDONTCALL

CICS: OPDONTCALL

OPMAXCMDID

If > 0, defines the maximum number of command IDs per Open User Session that may be active for a user at the same time.

This value cannot be greater than (1/240 * LQ) (where LQ is the ADARUN sequential command table length parameter value, which has a default of 10000).

OPMAXCMDID=100

Default = 0.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas
Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = OPMAXCMDID

Unix Environment Variable: OPMAXCMDID
Started Task:OPMAXCMDID
CICS: OPMAXCMDID

OPMAXHOLD

If > 0, defines the maximum number of records per Open User Session that a user may have in hold status at the same time. The default is the value set by the ADARUN NISNHQ parameter, and the maximum is 1/4 the value set by the ADARUN NH parameter minus 1, or 65535, whichever is smaller.

OPMAXHOLD=1000

Default = 0.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = OPMAXHOLD

Unix Environment Variable: OPMAXHOLD

Started Task:OPMAXHOLD

CICS: OPMAXHOLD

OPMAXISN

If > 0, defines the maximum number of ISNs per Open User Session that may be stored in the internal ISN element table resulting from the execution of an Sx command. Increasing the default setting decreases access to the Adabas Work dataset. The maximum allowed is 1000.

OPMAXISM=500

Default = 0.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = OPMAXISN

Unix Environment Variable: OPMAXISN

Started Task:OPMAXISN

CICS: OPMAXISN

OPNONACTIVITY

For Adabas, determines the non activity timeout for adabas calls.

OPNONACTIVITY = 65535

Default = 65535.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas
Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = OPNONACTIVITY

Unix Environment Variable: OPNONACTIVITY
Started Task:OPNONACTIVITY
CICS: OPNONACTIVITY

OPREADONLY

If > 0, instructs the CONNX for Adabas Open User Session logic to request an access-only user session.

Read-only users may not issue hold, update, delete, add record, ET (End Trasaction), or BT (Backout Transaction) commands.

OPREADONLY=1

Default = 0 = false.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = OPREADONLY

Unix Environment Variable: OPREADONLY

Started Task:OPREADONLY

CICS: OPREADONLY

OPSEARCHTIME

If > 0, defines per Open User Session the maximum amount of time in milliseconds permitted for the execution of an Sx command.

OPSEARCHTIME=3000

Default = 0.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas
Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = OPSEARCHTIME

Unix Environment Variable: OPSEARCHTIME
Started Task:OPSEARCHTIME
CICS: OPSEARCHTIME

OPTIMEOUT

For Adabas, if > 0, provides a user-specific transaction time limit in milliseconds for the Open User Session call. The limit must conform to the maximum specified by the ADARUN parameter MXTT. If this field contains binary zeros, the transaction time limit specified by the ADARUN TT parameter for the Adabas session is in effect.

OPTIMEOUT=50

Default = 0.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = OPTIMEOUT

Unix Environment Variable: OPTIMEOUT

Started Task:OPTIMEOUT

CICS: OPTIMEOUT

OracleBulkModeDisabledFlag

To indicate that Oracle does not use Bulk mode, set the following value:

```
ORACLEBULKMODEDISABLEDFLAG = 1
```

Default = 0 (Oracle Bulk mode enabled).

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = OracleBulkModeDisabledFlag

Unix Registry Setting: CONNX.ORACLEBULKMODEDISABLEDFLAG

ORACLE_HOME

This Unix-only setting contains the location of both the Oracle client libraries and the tnsnames.ora file. This is usually set to $CNXDIR/lib32 where CNXDIR is the location of the CONNX Client install for UNIX

Environments: Client

Unix Environment Variable: ORACLE_HOME

PacketSize

To establish the message buffers communications packet size between the CONNX32.dll file and database servers, set the following value:

```
PACKETSIZE=32000
```

Controls the size of message buffers between CONNX32.dll and the database servers. Default value is 8192; acceptable values are 4096 to 32000.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = PacketSize

Unix Registry Setting: CONNX.PACKETSIZE

PeekMessage

To cause CONNX to issue PeekMessage commands while waiting for TCP/IP commands to asynchronously complete, as required by some non-Microsoft TCP/IP stacks, set the following value:

```
PEEKMESSAGE=1
```

Default=0.

Note: This only applies to non-Microsoft TCP/IP stacks on the client computer.

Environments: Client, Windows

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = PeekMessage

PerformRangeChecks

If > 0, range checks are performed on data fields that are used during inserts and updates.

```
PERFORMRANGECHECKS=1
```

Default = 0.

Environments: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = PerformRangeChecks
Unix Registry Setting: CONNX.PERFORMRANGECHECKS

Port

Use Port to change the default CONNX JDBC server port.

Port = 7600

Default = 7500.

Environments: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = JDBC; Key Value/Value Name = PORT

Unix Configuration Setting: JDBC.PORT

PrecisionOverride

To enable the user to override the default SQL precision in the CONNX Data Dictionary for packed fields, use the following value:

```
PRECISIONOVERRIDE = 1
```

Default = 0 (Override turned off).

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = PrecisionOverride

Unix Registry Setting: CONNX.PRECISIOVERRIDE

QuadwordDouble

To import QuadWord data types as Double data types (Oracle Rdb), set the following value:

```
QUADWORDDOUBLE=1
```

Setting QuadwordDouble to = 1 enables QuadWord data types to be imported as Double data types.

The default is = 0, which means that QuadwordDouble data type is disabled.

Environments: Client, Windows, Unix, CDD
Configuration Manager: CDD Settings; Current Key = CONNXCDD; Key Value/Value Name = QuadwordDouble
Unix Registry Setting: CONNXCDD.QUADWORDDOUBLE

ReadTimeout

To change the default value for the DB2 TCP/IP timeout, use the following value:

```
READTIMEOUT = 1000000000
```

Default = 60000 milliseconds (60 seconds).

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = ReadTimeout

Unix Registry Setting: CONNX.READTIMEOUT

REISSUEOP

The Adabas "OP" command will be re-issued in the event that during query operations it receives an Adabas response response 9 with a subcode of 2 or 3 (on mainframe) or subcode of (OR, or TN) on open systems. When this setting is enabled, when a timeout is received, then ACE will issue a new OP command, and then retry the last operation.

```
REISSUEOP = 1
```

Environments: Client, Server,Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = REISSUEOP

Unix Environment Variable: REISSUEOP

Started Task: REISSUEOP

CICS: REISSUEOP

ReplacePrefix

To replace prefixes attached to COBOL FD imported fields, use the following:

```
ReplacePrefix="" (insert name of prefix)
```

Default = ""

Use ReplacePrefix in conjunction with ImportPrefix to exchange the ImportPrefix string with the ReplacePrefix string.

If ReplacePrefix does not exist (or is ""), then ImportPrefix prefaces the field name.

Example:

If the field is "BBB_Customer" and

1. ImportPrefix is "AAA_"
2. ReplacePrefix does not exist

the field in the CDD would be "AAA_BBB_Customer".

Environment: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = ReplacePrefix

Unix Registry Setting: CONNXCDD.OPTIONS.REPLACEPREFIX

RMSPaddedNoNulls

RMSPaddedNoNulls controls whether CONNX treats a field filled with binary zeros as either NULL, or as an empty string, when converted to SQL. The default behavior is to read a field with all binary zeros as SQL NULL. To change this behavior, set the variable to 1.

```
RMSPADDEDNONULLS = 1
```

Default = 0 (binary zeros are treated as a SQL NULL).

Environments: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNX\RMS; Key Value/Value Name = RMSPaddedNoNulls
Unix Registry Setting: CONNX.RMS.RMSPADDEDNONULLS

SBCCSID

To set a single-byte code page, set the following value:

CONNX
SBCCSID=<code page>

Default:

Windows = Native Locale
Mainframe = 37 (EBCDIC)
Unix = 1252

http://www-03.ibm.com/systems/i/software/globalization/codepages.html contains a list of possible mainframe settings.

See also Code Pages.

Environments: Client, Server, Windows, Unix, Mainframe
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = SBCCSID
Unix Environment Variable: SBCCSID
Started Task: SBCCSID
CICS: SBCCSID

SCTLogical

To define the RMS file name path when importing files via the SCT COBOL FD files option, set the following value:

```
SCTLOGICAL=logical
```

Example: SCTLOGICAL=SI$CONNX.

There is only one entry for this setting.

Environment: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = SCTLogical
Unix Registry Setting: CONNXCDD.OPTIONS.SCTLOGICAL

SetEnabled

Specifies whether or not a file will be set to enabled if it is in a disabled state under CICS.

```
SetEnabled = 1
```

If SETENABLED = 1, files will be enabled when accessed even if they were previously set to disabled.
If SETENABLED = 0, files that are in a disabled state will not be enabled and an error will be returned.

This only applies to CICS.

Default = 1 (Files will be enabled when accessed).

Environments: Server
CICS: SETENABLED

SetOpen

Specifies whether or not a file will be set to open if it is in a closed state under CICS.

```
SetOpen = 1
```

If SETOPEN = 1, files will be opened when accessed even if they were previously set to closed.
If SETOPEN = 0, files that are in a closed state will not be opened and an error will be returned.

This only applies to CICS.

Default = 1 (Files will be opened when accessed).

Environments: Server
CICS: SETOPEN

ShareConnectionCount

To control the number of logical connections mapped to a physical connection, set the following value:

```
SHARECONNECTIONCOUNT = VALUE
```

Replace value with the number of logical connections.

If ShareConnections = 0 (disabled), ShareConnectionCount is ignored.

Default = 5.

Environment: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = ShareConnectionCount
Unix Registry Setting: CONNX.SHARECONNECTIONCOUNT

ShareConnections

To turn connection sharing on or off, set the following value:

ShareConnections= 0, 1, or 2

If ShareConnections = 0, this option is disabled. Each connection from a single computer gets a dedicated socket to a server. This consumes more server resources.

If ShareConnections = 1, connection sharing is enables for all applications.

If ShareConnections = 2, connection sharing is enabled, but only for the following applications:

- CNXJDBC - Our JDBC Service
- IISADMIN - Internet Information Server
- aspnet_wp - ASP.NET
- INETINFO - Internet Information Server (used in debugging)
- DLLHOST - Any COM/DCOM DLL based applications
- Svchost - Any non-COM/DCOM DLL based applications
- SQLSERVR - SQL Server

If more than one connection is made on the same computer, using the same user ID and password, CONNX makes a single physical connection to the server instead of creating multiple connections.

Default = 2

Setting ShareConnections affects ShareConnectionCount.

Environment: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = ShareConnections

Unix Registry Setting: CONNX.SHARECONNECTIONS

SocketPause

To determine the length of time in milliseconds that CONNX waits for the TCP/IP stack to initialize set the following value:

```
SOCKETPAUSE=15000
```

Default = 0.

Note: This only applies to non-Microsoft TCP/IP stacks on the client computer.

Environment: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = SocketPause
Unix Registry Setting: CONNX.SOCKETPAUSE

STRICTCONVERSIONRULES

Whenever possible, CONNX does an implicit data conversion when it encounters data that is not represented as the type required. For example, if a character field is being returned as a number the value 'ABC123' will be automatically converted to 123. The value 'ABC' as a number however, has no meaning and a 0 will be returned.

In some situations, the preferred behavior would be to return an error rather than a 0. When set to 1, STRICTCONVERSIONRULES will cause an error to be returned when the conversion is not valid.

To set the STRICTCONVERSIONRULES parameter, set the following value:

```
STRICTCONVERSIONRULES = 1
```

Default = 0

Environment: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = STRICTCONVERSIONRULES

Unix Registry Setting: CONNX.STRICTCONVERSIONRULES

SUBQUERYROWESTIMATE

Use SUBQUERYROWESTIMATE to give CONNX a hint as to how it should report the query plan when {statistics} is used.

Some queries that have a sub query may use a less optimized query plan if the sub query returns more than one row. In this case, CONNX will automatically switch to the correct query plan at runtime, but the query plan reported by the {statistics} function may not be the same as the plan which was ultimately used. If you find that the query plan reported by {statistics} is different than what is being used at runtime due to multiple rows being returned in a sub query, set SUBQUERYROWESTIMATE to a value greater than 1.

The query plan will only change at run time if returning multiple rows in a sub query will cause duplicate rows to be returned if a more optimized path is used. For example, if an Adabas sub query uses a PE index and the sub query only returns 1 row, the PE index can be used. If the sub query returns multiple rows however, duplicates would be returned so CONNX must switch to a different query plan at runtime. Since the {statistics} function does not know the number of rows that will be returned, it is possible that it will return a query plan that is different than the one that will ultimately be used.

To tell CONNX to return a query plan for {statistics} that is based on a sub query returning multiple rows, set:

SUBQUERYROWESTIMATE=2

Default = 0.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = SUBQUERYROWESTIMATE

Unix Environment Variable: CONNX.SUBQUERYROWESTIMATE

SUPERDESCRIPTORASFIELD

For non-MU fields, enables the import feature to include a superdescriptor so that it appears as a field in the CONNX Data Dictionary.

SUPERDESCRIPTORASFIELD=0

The superdescriptor field is read-only.

Default = 1.

Environments: Server,Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = SUPERDESCRIPTORASFIELD

Unix Environment Variable: SUPERDESCRIPTORASFIELD

Started Task: SUPERDESCRIPTORASFIELD

CICS: SUPERDESCRIPTORASFIELD

TABLECACHE

To close open tables after processing data, set the following value:

```
TABLECACHE=0
```

CONNX caches tables automatically to save reconnect time.

Note: By default, CONNX keeps logical links open to any table/file that is accessed during the lifetime of a connection. For flat file data sources, such as RMS, VSAM, C-ISAM, DISAM, Micro Focus, Dataflex, and PowerFlex, TableCache controls how CONNX opens and closes the underlying data file.

For relational databases, such as DB2, Oracle, and SQL Server, for example, TableCache has very little impact because relational tables do not need to be "opened".

When TableCache is set to 0, CONNX opens the file in the appropriate access mode when necessary, and closes the file as soon as it is no longer needed.

When TableCache is set to 1, the first time a query is executed that accesses a table in a given mode (either read only, or read/write), CONNX opens that file in the specified mode, and keeps the file open for the duration of the connection. When TableCache is enabled, the amount of memory used by the data server is increased.

For backward compatibility, the setting of 2 and 1 have the same meaning.

Default = 2.

Environment: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = TABLECACHE
Unix Registry Setting: CONNX.TABLECACHE

TCPIP

This setting is used by the CONNX Installation utility and does not need to be edited. It specifies whether TCP/IP was selected during the last installation.

TCPIPDEBUG

To turn on TCP/IP Debug Messages, set the following values:

```
TCPIPDEBUG=1
```

Default=0 (no debug messages)

Note: Unless you are debugging, we recommend that you use the default setting.

Environment: Client, Windows
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = TCPIPDEBUG

TCPSIZE

To establish the CONNX JDBC Server TCP/IP packet size, set the following value:

```
TCPSIZE =32000
```

Controls the size of message buffers between CONNX32.dll and the database servers; acceptable values are 4096 to 32000.

Default value is 8192

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = JDBC; Key Value/Value Name = TCPSIZE

Unix Registry Setting: JDBC.TCPSIZE

TEMPPATH

To change the CONNX temporary directory for files created during query processing, set the following value:

```
TEMPPATH = c:\temp
```

If TEMPPATH is set, CONNX temporary files created during query processing will be placed in this directory.

If TEMPPATH is not set, temporary files will be placed in either (Windows) the directory indicated by the TEMP environment variable, or (Unix) the CONNX install directory.

Default values are:

- Windows: The directory indicated by the TEMP environment variable.
- Unix: The CONNX install directory.

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX ; Key Value/Value Name = TEMPPATH

Unix Registry Setting: CONNX .TEMPPATH

TEXTDBMS

Enables the dynamic compression/uncompression of specialized record layouts with variable length fields. Enabling this setting works in conjunction with the TextDBMS data type for variable length fields.

```
TEXTDBMS = 1
```

Do not enable this setting unless instructed to do so by CONNX Technical Support.

Default = 0.

Environments: Server
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = TEXTDBMS

Unix Environment Variable: TEXTDBMS
VMS Logical: TEXTDBMS
Started Task: TEXTDBMS
CICS: TEXTDBMS

TIMEOUT

To set the TCP/IP timeout (the number of milliseconds that CONNX waits for a response) set the following value:

```
TIMEOUT=value
```

Replace value with the number of milliseconds to wait for a response from the CONNX server component. The default value is 3600000 (1 hour).

Environment: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = TIMEOUT
Unix Registry Setting: CONNX.TIMEOUT

TWOPHASECOMMIT

To establish two-phase commit when connecting to multiple Rdb databases located on the same physical VMS server, set the following value:

```
TWOPHASECOMMIT=value
```

Default = 1.

Environment: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX\RDB; Key Value/Value Name = TWOPHASECOMMIT

Unix Registry Setting: CONNX.RDB.TWOPHASECOMMIT

UPPERCASEONLY

To force column and table names to appear in uppercase upon import, set the following value:

```
UPPERCASEONLY=1
```

Forces column and table names to appear in uppercase upon import.

Default = 0.

Environment: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNXCDD\OPTIONS; Key Value/Value Name = UPPERCASEONLY

Unix Registry Setting: CONNXCDD.OPTIONS.UPPERCASEONLY

USE32KRECORDBUFFER

For Adabas on Open Systems (Unix/Linux and Windows), this setting forces the use of a 32K record buffer rather than the native 64K record buffer. This setting may be useful when connecting to an Adabas database that is running on a non Open Systems platform such as zOS, VSE or BS2000 via Entire Network running on an Open Systems platform. In this situation, CONNX detects that it is running on an Open Systems platform and attempts to use a 64K record buffer which will create an error condition if the database itself does not support 64K record buffers. Setting USE32KRECORDBUFFER to a value of 1 will force CONNX to use a 32K record buffer.

USE32KRECORDBUFFER = 1

Default = 0.

Note:

Setting USE32KRECORDBUFFER to 1 will force the use of a 32K record buffer for all connections regardless of whether or not the connection is being made to a non Open Systems platform.

Environments: Server,Windows, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = USE32KRECORDBUFFER

Unix Environment Variable: USE32KRECORDBUFFER

USECONNXSCHEMAFORNATIVE

To create tables in relational databases (such as SQL Server, Oracle, DB2, Sybase, and Informix) using the current CONNX schema (instead of the schema specified at import time), set the following value:

```
USECONNXSCHEMAFORNATIVE =1
```

When USECONNXSCHEMAFORNATIVE = 0, relational database tables are created using the schema specified at import time.

When USECONNXSCHEMAFORNATIVE = 1, relational database tables are created using the current CONNX schema.

NOTE: If the current CONNX schema name does not exist in the database, the specified schema must be valid on the target database. Otherwise, an error can occur on the Create Table command.

Default = 0

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = USECONNXSCHEMAFORNATIVE

Unix Configuration Setting: USECONNXSCHEMAFORNATIVE

USEDOUBLEFORNUMBER

To convert all Oracle Number (Numeric) data types to Double (instead of Decimal), set the following value:

```
USEDOUBLEFORNUMBER = 1
```

This setting only applies to Oracle.

Default = 0 (Function enabled).

Environments: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = USEDOUBLEFORNUMBER
Unix Registry Setting: CONNX.USEDOUBLEFORNUMBER

USEOLDADABASDECIMALDATATYPE

To map Adabas Unpacked Numeric fields to DECIMAL rather than SQL Numeric, set the following value:

```
USEOLDADABASDECIMALDATATYPE = 1
```

This setting only applies to Adabas.

In older versions of CONNX, the Adabas type Unpacked Numeric was mapped to decimal. In current versions, Adabas Unpacked Numeric is mapped to SQL Numeric. In most situations, this change in mapping will not affect the calling application. Set this value to 1 if you need to retain the older mapping.

Default = 0

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = USEOLDADABASDECIMALDATATYPE

Unix Registry Setting: CONNX.USEOLDADABASDECIMALDATATYPE

USEREXITFLAGS

This flag will determine when the Adabas user exit (if defined) is called.

```
enum CNXUSEREXITFLAGS
{
  CNX_USEREXIT_NONE = 0x00,
  CNX_USEREXIT_PRECALL = 0x01,
  CNX_USEREXIT_POSTCALL= 0x02
};
```

A value of 1 means the exit will be called before every adabas call.
A value of 2 means the exit will be called after every adabas call.
A value of 3 means the exit will be called before and after every adabas call.

Please see USEREXITNAME32 and USEREXITNAME64 for more details.

Default = 0

Environments: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX\ADABAS; Key Value/Value Name = USEREXITFLAGS

Unix Environment Variable : USEREXITFLAGS

USEREXITNAME32

The 32 bit CONNX Data Server user exit is currently available only for Adabas and C-ISAM.

For example -

On windows:

USEREXITNAME64 = e:\mydir\myuserexit.dll

On unix:

USEREXITNAME64 = /home/mydir/myuserexit.sl

The specified load module must export a public "C" function called "UserExitFunc"

The function must have the following signature:

For Adabas:

```
int  UserExitFunc(unsigned int threadid,
const char *userid,
const char * clientid,
void * AdabasControlBlock,
void * szFormatBuffer,
void * szRecordBuffer,
void *szSearchBuffer,
void *szValueBuffer,
void *szISNBuffer,
int nPrePost );
```

For C-ISAM:

```
enum CNXUSEREXITFLAGS
{
  CNX_USEREXIT_NONE = 0x00,
  CNX_USEREXIT_PRECALL = 0x01,
  CNX_USEREXIT_POSTCALL= 0x02
};
enum CNXUSEROPERATIONS
{
  CNX_USEREXIT_UPDATE = 0x01,
  CNX_USEREXIT_DELETE = 0x02,
  CNX_USEREXIT_INSERT = 0x04
};

int UserExitFunc(const char *userid,
const char* szFileName,
CNXUSEROPERATIONS nOperation,
CNXUSEREXITFLAGS flags,
const void * uniqueID,
```

```
const void * recordnumber,
const void * szRecord,
int nRecordLength);
```

For Adabas the exit will be called either pre or post direct call based on the setting of the USEREXITFLAGS setting.

For C-ISAM

CONNX will call this function under the following circumstances:

1) When a record is inserted, the function is called after the insert has been successful.

2) When a record is updated, the function is called twice, once before the update, once after the update.

3) When a record is deleted, the function is called once after the delete is successful

Sample exit for Adabas:

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#pragma pack(1)
typedef struct _ADABASCONTROLBLOCK
{
char cb_cmd_res[2]; // Reserved
char cb_cmd_code[2]; // Command Code
char cb_cmd_id[4]; // COMMAND ID
#if defined (CNX_LITTLE_ENDIAN)
char cb_file_nr; // File#
char cb_db_id; // Database Number
#else
char cb_db_id; // Database Number
char cb_file_nr; // File#
#endif
   //short cb_file_nr;
short cb_return_code;
int cb_isn;
int cb_isn_l1;
int cb_isn_quantity;
unsigned short cb_fmt_buf_lng;
unsigned short cb_rec_buf_lng;
unsigned short cb_sea_buf_lng;
unsigned short cb_val_buf_lng;
unsigned short cb_isn_buf_lng;
char cb_cop1;
char cb_cop2;
```

```
char cb_add1[8];
char cb_add2[4];
char cb_add3[8];
char cb_add4[8];
char cb_add5[8];
int cb_cmd_time;
char cb_user_area[4];
//
} ADABASCONTROLBLOCK;
#pragma pack()
// This is an example of an exported function.
extern "C" __declspec(dllexport) int UserExitFunc(unsigned int threadid, const char *userid, const char * clientid, void * AdabasControlBlock, void * szFormatBuffer, void * szRecordBuffer,  void *szSearchBuffer, void *szValueBuffer, void *szISNBuffer, int nPrePost )
{
        FILE * fp;
        fp = fopen("c:\\adaexit.log", "a+");
        if (fp)
                {
                ADABASCONTROLBLOCK * pBlock = (ADABASCONTROLBLOCK *)AdabasControlBlock;
                fprintf(fp, "thread (%d), user(%s), clientid(%s), adabas command(%2.2s)\n", threadid, userid, clientid, pBlock->cb_cmd_code);
                fclose(fp);
                }

        return 0;
}
```

Sample exit for CISAM:

```
#include "stdafx.h"
#include "stdio.h"
extern "C" __declspec(dllexport) int UserExitFunc(const char *userid, const char* szFileName, int nOperation, int flags, const void * uniqueID, const void * recordnumber, const void * szRecord, int nRecordLength)
{
FILE * fp;
fp = fopen("c:\\cisamexit.log", "a+");
if (fp)
        {
        fprintf(fp, "user(%s), filename(%s), operation(%d), flag(%d) record(%-*.*s)\n", userid, szFileName, nOperation, flags,  nRecordLength, nRecordLength, szRecord);
        fclose(fp);
```

```
        }
    return 0;
    }
```

Default = blank

Environments: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNX\ADABAS or CONNX\CISAM; Key Value/Value Name = USEREXITNAME32
Unix Environment Variable : USEREXITNAME32

USEREXITNAME64

The 64bit CONNX Data Server user exit is currently available only for Adabas and C-ISAM.

For example -

On windows:

USEREXITNAME64 = e:\mydir\myuserexit.dll

On unix:

USEREXITNAME64 = /home/mydir/myuserexit.sl

The specified load module must export a public "C" function called "UserExitFunc"

The function must have the following signature:

For Adabas:

```
int  UserExitFunc(unsigned int threadid,
const char *userid,
const char * clientid,
void * AdabasControlBlock,
void * szFormatBuffer,
void * szRecordBuffer,
void *szSearchBuffer,
void *szValueBuffer,
void *szISNBuffer,
int nPrePost );
```

For C-ISAM:

```
enum CNXUSEREXITFLAGS
{
  CNX_USEREXIT_NONE = 0x00,
  CNX_USEREXIT_PRECALL = 0x01,
  CNX_USEREXIT_POSTCALL= 0x02
};
enum CNXUSEROPERATIONS
{
  CNX_USEREXIT_UPDATE = 0x01,
  CNX_USEREXIT_DELETE = 0x02,
  CNX_USEREXIT_INSERT = 0x04
};

int UserExitFunc(const char *userid,
const char* szFileName,
CNXUSEROPERATIONS nOperation,
CNXUSEREXITFLAGS flags,
```

```
const void * uniqueID,
const void * recordnumber,
const void * szRecord,
int nRecordLength);
```

For Adabas the exit will be called either pre or post direct call based on the setting of the USEREXITFLAGS setting.

For C-ISAM

CONNX will call this function under the following circumstances:

1) When a record is inserted, the function is called after the insert has been successful.

2) When a record is updated, the function is called twice, once before the update, once after the update.

3) When a record is deleted, the function is called once after the delete is successful

Sample exit for Adabas:

```
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#pragma pack(1)
typedef struct _ADABASCONTROLBLOCK
{
char cb_cmd_res[2]; // Reserved
char cb_cmd_code[2]; // Command Code
char cb_cmd_id[4]; // COMMAND ID
#if defined (CNX_LITTLE_ENDIAN)
char cb_file_nr; // File#
char cb_db_id; // Database Number
#else
char cb_db_id; // Database Number
char cb_file_nr; // File#
#endif
   //short cb_file_nr;
short cb_return_code;
int cb_isn;
int cb_isn_l1;
int cb_isn_quantity;
unsigned short cb_fmt_buf_lng;
unsigned short cb_rec_buf_lng;
unsigned short cb_sea_buf_lng;
unsigned short cb_val_buf_lng;
unsigned short cb_isn_buf_lng;
char cb_cop1;
```

```
char cb_cop2;
char cb_add1[8];
char cb_add2[4];
char cb_add3[8];
char cb_add4[8];
char cb_add5[8];
int cb_cmd_time;
char cb_user_area[4];
//
} ADABASCONTROLBLOCK;
#pragma pack()
// This is an example of an exported function.
extern "C" __declspec(dllexport) int UserExitFunc(unsigned int threadid, const char *userid, const
char * clientid, void * AdabasControlBlock, void * szFormatBuffer, void * szRecordBuffer,  void
*szSearchBuffer, void *szValueBuffer, void *szISNBuffer, int nPrePost )
{
        FILE * fp;
        fp = fopen("c:\\adaexit.log", "a+");
        if (fp)
                {
                ADABASCONTROLBLOCK * pBlock = (ADABASCONTROLBLOCK
                *)AdabasControlBlock;
                fprintf(fp, "thread (%d), user(%s), clientid(%s), adabas command(%2.2s)\n", threadid,
                userid, clientid, pBlock->cb_cmd_code);
                fclose(fp);
                }

        return 0;
}
```

Sample exit for CISAM:

```
#include "stdafx.h"
#include "stdio.h"
extern "C" __declspec(dllexport) int UserExitFunc(const char *userid, const char* szFileName, int
nOperation, int flags, const void * uniqueID, const void * recordnumber, const void * szRecord, int
nRecordLength)
{
FILE * fp;
fp = fopen("c:\\cisamexit.log", "a+");
if (fp)
        {
        fprintf(fp, "user(%s), filename(%s), operation(%d), flag(%d) record(%-*.*s)\n", userid,
        szFileName, nOperation, flags,  nRecordLength, nRecordLength, szRecord);
```

```
        fclose(fp);
        }
return 0;
}
```

Default = blank

Environments: Client, Windows, Unix
Configuration Manager: CONNX Settings; Current Key = CONNX\ADABAS or CONNX\CISAM; Key Value/Value Name = USEREXITNAME64
Unix Environment Variable : USEREXITNAME64

USERID

The default userid that will populate the logon dialog when making a connection if no user is specified.

Environments: Client, Windows CONNX

Configuration Manager: CONNX Settings; Current Key = CONNX/Defaults; Key Value/Value Name = USERID

USESXCALL

For Adabas, disables the use of search SX type adabas commands when set to zero. This setting may be useful in preventing the overflow of adabas WORK area for large databases, but will result in slower performance if set to zero.

USESXCALL = 0

Default = 1.

Environments: Client, Server,Windows, Mainframe, Unix, Adabas

Configuration Manager: CONNX Settings; Current Key = CONNX\Adabas; Key Value/Value Name = USESXCALL

Unix Environment Variable: USESXCALL

Started Task:USESXCALL

CICS: USESXCALL

VMSServerType

This setting is used by the CONNX Installation utility and does not need to be edited. This specifies the type of VMS server selected during the install (VAX, Alpha, Itanium).

YEARWINDOW

To configure the sliding date window for Year 2000 dates and help convert non-Y2K legacy date data, set the following values:

YEARWINDOW=nn

Enter the desired year-window where nn represents the dividing year between centuries.

Interpretation using the default value of 40 years:

Dates nn/nn/59 to nn/nn/99 are seen as 1959 to 1999.

Dates nn/nn/00 to nn/nn/58 are seen as 2000 to 2058.

The YearWindow setting affects the data types shown in the following table:

Sliding Date Window Data Types

CONNX Data Type	SQL Data Type	Length	Description
Text Date 2000 (YYMMDD)	DATE	6	Text Date in the specified format. Handles years 1941 to 2040.
Text Date 2000 (MMDDYY)	DATE	6	Text Date in the specified format. Handles years 1941 to 2040.
Text Date 2000 (DDMMYY)	DATE	6	Text Date in the specified format. Handles years 1941 to 2040.
Long Date 2000 (YYMMDD)	DATE	6	Long Date in the specified format. Handles years 1941 to 2040.
Long Date 2000 (MMDDYY)	DATE	6	Long Date in the specified format. Handles years 1941 to 2040.
Long Date 2000 (DDMMYY)	DATE	6	Long Date in the specified format. Handles years 1941 to 2040.
Packed Date 2000 (YYMMDD)	DATE	4	Packed Date in the specified format. Handles years 1941 to 2040.
Packed Date 2000 (MMDDYY)	DATE	4	Packed Date in the specified format. Handles years 1941 to 2040.
Packed Date 2000 (DDMMYY)	DATE	4	Packed Date in the specified format. Handles years 1941 to 2040.

Default = 40 years.

Environment: Client, Windows, Unix

Configuration Manager: CONNX Settings; Current Key = CONNX; Key Value/Value Name = YearWindow

Unix Registry Setting: CONNX.YEARWINDOW

Managing Windows Configuration Settings

Use the CONNX Configuration Manager to manage your Windows configuration settings.

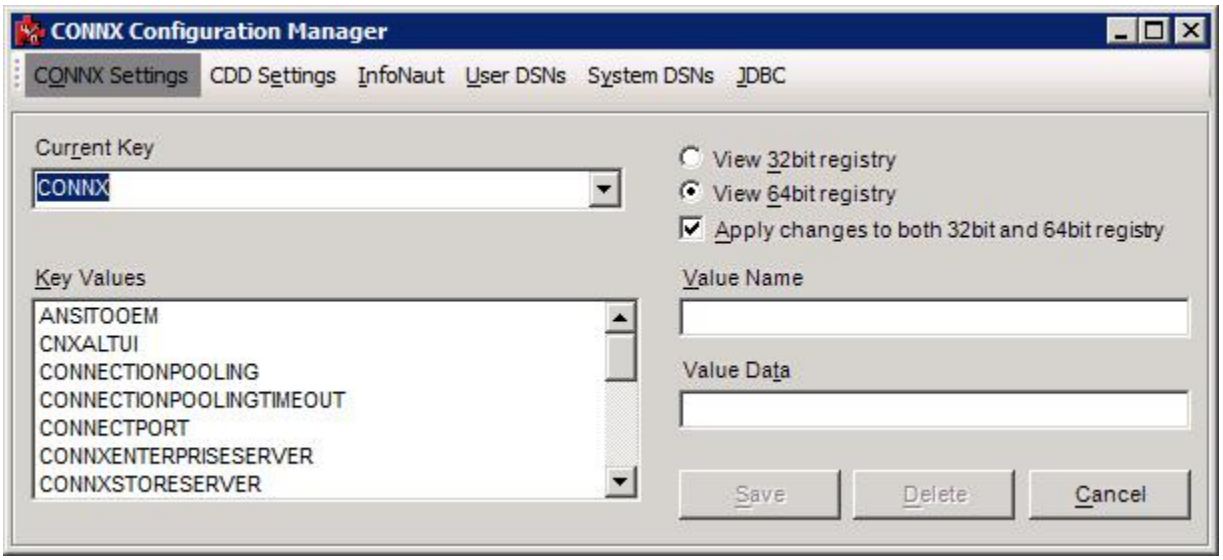
- Add Windows configuration settings
- Change Windows configuration settings
- Remove Windows configuration settings

To use the CONNX Configuration Manager to add new configuration settings

1. Click the **Start** button, and then point to **All Programs.** Point to **CONNX Driver** and then click **CONNX Configuration Manager.**

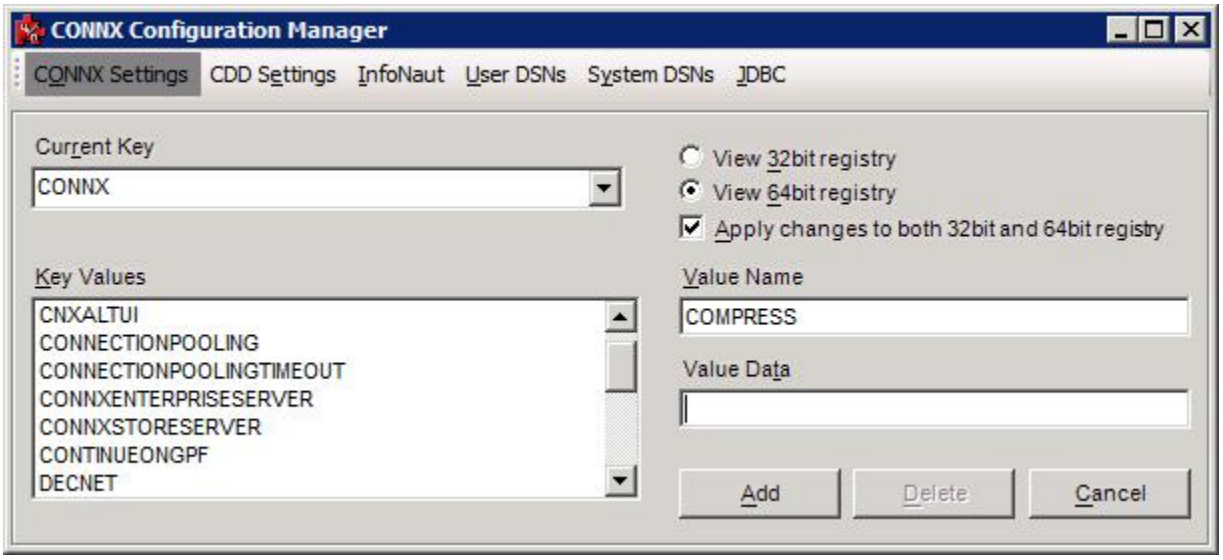
   A dialog box appears for users who do not have write access to the registry. Such users cannot modify any registry values. If you need to change registry settings, contact your network administrator.

2. The CONNX Configuration Manager window appears.



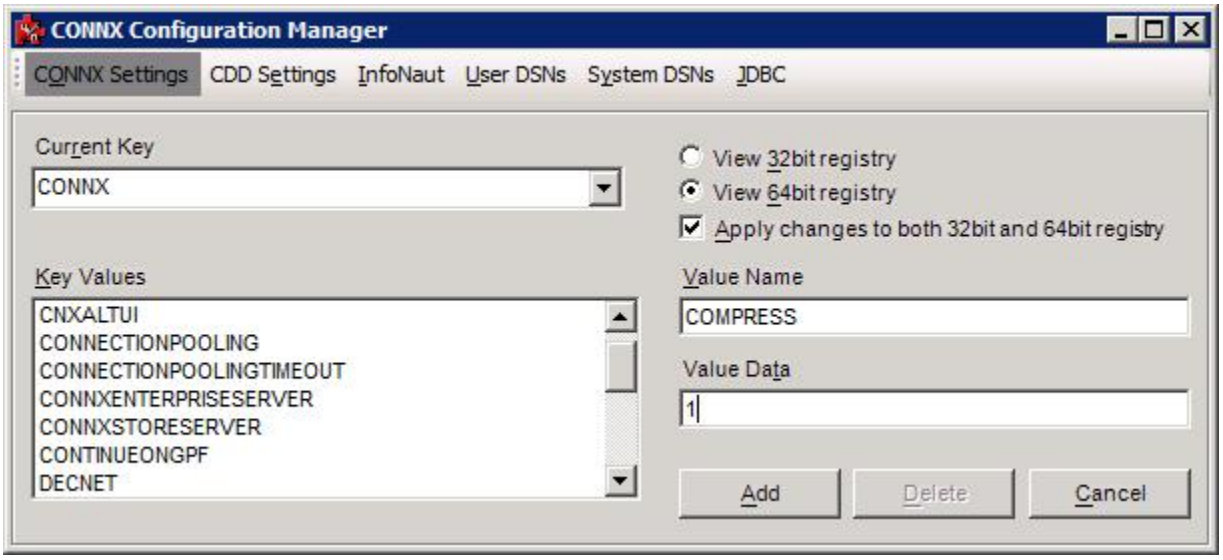
   The settings are for this machine running CONNX and are displayed on five tabs: CONNX Settings, CDD Settings, InfoNaut, User DSNs, and System DSNs. The **CONNX Settings** tab is selected by default.

3. Select a key location in the **Current Key** list box.



4. Under **Value Name,** type the value to add. We will use COMPRESS for our example. The **Save** button changes to the **Add** button.

5. Enter the value data in the **Value Data** text box. In this example, COMPRESS was added as a Value with the selected data entered as "1".



6. Click **Add** to add the new value to the registry.

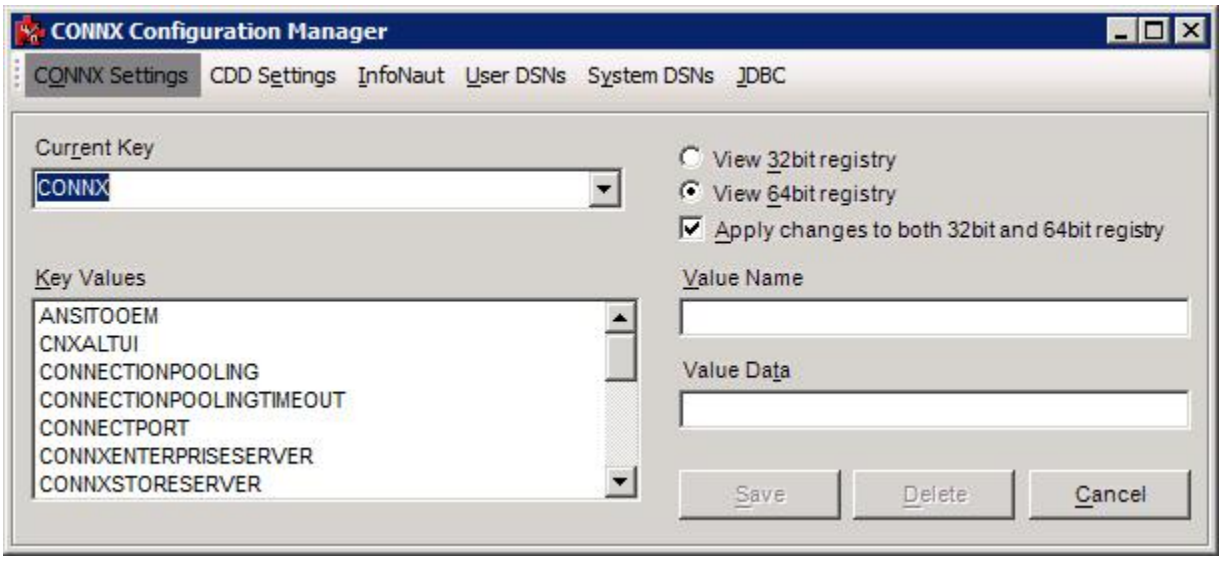
7. On a 64bit Windows operating system, the radio buttons to view 32bit or 64bit registry settings is displayed. If you wish the setting be applied to both the 32bit and 64bit registry, check the "Apply changes to both..." check box. If a setting should only be applied to one, select the appropriate radio button and uncheck the "Apply changes to both..." check box.

To use the CONNX Configuration Manager to change or update configuration settings

1. Click the **Start** button, and then point to **All Programs.** Point to **CONNX Driver** and then click **CONNX Configuration Manager.**

   A dialog box appears for users who do not have write access to the registry. Such users cannot modify any registry values. If you need to change registry settings, contact your network administrator.

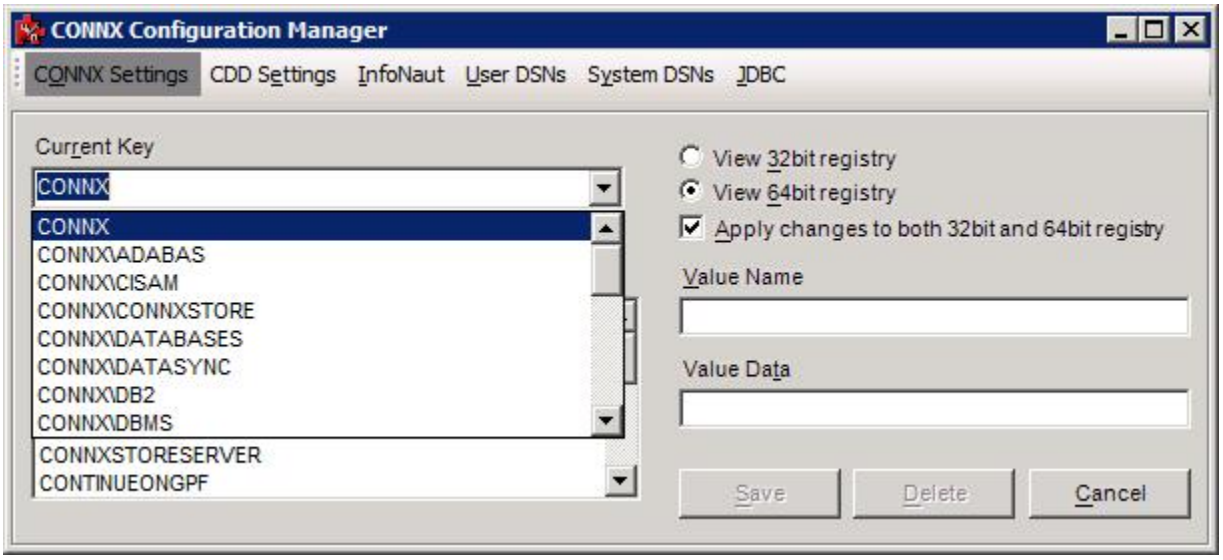
2. The CONNX Configuration Manager window appears.



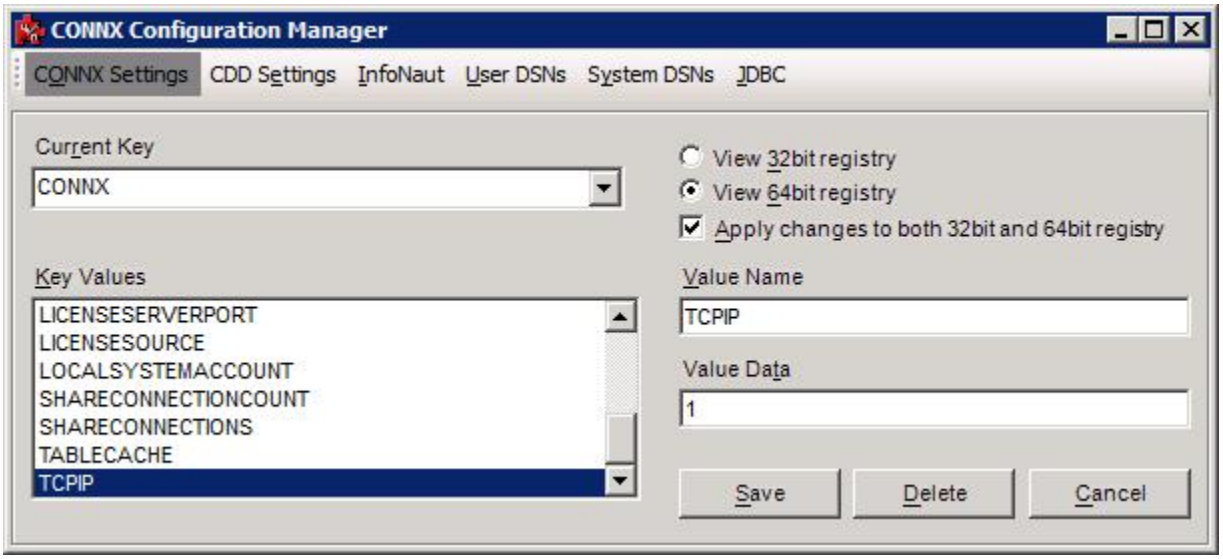
   The settings are for this machine running CONNX and are displayed on five tabs: CONNX Settings, CDD Settings, InfoNaut, User DSNs, and System DSNs. The **CONNX Settings** tab is selected by default.

   On 64bit Windows operating systems, the View 32bit registry and View 64bit registry radio buttons will be shown. These radio buttons allow different configuration settings for 32bit and 64bit versions of the CONNX driver, JDBC server, etc. **NOTE:** Most settings should be the same for both, but in some cases - such as path locations or port numbers - the values need to be different between the 32bit and 64bit version. In these cases, the "Apply changes to both..." check box should be unchecked when making changes.

3. Select the drop down list to change the current key.

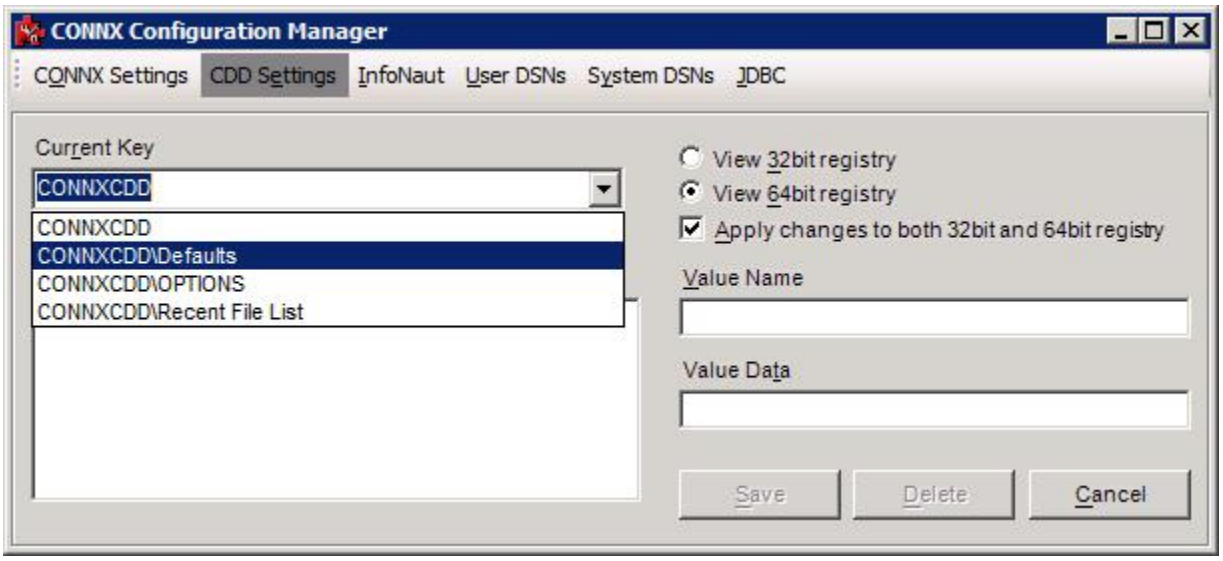


4. The Key Values for the selected key are displayed in the **Key Values** text box. Select a Value from the list to change. In this example, TCPIP is selected.



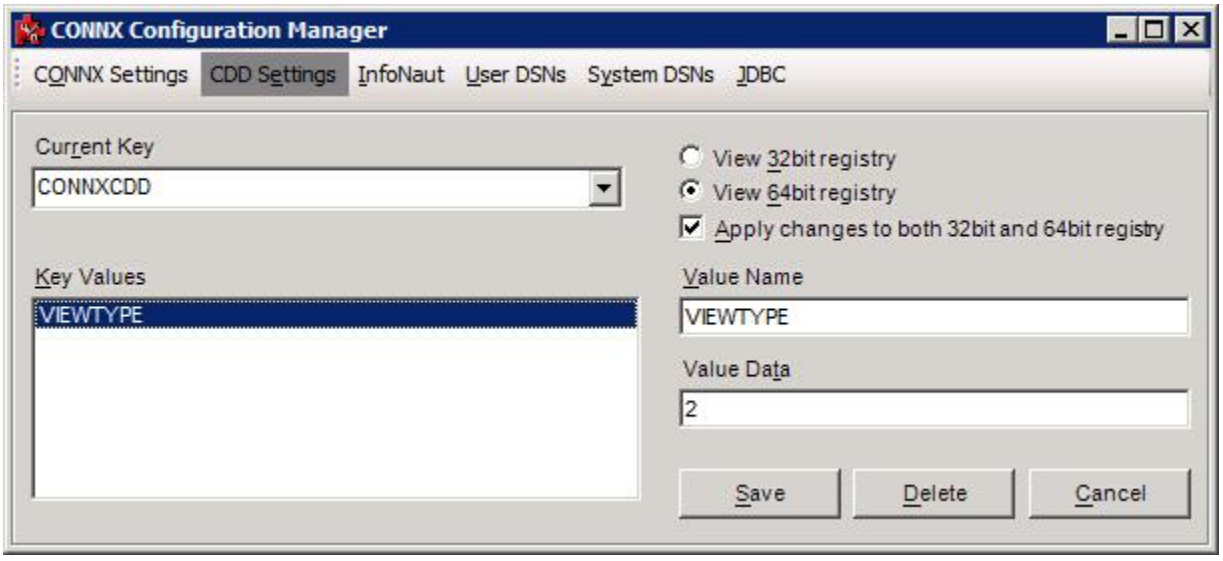
5. The data for the Value appears in the **Value Data** text box. The data stored in the Value can be modified by changing the contents of the **Value Data** text box. Once the data is changed, click on the **Save** button to store the change. See CONNX File Settings for more information on Value settings for each selected or available Value.

6. Select the **CDD Settings** tab for information on the CDD setting for the user machine. The default setting appears in the list box. See CONNX CDD Registry File Settings for more information on CDD Value settings.



Select the down arrow in the list box to view other options related to this computer.

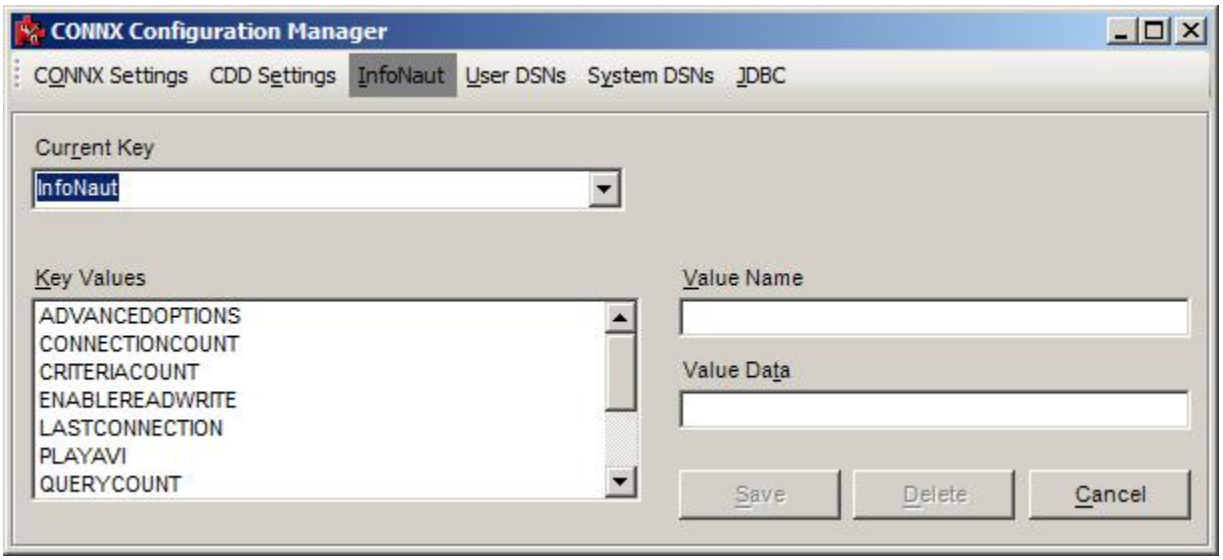
7. Select a Value in the **Key Value** list box to view the value name and value data. In this example, you can view the USERID and NODE Values.



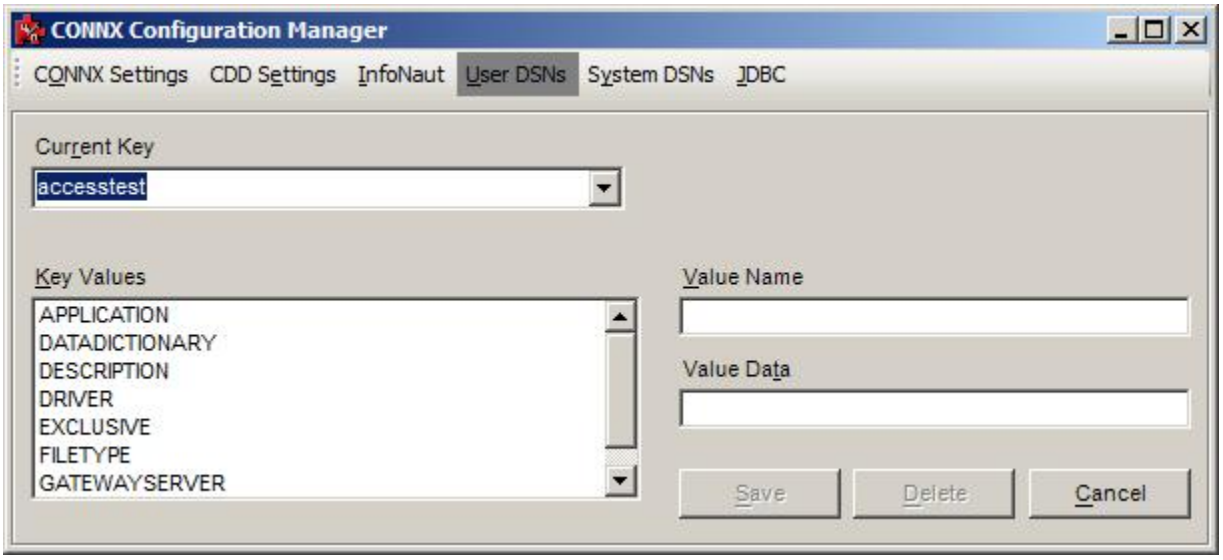
8. Type any desired changes in the **Value Data** text box and press the Save button..

9. Select the **InfoNaut** tab to view the InfoNaut registry keys available to this user on this machine. Registry configuration options for InfoNaut are listed in InfoNaut Registry Configuration Options in the InfoNaut Help File.

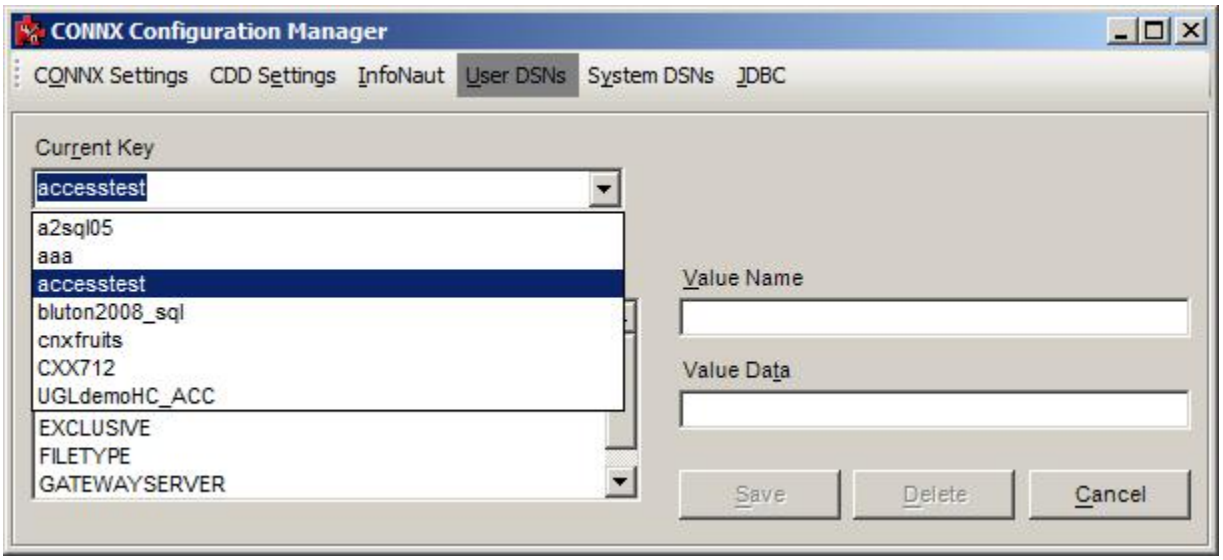
NOTE: InfoNaut settings are stored in the HKEY_CURRENT_USER area of the registry and therefore have no 32bit/64bit difference. All changes to InfoNaut settings apply to both the 32bit and 64bit versions of the product.



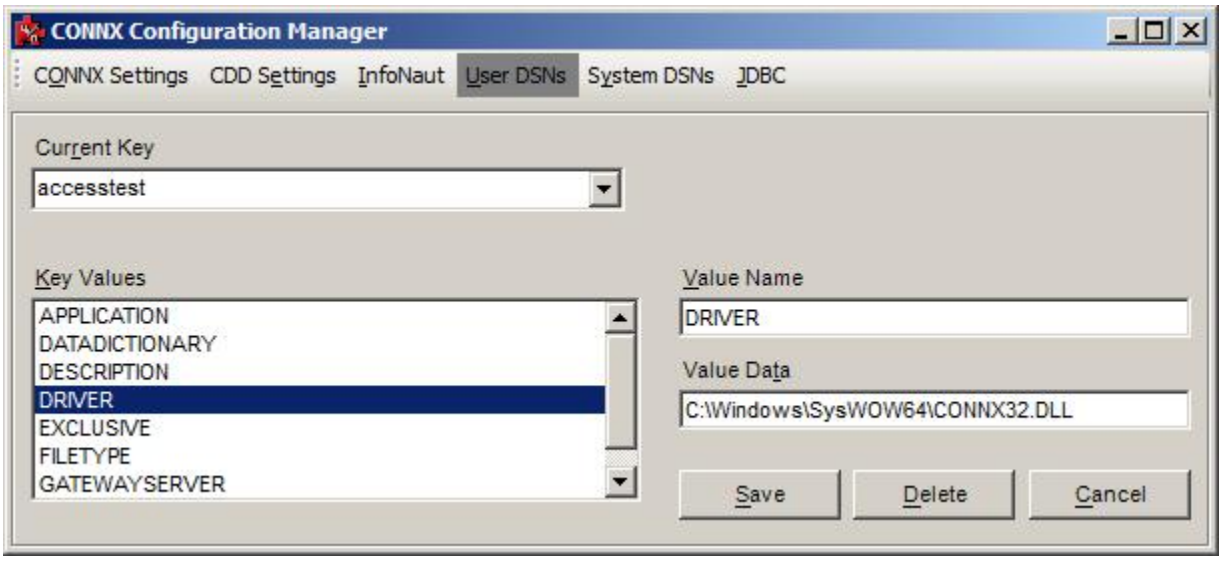
10. Select the **User DSNs** tab to view the CONNX User DSNs available to this user on this machine.



11. Select the arrow in the list box to view the entire list of CONNX User DSNs available to this user on this machine.

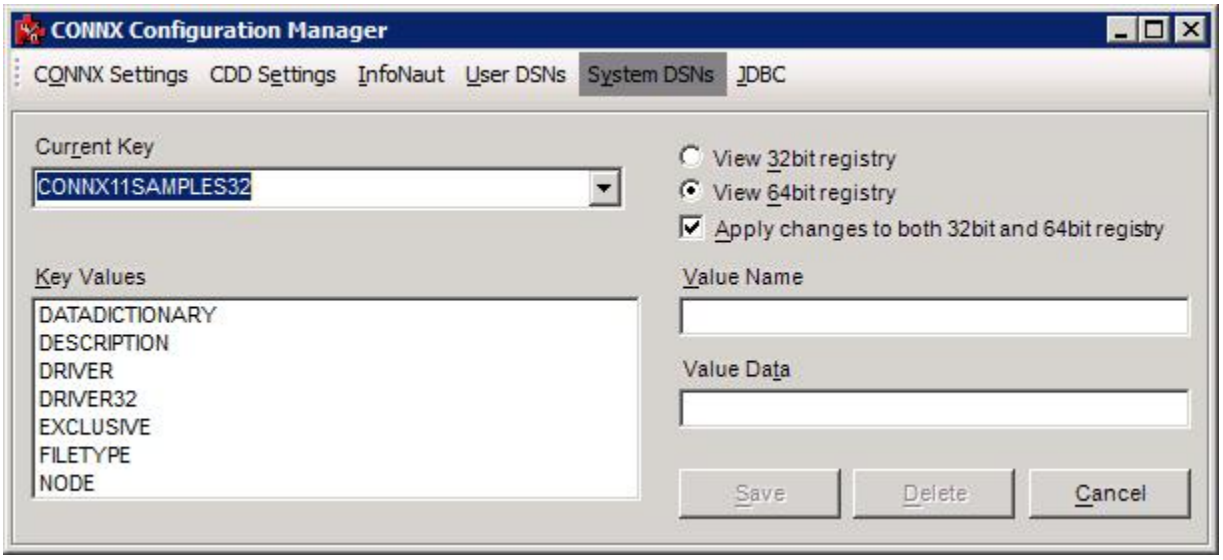


12. Select a **User DSN** containing the settings to alter or add, and then select a **Value** from the **Key Values** list box.



13. By selecting CONNX9SAMPLES32/DRIVER, you can view the location of the .DLL associated with the ODBC driver used for the CONNX sample databases. Select other Values to view related data.

14. Select the **System DSNs** tab to see the CONNX System DSNs available on this machine.

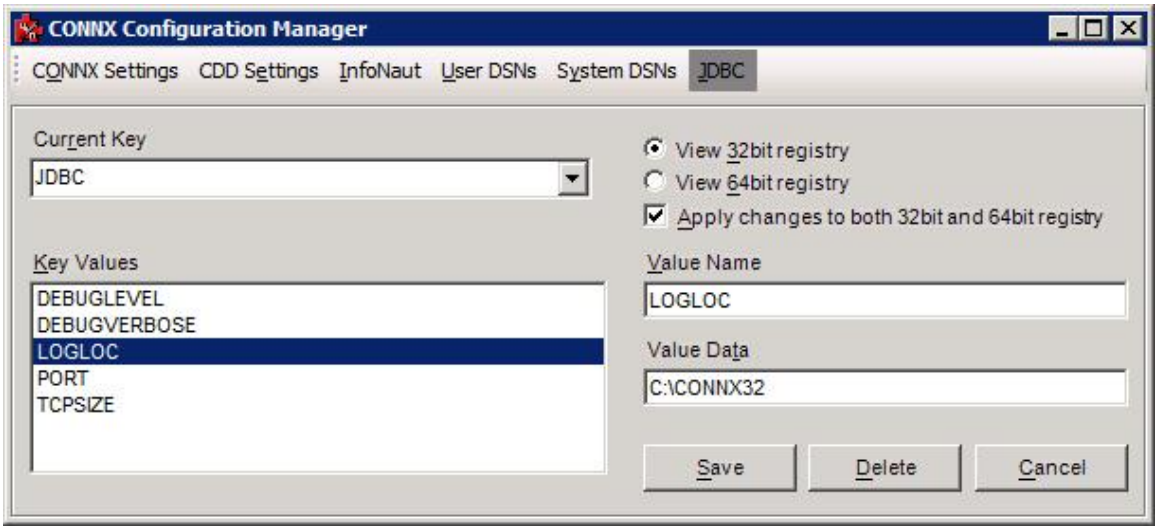


15. Select the arrow in the list box to view the entire list of CONNX System DSNs available on this machine.

16. Select a **System DSN** containing the settings to change, and then select a Value from the **Key Values** list box.

17. Change the selected Value data by entering new data in the **Value Data** text box.

18. Select the **JDBC** tab to see the CONNX System JDBC parameters available to this user on this machine. In this example, by selecting LOGLOC from the **Key Values** list box, you can change the location of the .log file.



21. Click the **Save** button to store these changes.

22. Click the **Close** button to close the application.

Note: *If no User or System DSNs exist on this machine, a dialog box appears stating that no CONNX data source names were found.*

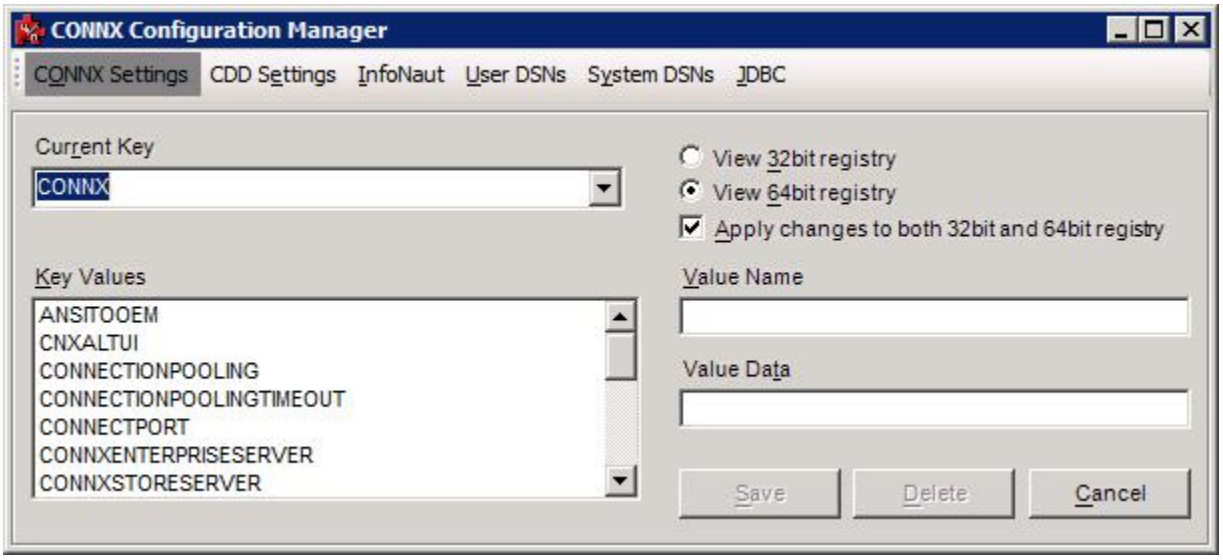
To use the CONNX Configuration Manager to remove file settings

1. Click the **Start** button, and then point to **All Programs.** Point to **CONNX Driver** and then click **CONNX Configuration Manager.**

   A dialog box appears for users who do not have write access to the registry. Such users cannot modify any registry values. If you need to change registry settings, contact your network administrator.

-

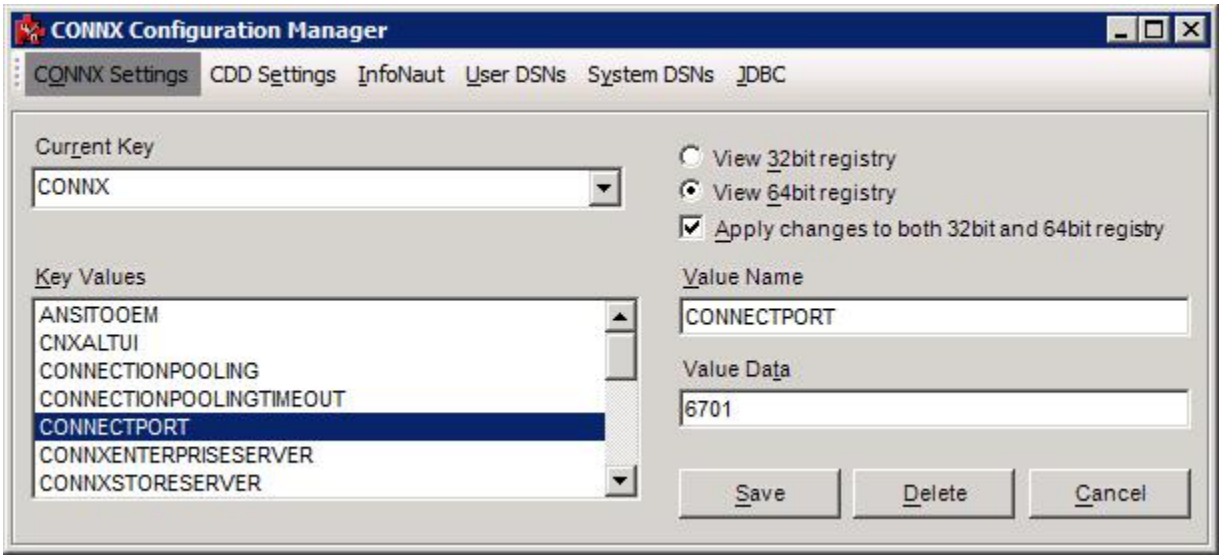
2. The CONNX Configuration Manager window appears.



The settings are for this machine running CONNX and are displayed on five tabs: CONNX Settings, CDD Settings, InfoNaut, User DSNs, and System DSNs. The **CONNX Settings** tab is selected by default.

On 64bit Windows operating systems, the View 32bit registry and View 64bit registry radio buttons will be shown. These radio buttons allow different configuration settings for 32bit and 64bit versions of the CONNX driver, JDBC server, etc. **NOTE:** Most settings should be the same for both, but in some cases - such as path locations or port numbers - the values need to be different between the 32bit and 64bit version. In these cases, the "Apply changes to both..." check box should be unchecked when making changes.

3. Select a Key location in the **Current Key** list box.



4. Select a Value to delete in the **Key Value** list box.
5. Click **Delete**. The Value is removed from the **Key Value** list box and the Registry.
6. Click **Close**.

CONNX SQLRegistry Program (Unix)

Configuring the CONNX Client and the JDBC Server

CONNX Client

When the CONNX Client is installed and run on Unix, the shared object (libconnx32) uses and retrieves the required settings from the connxreg.db file. SQLRegistry Program - update configuration settings contains information about specific registry settings.

JDBC Server

When the JDBC Server is installed and run on a non-Windows platform, however, the cnxjdbc.exe program retrieves the required JDBC settings and DSN registrations from connxreg.db file. SQLRegistry Program - update configuration settings contains information about specific registry settings.

SQLRegistry Program - update UNIX configuration settings

A SQLRegistry executable file (.exe) is installed on your machine when you install the CONNX client component on UNIX.

After the installation process is complete, export the CONNXREGISTRY environment variable (CONNXREGISTRY = /home/cnxuser/connx/connxclient/connxreg.db) into your UNIX environment. Each UNIX system may have a slightly different export method.

Note: The CONNXREGISTRY configuration setting (environment variable) is required and referenced by the CONNX client component and the JDBC Server program to access the registry file.

The SQLRegistry program is a command-line program. To run the program, proceed to the directory where the program is installed on your machine without passing an additional argument, for example, "/sqlregistry".

A menu is displayed:

1. Display registry keys and values
2. Create a registry key value
3. Update registry key value
4. Delete registry key value
5. Exit Program

You may also pass additional argument(s) when you run this program. To learn the usage, type "./sqlregistry help", the following messages will be displayed:

```
Usage ./sqlregistry: [Registry file,] (option) 1|2|3|4 , Key,
   (datatype) 0|1|2, Value

option:    1=display, 2=create, 3=update, 4=delete, 5=create_or_update

datatype:  0=integer, 1=string, 2=binary
```

For example:

To display the settings stored in the registry file, type:

```
"./sqlregistry display" or "./sqlregistry 1"
```

To insert/update a string setting, use the following sequence:

```
./sqlregistry update CONNX.DSNS.mYDSN string
   "/home/cnxuser/connx/connxclient.sample.cdd; My DSN connections;"
```

Links to Data Server Configuration Settings

Links to all the CONNX Data Server Configuration settings topics, in alphabetical order, can be found here.

"A" Configuration Settings

ADA_DEBUG_TRACE_MASK
ADA_LOCKDONTWAIT
ADA_NOQUALIFYBINARY
ADA_RESPECT_ISN_ON_INSERT
ADA_WAITTIME
ALLOWMIXEDPWD

"C" Configuration Settings

CNXBARNARD
CNXBATCHBUFFER
CNXCONNECTBACK
CNXDECNETTASK
CNXDIR
CNXHASH
CNXKBAUTHORIZE
CNXLOCALIP
CNXLISTENER
CNX_LIBRARY_PATH
CNXMUALPHA
CNXNOPREAUTHORIZE
CNXNOPOST
CNXNOQIO
CNX_NO_TIMER
CNXOLDLICENSE
CNX_PASS_TICKETS
CNXPOSTSERVERNAME
CNXRUNPORT

"D" Configuration Settings

"E" Configuration Settings

"F" Configuration Settings

"H" Configuration Settings

"K" Configuration Settings

"L" Configuration Settings

"M" Configuration Settings

MAXROWSCOMPARED
MAXROWSFETCHED
MAXROWSFETCHEDFAIL
MAXSOCKET
MULTIFETCH

"O" Configuration Settings

OPATTRIBUTES
OPDONTCALL
OPEXF
OPEXU
OPMAXCMDID
OPMAXHOLD
OPMAXISN
OPNONACTIVITY
OPREADONLY
OPSEARCHTIME
OPTIMEOUT
ORACLEBULKMODEDISABLEFLAG

"R" Configuration Settings

REISSUEOP

"S" Configuration Settings

SBCCSID
SUPERDESCRIPTORASFIELD

"T" Configuration Settings

TEXTDBMS

"U" Configuration Settings

USESXCALL

How to set Mainframe Started Task Configuration Settings

To add, change or delete CONNX configuration settings/environmental variables parameters for mainframe started task jobs (VSAM, QSAM, ADABAS), change CNXPARMS.

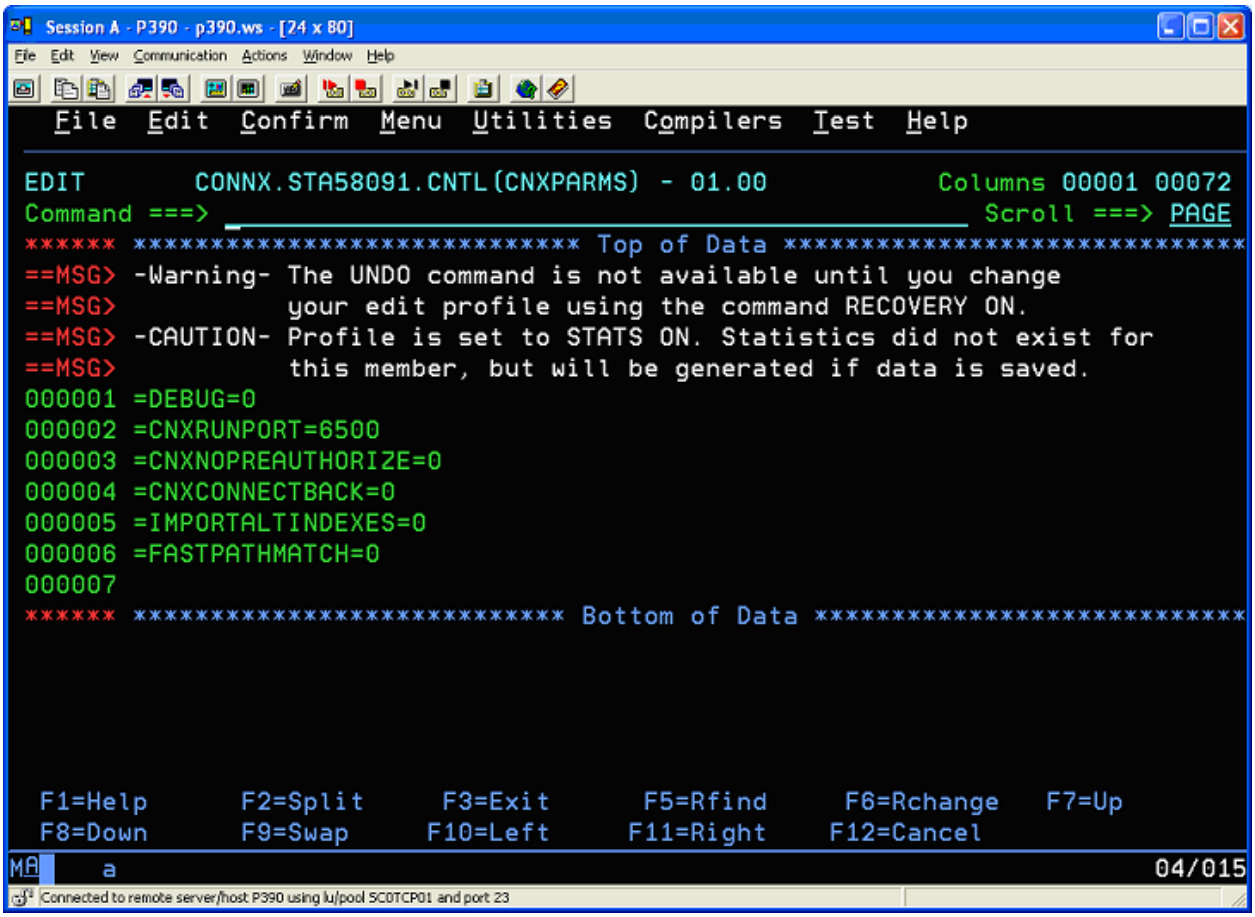
All settings are read once at either listener or server startup.

It always requires either a restart of the listener or connection of the new server for a new setting to take effect.

CNXPARMS is a member of the .CNTL dataset in the CONNX DSN HLQ that you specified when you installed CONNX to the mainframe

Example: If your CONNX DSN HLQ name is CONNX.TEST, you will find CNXPARMS in CONNX.TEST.CNTL.

When you install CONNX, a CNXPARMS member is created with some default values, as shown below:



To add a configuration setting or environmental variable, add it to CNXPARMS and use the format:

=PARAMETER=value

Example: To set ADA_DEBUG_TRACE_MASK, enter the following in CNXPARMS:
=ADA_DEBUG_TRACE_MASK=1

To remove a configuration setting/environmental variable, delete it from CNXPARMS.

Note: Any changes to CNXPARMS are not picked up until the started task job is stopped and then restarted.

CICS Configuration Instructions (Mainframe)

To add, change or delete CONNX configuration settings/environmental variables for mainframe CICS jobs (VSAM, CISAM, ADABAS), use the NX01 transaction.

All settings are read once at either listener or server startup.

It always requires either a restart of the listener or connection of the new server for a new setting to take effect.

The basic format of the command is:

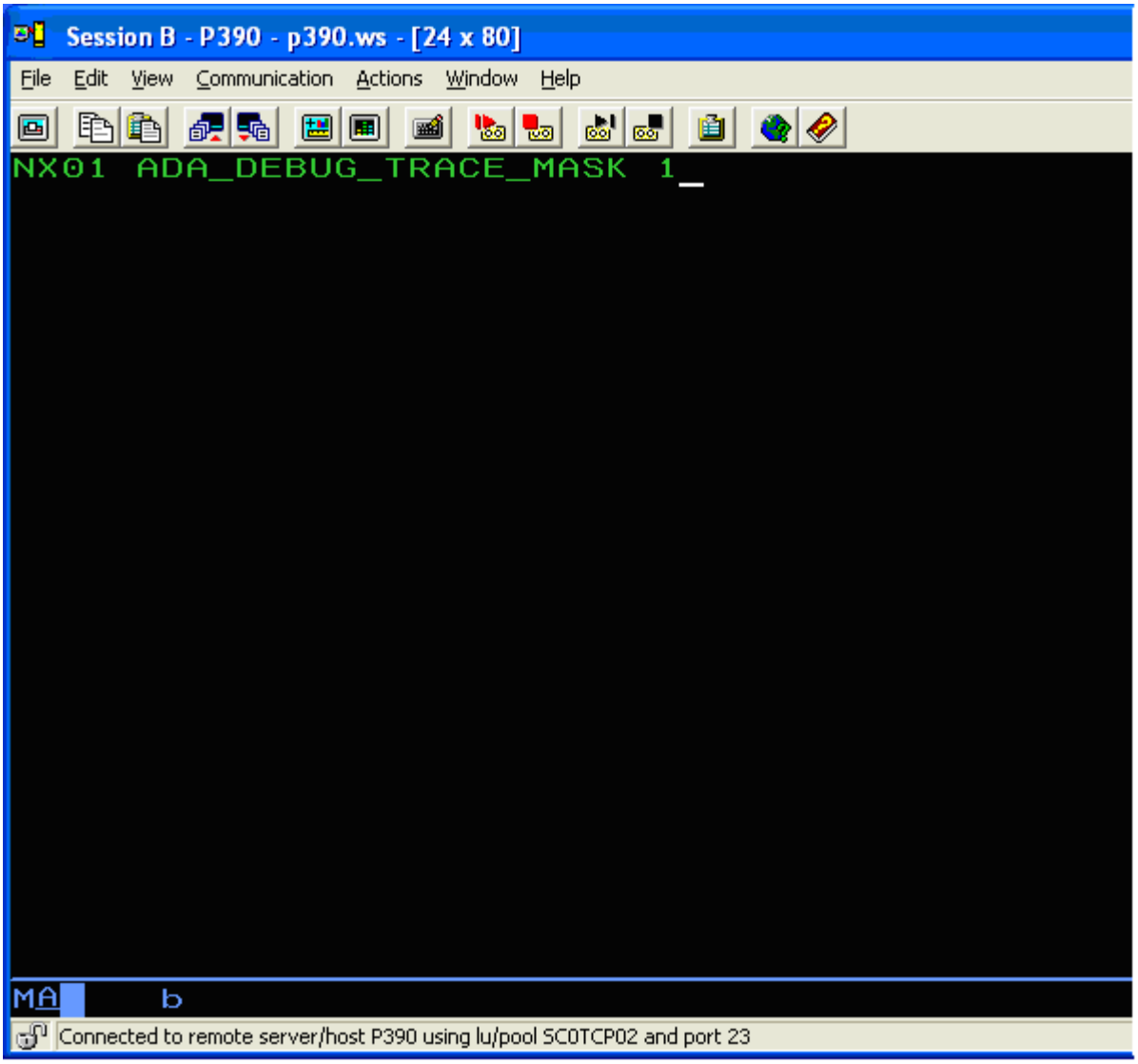
```
NX01 PARAMETER value
```

To remove a parameter and return to default behavior, enter the command without a value

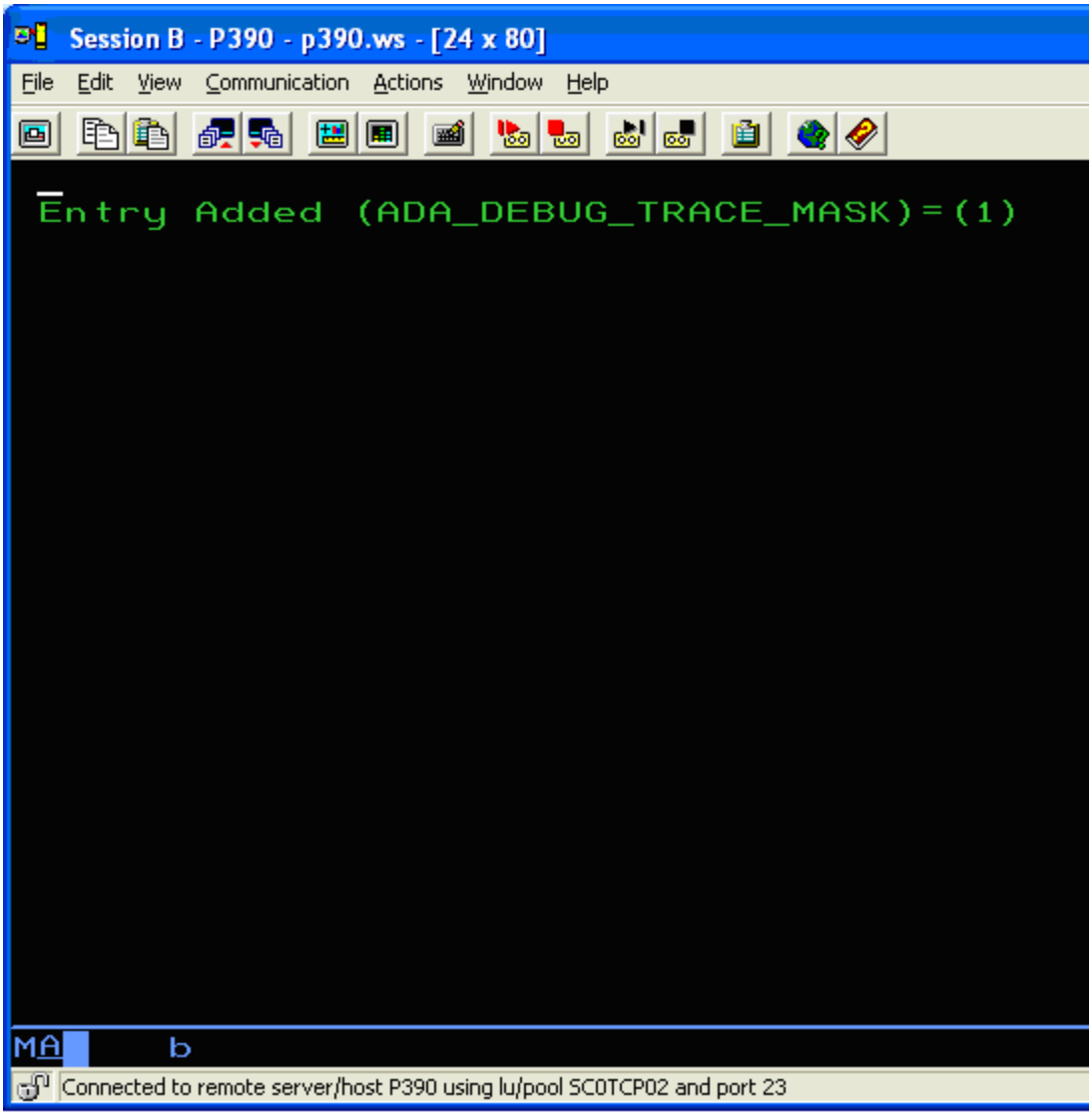
```
NX01 PARAMETER
```

Example: To set the environmental variable ADA_DEBUG_TRACE_MASK enter:

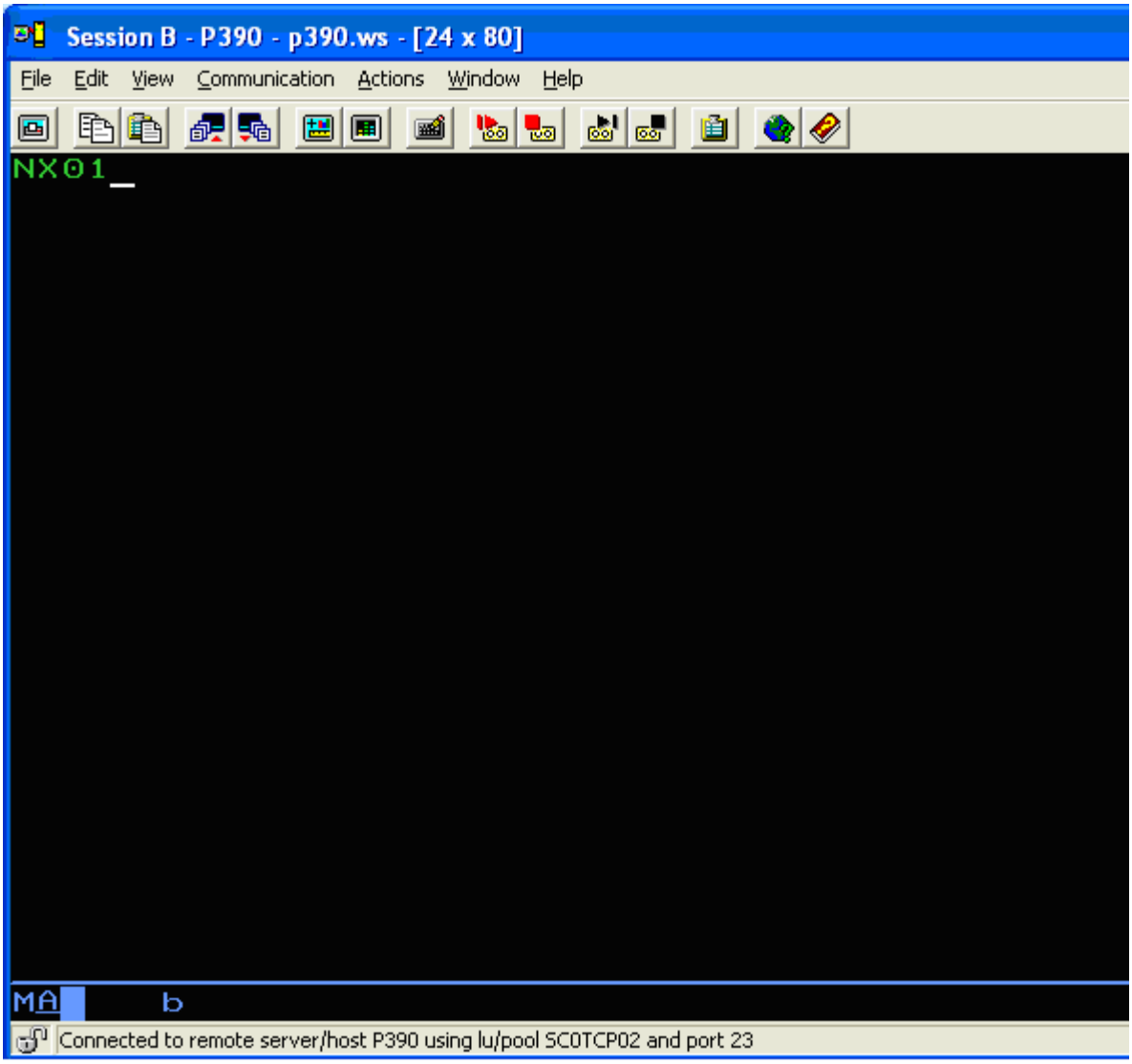
```
NX01 ADA_DEBUG_TRACE_MASK 1
```



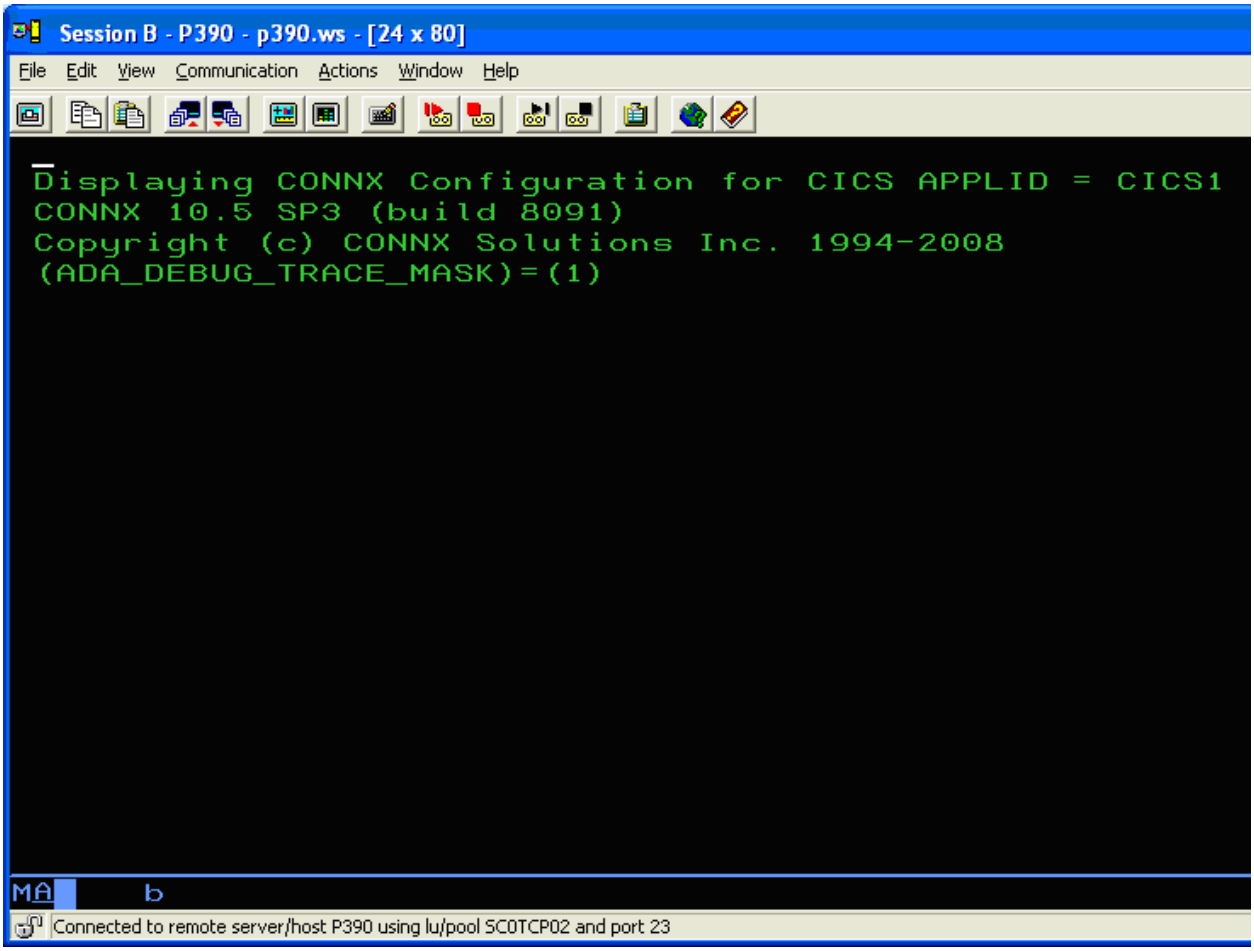
After the environmental variable has been entered the following appears:



Entering the NX01 transaction without any values allows you to view any CONNX configuration settings/environmental variables that have been set using NX01:



The variables that have been set by the NX01 transaction appear:


Session B - P390 - p390.ws - [24 x 80]
File Edit View Communication Actions Window Help
Displaying CONNX Configuration for CICS APPLID = CICS1
CONNX 10.5 SP3 (build 8091)
Copyright (c) CONNX Solutions Inc. 1994-2008
(ADA_DEBUG_TRACE_MASK)=(1)
MA b
Connected to remote server/host P390 using lu/pool SCOTCP02 and port 23

Schemas

An Information Schema provides metadata about a given database (catalog).

CONNX provides this capability through System Catalog tables in a CONNX CDD. System catalog tables represent the CDD tables/database/columns metadata (which include Information Schema metadata).

In CONNX, the System Catalog tables are in a single schema: CONNXSCHEMA.

You can access CONNXSCHEMA (the CONNX Catalog Tables) using SQL Query Statements. Your applications would use the CONNXSCHEMA system tables to determine the SQL statement metadata.

All of the schema tables can also be accessed using INFORMATION_SCHEMA.<schema table name>

Example: An application which wants to know the total length of a column so they would know how big to make a text box would issue a call similar to:

```
Select * from CONNXSCHEMA.dbo.COLUMNS where columnname='CUSTOMERNAME'.
or
Select * from INFORMATION_SCHEMA.COLUMNS where
   columnname='CUSTOMERNAME'.
```

Below is a list of the CONNXSCHEMA tables:

CONNXSCHEMA.dbo.BASE_TABLES
CONNXSCHEMA.dbo.CLUSTERS
CONNXSCHEMA.dbo.COLUMN_PRIVILEGES
CONNXSCHEMA.dbo.COLUMNS
CONNXSCHEMA.dbo.CONSTRAINT_COLUMN_USAGE
CONNXSCHEMA.dbo.CONSTRAINT_TABLE_USAGE
CONNXSCHEMA.dbo.DATABASES
CONNXSCHEMA.dbo.INFORMATION_SCHEMA_CATALOG_NAME
CONNXSCHEMA.dbo.KEY_COLUMN_USAGE
CONNXSCHEMA.dbo.NOT_NULL_CONSTRAINTS
CONNXSCHEMA.dbo.REFERENTIAL_CONSTRAINTS
CONNXSCHEMA.dbo.SCHEMATA
CONNXSCHEMA.dbo.SERVER_INFO
CONNXSCHEMA.dbo.syscnxColumnPrivleges
CONNXSCHEMA.dbo.syscnxColumns
CONNXSCHEMA.dbo.syscnxForeighKeys
CONNXSCHEMA.dbo.syscnxGetTypeInfo
CONNXSCHEMA.dbo.syscnxPrimaryKeys
CONNXSCHEMA.dbo.syscnxProcedureColumns
CONNXSCHEMA.dbo.syscnxProcedures
CONNXSCHEMA.dbo.syscnxSpecialColumns
CONNXSCHEMA.dbo.syscnxStatistics

CONNXSCHEMA.dbo.syscnxTablePrivileges
CONNXSCHEMA.dbo.syscnxTables
CONNXSCHEMA.dbo.TABLE_CONSTRAINTS
CONNXSCHEMA.dbo.TABLE_INDEXES
CONNXSCHEMA.dbo.TABLE PRIVILEGES
CONNXSCHEMA.dbo.TABLES
CONNXSCHEMA.dbo.USERS
CONNXSCHEMA.dbo.VIEW_COLUMN_USAGE
CONNXSCHEMA.dbo.VIEW_TABLE_USAGE
CONNXSCHEMA.dbo.VIEWS

If you want to see the CONNXSCHEMA tables, go to INFONAUT, and select the checkbox where it asks to see the system tables. This will allow you to view the table's contents.

View Description Tables

Table Notation

For each view, the columns are listed with their names, data types, indications, if NULL is possible, and short description. The data types are shown here as symbolic identifiers to describe a specific role:

identifier	CHAR(128).
yes_no	CHAR(3) with a content of "YES" or "NO".

Notes:

1. all undelimited identifiers in the catalog are represented in upper case characters.
2. "y" in the column "N" indicates that the column may contain NULL values.
3. "n" in the column "N" indicates that the column must not contain NULL values.

Base_Tables View

Lists the base tables defined in the catalog.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
TABLE_CATALOG	identifier	n	table catalog
TABLE_SCHEMA	identifier	n	table schema
TABLE_NAME	identifier	n	table name
DB_NR	integer	n	Adabas database number
FILE_NR	integer	n	Adabas file number
TABLE_LEVEL	integer	n	table level: 0, 1, 2

Clusters View

Describes the clusters contained in the catalog.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
CLUSTER_CATALOG	identifier	n	cluster catalog
CLUSTER_SCHEMA	identifier	n	cluster schema name
CLUSTER_NAME	identifier	n	cluster name
DB_NR	integer	n	Adabas database number
FILE_NR	integer	n	Adabas file number

Databases View

Identifies the database properties for each catalog in the data dictionary.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
DATABASE_NAME	identifier	n	database/catalog name
DATABASE_PHYSICAL_NAME	identifier	n	database physical description
DATABASE_TYPE	identifier	n	database type
PORT	integer	n	tcp/ip port number for the data server
ENTERPRISE_SERVER_SERVICE_ENABLED	integer	n	is this database configured to use the enterprise server service
DEFAULT_HOST_NAME	identifier	n	default server name for the database connection
REMARKS	identifier	n	comments

Server_Info View

Provides server information.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
SERVER_NAME	identifier	n	server name
NODE_NAME	identifier	n	database name
SERVER_VERSION	identifier	n	server version (currently set to client version)
CATALOG_VERSION	identifier	n	catalog version (currently unused)

Information_Schema_Catalog_Name View

This view is part of the ANSI SQL-2 standard information schema.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
CATALOG_NAME	identifier	n	"CONNXSCHEMA"

Schemata View

Describes the catalog schemas.

This view is part of the ANSI SQL-2 standard information schema.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
CATALOG_NAME	identifier	n	schema catalog
SCHEMA_NAME	identifier	n	schema name
SCHEMA_OWNER	identifier	n	schema owner
DEFAULT_CHARACTER_SET_CATALOG	identifier	n	default character set for catalog (currently unused)
DEFAULT_CHARACTER_SET_SCHEMA	identifier	n	default character set for schema (currently unused)
DEFAULT_CHARACTER_SET_NAME	identifier	n	default character set for name (currently unused)

Tables View

Lists the tables defined in the catalog.

This view is part of the ANSI SQL-2 standard Information Schema.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
TABLE_CATALOG	identifier	n	table catalog
TABLE_SCHEMA	identifier	n	table schema
TABLE_NAME	identifier	n	table name
TABLE_TYPE	enumeration	n	table type: Base Table or View

Views View

Lists the views defined in the catalog.

This view is part of the ANSI SQL-2 standard Information Schema.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
TABLE_CATALOG	identifier	n	table catalog
TABLE_SCHEMA	identifier	n	table schema
TABLE_NAME	identifier	n	table name
VIEW_DEFINITION	char(4000)	n	The SQL source string of the view definition.
CHECK_OPTION	enumeration	n	Cascade
IS_UPDATABLE	yes_no	n	is this field updatable?
SCHEMA_CONTEXT	identifier	n	dbo

Columns View

Describes the columns of the tables defined in the catalog.

This view is part of the ANSI SQL-2 standard Information Schema.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
TABLE_CATALOG	identifier	n	table catalog
TABLE_SCHEMA	identifier	n	table schema
TABLE_NAME	identifier	n	table name
COLUMN_NAME	identifier	n	column name
ORDINAL_POSITION	integer	n	ordinal position inside the containing table.
COLUMN_DEFAULT	long_alpha	y	default option in character representation.
IS_NULLABLE	yes_no	n	Y if Null is possible, N if not possible.
DATA_TYPE_NUMBER	short	n	data type (in integer form)
DATA_TYPE	enumeration	n	data type (in text form)
CHARACTER_MAXIMUM_LENGTH	long	n	physical length of data in characters.
CHARACTER_OCTET_LENGTH	long	n	physical length of data in bytes.
NUMERIC_PRECISION	long	n	precision of the data.
NUMERIC_PRECISION_RADIX	short	n	Radix of the precision. Radix is 2 for floating point data and 10 for integral data.
NUMERIC_SCALE	short	n	scale of the data.
DATETIME_PRECISION	short	n	precision of the timestamp
SHORT_NAME	char(2)	y	Adabas short name
MU_POS	short	y	Position of the column within an Adabas MU field.
COLUMN_LEVEL	short	n	0 = non MU/PE column, 1 = top level MU or PE, 2 = MU within PE
COLUMN_TYPE	enumeration	n	always returns "Ordinary"
CHARACTER_SET_CATALOG	identifier	n	character set of the catalog (currently not used)
CHARACTER_SET_SCHEMA	identifier	n	character set of the schema (currently not used)
CHARACTER_SET_NAME	identifier	n	character set of the name (currently not used)
COLLATION_CATALOG	identifier	n	collation of the catalog (currently

			not used)
COLLATION_SCHEMA	identifier	n	collation of the schema (currently not used)
COLLATION_NAME	identifier	n	collation of the name (currently not used)
DOMAIN_CATALOG	identifier	n	domain name of the catalog (currently not used)
DOMAIN_SCHEMA	identifier	n	domain name of the schema (currently not used)
DOMAIN_NAME	identifier	n	domain name (currently not used)

Table_Privileges View

Describes the privileges for the tables defined in the catalog.

This view is part of the ANSI SQL-2 standard Information Schema.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
GRANTOR	identifier	n	User who granted privilege.
GRANTEE	identifier	n	User who was granted privilege.
TABLE_CATALOG	identifier	n	catalog name
TABLE_SCHEMA	identifier	n	schema name
TABLE_NAME	identifier	n	table name
PRIVILEGE_TYPE	enumeration	n	privilege type: SELECT or INSERT or DELETE or UPDATE or EXECUTE or REFERENCES
IS_GRANTABLE	yes_no	n	YES if privilege is grantable; NO if not

Column_Privileges View

Describes the privileges on the columns defined in the catalog.

This view is part of the ANSI SQL-2 standard Information Schema.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
GRANTOR	identifier	n	user who granted privilege.
GRANTEE	identifier	n	user who was granted privilege.
TABLE_CATALOG	identifier	n	table catalog
TABLE_SCHEMA	identifier	n	schema name
TABLE_NAME	identifier	n	table name
COLUMN_NAME	identifier	n	column name
PRIVLEGE_TYPE	enumeration	n	privilege type: SELECT or INSERT or DELETE or UPDATE or EXECUTE or REFERENCES
IS_GRANTABLE	yes_no	n	YES if privilege is grantable; NO if not

Table_Constraints View

Describes the table constraints.

This view is part of the ANSI SQL-2 standard Information Schema.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
CONSTRAINT_CATALOG	identifier	n	constraint catalog
CONSTRAINT_SCHEMA	identifier	n	constraint schema
CONSTRAINT_NAME	identifier	n	constraint name
TABLE_CATALOG	identifier	n	table catalog
TABLE_SCHEMA	identifier	n	table schema
TABLE_NAME	identifier	n	table name
CONSTRAINT_TYPE	enumeration	n	NOT NULL or UNIQUE or PRIMARY KEY or FOREIGN KEY
IS_DEFERABLE	yes_no	n	currently always 'N'
INITIALLY_DEFERRED	yes_no	n	currently always 'N'
SHORT_NAME	char(2)	y	Internal Adabas Identification of the constraint

Table_Indexes View

Describes the indices of the tables and the type of the index.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
INDEX_CATALOG	identifier	n	index catalog
INDEX_SCHEMA	identifier	n	index schema
INDEX_NAME	identifier	n	index name
TABLE_CATALOG	identifier	n	table catalog
TABLE_SCHEMA	identifier	n	Table schema
TABLE_NAME	identifier	n	Table name
SHORT_NAME	char(2)	n	Internal Adabas identification of the index
IS_MULTIPLE	yes_no	n	Index is based on Adabas MU/PE field
IS_MU_UQINDEX	yes_no	n	Multiple index has the Adabas UQ option
INDEX_DATA_TYPE	char(128)	y	contains zero or more of the following attributes: INDEX, PRIMARY, UNIQUE

Key_Column_Usage View

For every constraint (except NOT_NULL) and every index, the used columns and their ordinal positions within this order are listed.

This view is part of the ANSI SQL-2 standard Information Schema.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
CONSTRAINT_CATALOG	identifier	n	constraint catalog
CONSTRAINT_SCHEMA	identifier	n	constraint schema
CONSTRAINT_NAME	identifier	n	constraint name
TABLE_CATALOG	identifier	n	table catalog
TABLE_SCHEMA	identifier	n	table schema
TABLE_NAME	identifier	n	table name
COLUMN_NAME	identifier	n	column name
ORDINAL_POSTION	short	n	ordinal position of the column element inside the containing key
CONSTRAINT_TYPE	enumeration	n	contains zero or more of the following attributes: INDEX, PRIMARY, UNIQUE

Not_Null_Constraints View

Describes the NOT NULL constraints.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
CONSTRAINT_CATALOG	identifier	n	constraint catalog
CONSTRAINT_SCHEMA	identifier	n	constraint schema
CONSTRAINT_NAME	identifier	n	constraint name
TABLE_CATALOG	identifier	n	table catalog
TABLE_SCHEMA	identifier	n	table schema
TABLE_NAME	identifier	n	table name
COLUMN_NAME	identifier	n	column name

View_Table_Usage View

Identifies the table on which the catalog's views are dependent.

This view is part of the ANSI SQL-2 standard Information Schema.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
VIEW_CATALOG	identifier	n	view catalog (currently always CONNXDB)
VIEW_SCHEMA	identifier	n	view schema
VIEW_NAME	identifier	n	view name
TABLE_CATALOG	identifier	n	table catalog
TABLE_SCHEMA	identifier	n	table schema
TABLE_NAME	identifier	n	table name

View_Column_Usage View

Identifies the columns on which the catalog's views are dependent.

This view is part of the ANSI SQL-2 standard Information Schema.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
VIEW_CATALOG	identifier	n	view catalog (currently always CONNXDB)
VIEW_SCHEMA	identifier	n	view schema
VIEW_NAME	identifier	n	view name
TABLE_CATALOG	identifier	n	table catalog
TABLE_SCHEMA	identifier	n	table schema
TABLE_NAME	identifier	n	table name
COLUMN_NAME	identifier	n	column name

Constraint_Table_Usage View

Identifies the tables that are referenced by referential constraints.

This view is part of the ANSI SQL-2 standard Information Schema.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
TABLE_CATALOG	identifier	n	table catalog
TABLE_SCHEMA	identifier	n	table schema
TABLE_NAME	identifier	n	table name
CONSTRAINT_CATALOG	identifier	n	constraint catalog
CONSTRAINT_SCHEMA	identifier	n	constraint schema
CONSTRAINT_NAME	identifier	n	constraint name

Constraint_Column_Usage View

Identifies the columns that are referenced by referential constraints.

This view is part of the ANSI SQL-2 standard Information Schema.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
TABLE_CATALOG	identifier	n	table catalog
TABLE_SCHEMA	identifier	n	table schema
TABLE_NAME	identifier	n	table name
COLUMN_NAME	identifier	n	column name
CONSTRAINT_CATALOG	identifier	n	constraint catalog
CONSTRAINT_SCHEMA	identifier	n	constraint schema
CONSTRAINT_NAME	identifier	n	constraint name

Referential_Constraints View

Describes the referential constraints.

This view is part of the ANSI SQL-2 standard Information Schema.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
CONSTRAINT_CATALOG	identifier	n	constraint catalog
CONSTRAINT_SCHEMA	identifier	n	constraint schema
CONSTRAINT_NAME	identifier	n	constraint name
UNIQUE_CONSTRAINT_CATALOG	identifier	n	unique constraint catalog
UNIQUE_CONSTRAINT_SCHEMA	identifier	n	unique constraint schema
UNIQUE_CONSTRAINT_NAME	identifier	n	unique constraint name
MATCH_OPTION	enumeration	n	match option: NONE or PARTIAL or FULL
UPDATE_RULE	enumeration	n	update rule: NO ACTION or CASCADE or SET NULL or SET DEFAULT
DELETE_RULE	enumeration	n	delete rule: NO ACTION or CASCADE or SET NULL or SET DEFAULT
IS_CLUSTERING	yes_no	n	Y if the referential constraint is implied by a table cluster

Users View

Identifies users.

Column	Data Type	N	Description
RECORD_NUMBER	integer	n	unique identifier for this record of metadata
USER_ID	identifier	n	user name
USER_SCHEMA	identifier	n	schema for the user
CHARACTER_SET	identifier	n	character set for the user (not currently used)

OLE DB Data Types

The following table contains the available data types for OLE DB-compliant databases supported by CONNX.

CONNX Data Type	SQL Data Type	Length	VMS Equivalent Data Type	Description
Access Currency	CURRENCY	-1		
Binary	CHAR	8		This date should not be used. It is here for backward compatibility. See VMS DATE/Time. This converts a VMS date into a 23-character string.
Binary (Text)	CHAR	-1		
OLEDB Date	DATE	6		16-byte Timestamp with no time information.
OLEDB Decimal	DECIMAL	-1		4-byte number with precision and scale.
OLEDB Double	DOUBLE	8		Double precision floating pint.
OLEDB Number	DECIMAL	-1		4-byte number with precision and scale.
OLEDB Real	REAL	4		Single precision floating point.
OLEDB Time	TIME	16		16-byte Timestamp with no date information.
OLEDB Timestamp	TIMESTAMP	16		16-byte Timestamp
OLEDB Wstr	UNICODE	1-255		Unicode string
Text (Right Space Padded)	CHAR	1-30000	Text Maxcim Yes/No	Alphanumeric and symbols. (1-30000 chars); same as Text but with right space padding.
Unsigned Byte	TINYINT	1		
Unsigned Byte Double	DOUBLE	1		
Unsigned Byte -> Decimal	DECIMAL	1		
Unsigned Long	INTEGER	4		

Oracle Data Types

The following table contains the available data types for Oracle databases supported by CONNX.

CONNX Data Type	SQL Data Type	Length	VMS Equivalent Data Type	Description
Big Double Text	CHAR	48		Oracle Large Number (38)
Binary (Text)	CHAR	1-30000	Maxcim RFA Maxcim Byte Array	The binary field is not converted. It provides access to raw, unaltered data in an RMS file.
BLOB	LONG VARBINARY	0-2147483000		Binary Large Object
CLOB	LONG VARCHAR	0-2147483000		Character Large Object
Oracle Date	TIMESTAMP	16		Oracle Date
Oracle MLSLABEL	BINARY	1-19		Oracle MLSLABEL
Oracle Rowid	BINARY	16		Oracle Rowid
Oracle Timestamp	TIMESTAMP	-1		
Text (Right Space Padded)	CHAR	1-30000	Text Maxcim Yes/No	Alphanumeric and symbols. (1-30000 chars); same as Text but with right space padding.
Text Space Padded (no nulls)	CHAR	-1		
Varbinary	VARBINARY	-1		

DB2 Data Types

The following table contains the available data types for DB2 databases supported by CONNX.

CONNX Data Type	SQL Data Type	Length	ODBC Data Type	Description
DB2 Eur Date (dd.mm.yyyy)	Date	10	SQL_DATE	
DB2 Eur Time (hh.mm.ss.)	Time	8	SQL_TIME	
DB2 IEEE 4-byte float	Real	4	SQL_REAL	4-byte byte-reversed (little endian) IEEE float
DB2 IEEE 8-byte float	Double	8	SQL_DOUBLE	8-byte byte-reversed (little endian) IEEE float
DB2 IEEE Big E 4-byte float	Real	4	SQL_REAL	4-byte big endian IEEE float
DB2 IEEE Big E 8-byte float	Double	8	SQL_DOUBLE	8-byte big endian IEEE float
DB2 ISO Date (YYYY-MM-DD)	Date	10	SQL_DATE	ISO DATE column
DB2 ISO Time (hh.mm.ss)	Time	8	SQL_TIME	ISO TIME column
DB2 JIS Date (YYYY-MM-DD)	Date	10	SQL_DATE	
DB2 JIS Time (hh:mm:ss)	Time	8	SQL_TIME	
DB2 Longword Big Endian	Integer	4	SQL_INTEGER	
DB2 Longword Integer	Integer	4	SQL_INTEGER	4-byte signed long integer (-2147483648 thru +2147483647)
DB2 Mainframe 4-byte float	Real	4	SQL_REAL	Mainframe DB2 (MVS or OS/390) 4-byte float
DB2 Mainframe 8-byte float	Double	8	SQL_DOUBLE	Mainframe DB2 (MVS or OS/390) 8-byte float
DB2 Numeric	Numeric	1-31	SQL_NUMERIC	Zoned numeric columns defined as NUMERIC (precision, scale), where precision >= scale; 0 <= precision <=31; 0 <= scale <=31
DB2 Numeric Integer	Numeric	9	SQL_DECIMAL	Describes an SQL NUMERIC display column with a scale = 0 and precision < = 9, e.g., NUMERIC (9,0). The default conversion from host column to ODBC data type is from SQL_NUMERIC to signed long integer = SQL_C_SLONG.
DB2 Packed Decimal	Decimal	1-16	SQL_DECIMAL	Packed decimal columns defined as DECIMAL (precision, scale), where precision >= scale; 0 <= precision <=31; 0 <= scale <=31
DB2 Packed Decimal Integer	Decimal	9	SQL_DECIMAL	This data type corresponds to an SQL_DECIMAL column with a scale of 0 and a precision < = 9, e.g., DECIMAL(9,0). The default conversion from host column to ODBC data type is from SQL_DECIMAL to

				signed long integer = SQL_C_SLONG.
DB2 Small Integer	Smallint	2	SQL_SMALLINT	2-byte signed small integer (-32768 thru +32767)
DB2 Timestamp	Timestamp	26	SQL_TIMESTAMP	ISO Timestamp column (YYYY-MM-DD - HH.MM.SS.123456)
DB2 Text (Null Terminated)	Varchar	255-32767	SQL_VARCHAR	255+ byte variable-length binary (defined as VARCHAR/LONG VARCHAR)
DB2 Text (Right Space Padded)	Char	1-254	SQL_CHAR	1- to 254-byte fixed-length character
DB2 USA Date (mm/dd/yyyy)	Date	10	SQL_DATE	
DB2 USA Time (hh:mm xM)	Time	8	SQL_TIME	
DB2 Varbinary	Varbinary	255-32767	SQL_VARBINARY	255+ byte variable-length binary (defined as VARCHAR/LONG VARCHAR FOR BIT DATA)
DB2 Word Big Endian	SmallInt	2	SQL_SMALLINT	
Double Prec float (Mainframe)	Double	8	SQL_DOUBLE	
Longword 4 bytes	Integer	4	SQL_INTEGER	
Longword (BE) -> Double	Double	4	SQL_DOUBLE	
Longword (BE) -> Decimal	Decimal	4	SQL_DECIMAL	
Single Prec float (Mainframe)	Real	4	SQL_REAL	
Text (Right Space Padded)	Char	-1	SQL_CHAR	
Text DB2 (Right Space Padded)	Char	-1	SQL_CHAR	
Text DB2 (Null Terminated)	VarChar	-1	SQL_VARCHAR	
Unsigned Long (BE) -> Decimal	Decimal	2	SQL_DECIMAL	
Unsigned Word (BE) -> Decimal	Decimal	2	SQL_DECIMAL	
Unsigned Long (BE) -> Double	Double	4	SQL_DOUBLE	
Unsigned Word (BE) -> Double	Double	2	SQL_DOUBLE	
Word 2 bytes	SmallInt	2	SQL_SMALLINT	
Word (BE) -> Decimal	Decimal	2	SQL_DECIMAL	
Word (BE) -> Double	Double	2	SQL_DOUBLE	

Oracle Rdb Data Types

The following table contains the available data types for Oracle Rdb databases supported by CONNX.

CONNX Data Type	SQL Data Type	Length	VMS Equivalent Data Type	Description
Binary	BINARY	1-30000	Maxcim RFA Maxcim Byte Array	The binary field is not converted. It provides access to raw, unaltered data in an RMS file.
Binary (Text)	CHAR	-1		
Double	DOUBLE	8	Vms Double RealMaxcim Double Real	Double precision floating point (8 bytes)
Interval Date/Day/Second	DOUBLE	-1		Rdb Interval data type
Interval Date Hour	DOUBLE	-1		Rdb Interval data type
Interval Date Hour/Minute	DOUBLE	-1		Rdb Interval data type
Interval Date Hour/Second	DOUBLE	-1		Rdb Interval data type
Interval Date Minute	DOUBLE	-1		Rdb Interval data type
Interval Date Minute/Second	DOUBLE	-1		Rdb Interval data type
Interval Date Month	DOUBLE	-1		Rdb Interval data type
Interval Date Second	DOUBLE	-1		Rdb Interval data type
Interval Date Year	DOUBLE	-1		Rdb Interval data type
Interval Date Year/Month	DOUBLE	-1		
Interval Day	DOUBLE	-1		Rdb Interval data type
Interval Hour	DOUBLE	-1		Rdb Interval data type
Interval Minute	DOUBLE	-1		Rdb Interval data type
Longword -> Decimal	DECIMAL	4	VMS Longword Integer with implied decimal PIC S9(X1)V9(X2) COMP where X1 + X2 is between 5 and 9.	Longword Integer with an implied decimal place (4 bytes).
Longword -> Double	DOUBLE	4		

Longword	INTEGER	4	VMS Longword Integer PIC S9(X1) COMP where X1 is between 5 and 9.	Longword Integer (4 bytes)
Quadword -> Char	VARCHAR	8		
Quadword -> Char(DP)	VARCHAR	8		
Quadword -> Decimal	DECIMAL	8	VMS Quadword Integer with implied decimal PIC S9(X1)V9(X2) COMP where X1+X2 is between 10 and 20.	Quadword Integer with an implied decimal place (8 bytes) converts to a SQL String.
Rdb Date ANSI	DATE	16		
Rdb Date VMS	TIMESTAMP	16		
Rdb Date VMS (String)	TIMESTAMP	16		
Rdb Time	TIME	16		
Rdb Timestamp	TIMESTAMP	16		
Signed Overpunch -> Double	DOUBLE	1		
Single	REAL	16		
String Tinyint	TINYINT	-1		
String Smallint	SMALLINT	-1		
String Integer	INTEGER	-1		
String Quadword	VARCHAR	-1		
String Single	REAL	-1		
String Double	DOUBLE	-1		
Text (Expandable)	VARCHAR	1-30000		Alphanumeric and symbols (1-30000 chars).
Text (Null Terminated)	VARCHAR	-1		
Text (Right Space Padded)	CHAR	1-30000	Text Maxcim Yes/No	Alphanumeric and symbols. (1-30000 chars); same as Text but with right space padding.
VMS Date/Time	TIMESTAMP	8		VMS Binary Date/Time (8 bytes) converts to an ODBC Timestamp.

VMS G Float	DOUBLE	8		
Word	SMALLINT	2		

PostgreSQL Data Types

PostgreSQL Data Type	SQL Data Type	Length	Description
Binary (Text)	CHAR	-1	
CONNXStore Timestamp	TIMESTAMP	8	
CONNXStore Date	DATE	4	
CONNXStore Time	TIME	8	
Numeric	Integer		
PostgreSQL Numeric (BE) -> Decimal	DECIMAL	-1	
PostgreSQL Numeric (7.4+) -> Decimal	DECIMAL	-1	
PostgreSQL Numeric (7.4+ BE) -> Decimal	DECIMAL	-1	
Text -> Currency	CURRENCY	-1	

Non-relational Databases

Adabas Data Types

The following table contains the available data types for Adabas databases supported by CONNX.

Adabas Data Type	CONNX Data Type	SQL Data Type	Length	Description
Adabas Text (LA VarChar)		VarChar	-1	
Adabas Text (L4 VarChar)		VarChar	-1	
ADABASPACKED	Adabas PACKED Decimal -> Integer	LONG	-1	
ADABASPACKED_NUMERIC	Adabas PACKED Decimal -> Decimal	DECIMAL	-1	
NATURALTIMESTAMP_DATE	Adabas Natural Timestamp -> Date	DATE	7	
Alphanumeric	Char (Right Space Padded)	CHAR	253	Adabas Alphanumeric
Binary	Binary (Text)	BINARY	1-126	Adabas Binary
Double IEEE 8-byte (BE)	Double IEEE 8-byte (BE)	DOUBLE	8	
Double IEEE 8-byte	Double IEEE 8-byte	DOUBLE	8	
Fixed Point <= 2 digits	Word (BE) - mainframe Word - Non-mainframe	SMALLINT	1-2	Adabas Fixed Point with Lengths <= 2
Fixed Point > 2 digits	Longword (BE) - mainframe Longword - Non-mainframe	INTEGER	>2	Adabas Fixed Point with Lengths > 2
Float Point (4 bytes)	Single Prec float (mainframe) - mainframe Float IEEE 4-byte - Non-mainframe	REAL	4	Adabas Floating Point for Lengths <= 4
Float Point (8 bytes)	Double Prec float (mainframe) - mainframe	DOUBLE	8	Adabas Floating Point for Lengths > 4

	Double IEEE 8-byte - Non-mainframe			
Longword 4 bytes	Longword	INTEGER	4	
Longword (BE)	Longword (BE)	INTEGER	4	
Longword (BE) -> Decimal	Longword (BE) -> Decimal	DECIMAL	4	
Longword (BE) -> Double	Longword (BE) -> Double	DOUBLE	4	
Natural Date	Adabas Natural Date	DATE	6	This type refers to the Natural Data data type.
Natural Time	Adabas Natural Time	TIME	6	This type refers to the Natural Time data type.
Natural Timestamp	Adabas Natural Timestamp	TIMESTAMP	12	This type refers to the Natural Timestamp data type.
Packed Decimal (Length <= 5)	PACKED Decimal -> Integer	INTEGER	<= 5	Adabas Packed Decimal with Lengths <= 5
Packed Decimal	PACKED Decimal -> Decimal	DECIMAL	> 0	
Quadword (BE) -> Char (DP)	Quadword (BE) -> Char (DP)	VARCHAR	8	
Quadword (BE) -> Char	Quadword (BE) -> Char	VARCHAR	8	
Quadword (BE) -> Decimal	Quadword (BE) -> Decimal	DECIMAL	8	
Quadword (BE) -> Double	Quadword (BE) -> Double	DOUBLE	8	
Unicode Char (UTF-8)		Unicode	-1	
Unicode L4 VarChar (UTF-8)		Unicode	-1	
Unicode LA VarChar (UTF-8)		Unicode	-1	
Unicode VarChar (UTF-8)		Unicode	-1	
Unpacked Decimal (Length < 10)	Adabas Unpacked EBCDIC (Integer) - mainframe Adabas Unpacked Decimal -> Integer - Non-mainframe	INTEGER	< 10	Adabas Unpacked Decimal with Lengths < 10

Unpacked Decimal	Adabas Unpacked EBCDIC (Numeric) - mainframe Adabas Unpacked -> Decimal- Non-mainframe	NUMERIC	> 0	

All of the CONNX data types have been provided in the tables in the next several topics under CONNX Data Types, although only the text-based CONNX data types can be used with Adabas.

DataFlex/PowerFlex Data Types

The following table contains the available data types for DataFlex/PowerFlex databases supported by CONNX.

DataFlex Data Type	SQL Data Type	Length	Description
Binary (Text)	CHAR	-1	
Flex Binary	Binary	1-32000	Binary value
Flex Date	DATE	10	Date in ODBC format. DataFlex date can be in the MM/DD/YY or MM/DD/YYYY format.
Flex Double	DOUBLE	TSZ	
Flex Longword	INTEGER	TSZ	
Flex Text	CHAR	1-32000	Alphanumeric and symbols (1-32000 chars).
Flex Numeric	DECIMAL	-1	Dataflex numeric to SQL Decimal
Text -> Currency	CURRENCY	-1	
Text (Right Space Padded)	CHAR	-1	

C-ISAM, DISAM, Micro Focus, and RM Cobol Data Types

Import Code	C-ISAM, DISAM, and Micro Focus Equivalent	CONNX Data Type	Length	Description
307		4C Date	4	Date in the form of days since 12/31/1799 converts to SQL Date
421		Adabas Natural Timestamp ->	7	
36	CHAR	BINARY (TEXT) (C-ISAM CHARTYPE filled with binary data)	-1	Binary data.
280		CISAM DIBOL Date (5 bytes)	5	
251		CISAM Packed Decimal	1-255	Converts to a double precision floating point.
274		CISAM Packed Decimal 1		COBOL packed decimal data type.
275		CISAM Packed Decimal 2		Native CISAM data type for very large decimal numbers that will not fit into a double.
326		CISAM Zoned Numeric -> Decimal	1-17 char	p-y overpunch character to SQL Decimal
278		CISAM Zoned Numeric -> Double	1-19	Converts to a double precision floating point.
279		CISAM Zoned Numeric -> Integer	1-19	Converts to a long.
272	DOUBLE	DOUBLE IEEE 8 BYTE	8	For CISAM DOUBLE on non-RISC machines.
273	DOUBLE	DOUBLE IEEE 8 BYTE (BE)	8	For CISAM DOUBLE on RISC machines. (Unix only)
270	FLOAT	FLOAT IEEE 4 Bytes	4	For CISAM FLOAT on non-RISC machines.
271	FLOAT	FLOAT IEEE 4 Bytes (Big Endian)	4	For CISAM FLOAT on RISC machines. (Unix only)
308		Gestepargne Date	1	
433		HCHD Serial Date	-1	
13		LONGWORD	4	
	LONG	LONGWORD 4 Bytes (Big Endian)	4	For CISAM LONGTYPE.

424		Opus Date	6	
425		Opus Date Packed	3	
432		Opus Unsigned Packed -> Decimal	-1	
284		Quadword (BE) -> Char	8	
285		Quadword (BE) -> Chr(DP)	8	
327		Quadword Decimal (BE)	8	
288		Quadword Double (BE)	8	
1		Text (Right Space Padded)	-1	
399	CHAR	TEXT DATE (DD)	2	For DD, if DD = 30, the SQL Date would be January 30, 2003. The year returned is the current year. The month is always January.
41	CHAR	TEXT DATE (DDMMYY)	6	TEXT DATE in the specified format. Handles years between 1900 and 1999.
42	CHAR	TEXT DATE (DDMMYYYY)	8	TEXT DATE in the specified format.
398	CHAR	TEXT DATE (MM)	2	For MM, if MM was 11, the SQL Date would be November 1, 2003. The year returned is the current year. The day of the month is always one (1).
45	CHAR	TEXT DATE (MMDDYY)	6	TEXT DATE in the specified format. Handles years between 1900 and 1999.
46	CHAR	TEXT DATE (MMDDYYYY)	8	TEXT DATE in the specified format.
394	DATE	TEXT DATE (MMYYYY)	6	For MMYYYY , if MMYYYY was 032002, the SQL date would be March 1, 2003. The day of the month is always one (1).
43	CHAR	TEXT DATE (YYMMDD)	6	TEXT DATE in the specified format. Handles years between 1900 and 1999.

396	DATE	TEXT DATE (YYYY)	4	For YYYY, if YYYY was 2004, the SQL Date would be January 1, 2004. The day of the month is always one (1). The month is always January.
392	DATE	TEXT DATE (YYYYMM)	6	For YYYYMM , if YYYYMM was 200203, the SQL date would be March 1, 2003. The day of the month is always one (1).
44	CHAR	TEXT DATE (YYYYMMDD)	8	TEXT DATE in the specified format.
59	CHAR	TEXT DATE 2000 (DDMMYY)	6	TEXT DATE in the specified format. Handles years between 1941 and 2040.
60	CHAR	TEXT DATE 2000 (MMDDYY)	6	TEXT DATE in the specified format. Handles years between 1941 and 2040.
395	DATE	TEXT DATE 2000 (MMYY)	4	For MMYY , if MMYY was 0403, the SQL date would be April 1, 2003. The day of the month is always one (1). For MMYY , if MMYY was 0423, the SQL date would be April 1, 1923. The day of the month is always one (1). The break Year = 20. (For years below 20, add 2000; for years above 20, add 1900. The break year is configurable.)
397	DATE	TEXT DATE 2000 (YY)	2	For YY, if YY was 04, the SQL Date would be January 1, 2004. The day of the month is always one. The month is always January. For YY, if YY was 34, the SQL Date would be January 1, 1934. The day of the month is always one (1). The month is always January. The break Year = 20. (For years below 20, add 2000; for

				years above 20, add 1900. The break year is configurable.)
393	DATE	TEXT DATE 2000 (YYMM)	4	For YYMM , if YYMM was 0304, the SQL date would be April 1, 2003. The day of the month is always one (1). For YYMM , if YYMM was 2304, the SQL date would be April 1, 1923. The day of the month is always one (1). The break Year = 20. (For years below 20, add 2000; for years above 20, add 1900. The break year is configurable.)
61	CHAR	TEXT DATE 2000 (YYMMDD)	6	TEXT DATE in the specified format. Handles years between 1941 and 2040.
292		Unsigned Long 4 bytes (BE)	4	
329		Unsigned Quadword (BE) -> Dec	8	
295		Unsigned Quadword (BE) -> Integer	8	
299		Unsigned Quadword Double (BE)	8	
331		Unsigned VarLen Int(BE) -> Dec	-1	
297		Unsigned VarLen Int (BE) -> Char	-1	
298		Unsigned VarLen Int(BE) -> Chr(DP)	-1	
300		Unsigned Var Length Int Db (BE)	-1	
291		Unsigned Word 2 bytes (BE)	2	
286		VarLen Int (BE) -> Char	-1	
287		VarLen Int (BE) -> Chr(DP)		
289		VarLen Int (BE) -> Double	-1	

330		VarLen Int (BE) -> Decimal	-1	
262		Word (BE)	2	
180	INT	Word 2 Bytes (Big Endian)	2	CISAM Word data type for reversed bytes.
350		Word Big Endian Date (2 bytes)	6	CISAM Date, Big Endian number of days since 1/1/1900
351		Word Big Endian Time (4 bytes)	6	CISAM Time, Big Endian number of seconds since midnight
352		Word Big Endian Timestamp	16	CISAM Timestamp, Big Endian number of seconds since 1/1/1970 00:00:00

Codasyl DBMS Data Types

The following table contains the available data types for Codasyl DBMS databases supported by CONNX.

CONNX Data Type	SQL Data Type	Length	VMS Equivalent Data Type	Description
Binary	CHAR	8		This date should not be used. It is here for backward compatibility. See VMS DATE/Time. This converts a VMS date into a 23-character string.
Binary (Text)	CHAR	-1		
Double	DOUBLE	8	VMS Double Real Maxcim Double Real	Double precision floating point (8 bytes).
Left Separate -> Decimal	DECIMAL	-1		
Left Separate -> Double	DOUBLE	1-20	Left Separate	Left Separate Double
Left Separate -> Integer	LONG	1-20	Left Separate	Left Separate Integer
Longword	INTEGER	4		
Longword -> Decimal	DECIMAL	4	VMS Longword Integer with implied decimal PIC S9(X1)V9(X2) COMP where X1+X2 is between 5 and 9.	Longword Integer with an implied decimal place (4 bytes).
Longword -> Double	DOUBLE	4		
Quadword -> Char	VARCHAR	8		
Quadword -> Char(DP)	VARCHAR	8		
Quadword Decimal	CHAR	8	VMS Quadword Integer with implied decimal PIC S9(X1)V9(X2) COMP where X1+X2 is between 10 and 20.	Quadword Integer with an implied decimal place (8 bytes) converts to a SQL String.
Signed Overpunch -> Decimal	DECIMAL	-1		
Signed Overpunch -> Double	DOUBLE	-1		
Single	REAL	4	VMS Single Real Maxcim Single Real	Single precision floating point (4 bytes).
Text (Expandable)	VARCHAR	1-30000		Alphanumeric and symbols. (1-30000 chars).
Text (Right Space Padded)	CHAR	1-30000	Text Maxcim Yes/No	Alphanumeric and symbols. (1-30000 chars); same as Text but with right space padding.
VMS Date/Time	TIMESTAMP	8		VMS Binary Date/Time (8 Bytes) converts to an ODBC Timestamp.

VMS Double -> Currency	DECIMAL	8		
VMS G Float	DOUBLE	8		
Word	SMALLINT	2		
Word -> Float	REAL	2		
Word Double (2)	DOUBLE	2		Word converted to double with implied decimal point.
Word Numeric -> Decimal	DECIMAL	2		

IBM Mainframe Data Types

CONNX Data Type	SQL Data Type	Length	Description
Double Prec float (Mainframe)	DOUBLE	8	8-byte big-endian mainframe double precision: USAGE COMP-2.
Longword (BE)	INTEGER	4	Signed big-endian 4-byte INTEGER PIC S9(n) COMP, where 5 <= n <= 9.
PACKED (F) Decimal -> Double	DOUBLE	1-10	Unsigned Packed Decimal PIC 9(precision,scale) USAGE COMP-3, where scale > 0 or 10 <= precision <= 18, or both.
PACKED (F) Decimal -> Integer	LONG	1-5	Unsigned Packed Decimal PIC 9(n) USAGE COMP-3, where 1 <= n <= 9.
PACKED Decimal -> Decimal	DECIMAL	1-10	Signed Packed Decimal PIC S9(precision,scale) USAGE COMP-3, where scale > 0 or 10 <= precision <= 18, or both.
PACKED Decimal -> Integer	LONG	1-5	Signed Packed Decimal PIC S9(n) USAGE COMP-3, where 1 <= n <= 9.
Signed Overpunch -> Decimal	DECIMAL	1-18	Signed Numeric Display PIC S9(precision,scale), where scale > 0 or 10 <= precision <= 18, or both.
Signed Overpunch -> Integer	LONG	1-9	Signed Numeric Display PIC S9(n), where 1 <= n <= 9.
Single Prec float (Mainframe)	REAL	4	4-byte big-endian mainframe float: USAGE COMP-1.
Text -> Decimal	DECIMAL	1-18	Unsigned Numeric Display PIC 9(precision,scale), where scale > 0 or 10 <= precision <= 18, or both.
Text -> Integer	INTEGER	1-9	Unsigned Numeric Display PIC 9(n), where 1 <= n <= 9
Text (Right Space Padded)	CHAR	1-32767	Character display PIC X(nnnnn), where 1 <= n <= 32767
Unsigned Long 4 bytes (BE)	INTEGER	4	Unsigned big-endian 4-byte INTEGER PIC S9(n) COMP, where 5 <= n <= 9.
Unsigned Word 2 Bytes (BE)	SMALLINT	2	Unsigned big-endian 2-byte INTEGER PIC 9(n) COMP, where 1 <= n <= 4.
Word (BE)	SMALLINT	2	Signed big-endian 2-byte INTEGER PIC S9(n) COMP, where 1 <= n <= 4.

CONNX Data Types

CONNX Data Types

The following tables contain the available data types for databases supported by CONNX. Several of the data types have more than one import code, supplied for compatibility with earlier versions of CONNX.

CONNX Data Types

CONNX Data Type	Import Code	SQL Data Type	Length	VMS Equivalent Data Types	Description
(TXT) VMS Date/Time	26	CHAR	8		This date should not be used. It is here for backward compatibility. See VMS DATE/Time. This converts a VMS date into a 23-character string.
24 bit Pascal Integer	99	INTEGER	3		24-bit Pascal Integer 3 bytes.
4C Date	307	DATE	4		Date in the form of days since 12/31/1799 converts to SQL Date
Access Currency	252	CURRENCY	-1		
Adabas Natural Date	403	DATE	4		
Adabas Natural Time	404	TIME	7		
Adabas Packed Decimal -> Integer	421	LONG	-1		
Adabas Packed Decimal -> Decimal	422	DECIMAL	-1		
Adabas Natural Timestam -> Date	423	DATE	7		
Adabas Text (LA VarChar)	441	VarChar	-1		
Adabas Text (L4 VarChar)	442	VarChar	-1		
ADL Date	121	DATE	4		Custom Data Type for ADL
ADL GL Number	122	DOUBLE	4		Custom Data Type for ADL
ADL Math	119	DOUBLE	1-20		Custom Data Type for ADL

ADL MMIS Number	120	CHAR	23		Custom Data Type for ADL
ADL Old Math	125	VARCHAR	1-8		Custom Data Type for ADL
ADL Type Code	123	VARCHAR	2		Custom Data Type for ADL
ADL Zip Code	124	LONG	4		Custom Data Type for ADL
Adonix Date	383	DATE	6		Customer-specific Julian date offset from 12/31/1599
Allport Julian Date (2 byte)	155	DATE	2		Custom Data Type for Allport
Allport Julian Date (4 byte)	116	DATE	4		Allport Julian Date
Allport Julian Date String	356	DATE	5		Character string Julian (jumeric) date offset from 1/1/1970
Allport Time	117	TIME	4		Allport Time
ANSI/ISO SQL Timestamp	434	Timestamp	-1		
Astrazeneca Interval	277	DATE	8		Customer-specific data type
Basic Plus Double	38	DOUBLE	8	Basic Plus Double	The Basic Plus Double Datatype is identical to the Double Datatype, with all bytes in reverse order.
Basic Plus Long	40	INTEGER	4		
Basic Plus Single	37	REAL	4	Basic Plus Single	The Basic Plus Single data type is identical to the Single data type, with all bytes in reverse order.
Basic Plus Word	39	SMALLINT	2	Basic Plus Word Integer	The Basic Plus Word Integer data type is identical to the Word Integer data type, with all bytes in reverse order.
Binary	35	BINARY	1 -30000	Maxcim RFA Maxcim Byte Array	The binary field is not converted. It provides access to raw, unaltered data in an RMS file.
Binary (Text)	36	CHAR	1-32000		Each byte of data is returned as a 2-character hexadecimal representation of the original byte of data.
BMS Date	78	DATE	3		3-byte Date for BMS.
BMS Reverse Date	79	DATE	3		3-byte Reverse Date for BMS.
BOSS Date CCYY	221	DATE	6		Custom 6-byte, Y2K-compliant date for BOSS.
BOSS Julian	223	DATE	5		Custom 5-byte, Y2K-

CCYY					compliant Julian date for BOSS.
BOSS Julian YYYYJJJ	240	DATE	7		Custom 7-byte, Y2K-compliant Julian date for BOSS.
BOSS Year CCYY	222	SMALLINT	4		Custom 4-byte, Y2K-compliant year for BOSS.
Byte -> Float	8	REAL	1	VMS Byte Integer with implied decimal.	Byte Integer with an implied decimal place (1 byte).
Byte	7	TINYINT	1		
Byte Bit (1/0)	9 (1001 for backward compatibility)	BIT	1		Evaluates Byte to a True or False value, and represents it as a bit.
Byte Bit #0 (2^0)	83	BIT	1		This represents bit #1 (2^0) from the byte.
Byte Bit #1 (2^1)	84	BIT	1		This represents bit #2 (2^1) from the byte.
Byte Bit #2 (2^2)	85	BIT	1		This represents bit #3 (2^2) from the byte.
Byte Bit #3 (2^3)	86	BIT	1		This represents bit #4 (2^3) from the byte.
Byte Bit #4 (2^4)	87	BIT	1		This represents bit #5 (2^4) from the byte.
Byte Bit #5 (2^5)	88	BIT	1		This represents bit #6 (2^5) from the byte.
Byte Bit #6 (2^6)	89	BIT	1		This represents bit #7 (2^6) from the byte.
Byte Bit #7 (2^7)	90	BIT	1		This represents bit #8 (2^7) from the byte.
Byte Bit Text(Yes/No)	10	CHAR	3		Evaluates Byte to a True or False value, and represents it as Yes or No.
Byte Numeric -> Decimal	309	DECIMAL	1		Single signed byte value converts to a SQL Decimal
CA Bit Flag	306	INTEGER	1		Customer-specific (1 byte of individual bit flags)
CA Window	305	VarCHAR	150		Customer-specific (up to 5 lines of 30 chars each)
CISAM DIBOL Date (5 bytes)	280	DATE	5		
CISAM Zoned Numeric -> Double	278	DOUBLE	-1		
CISAM Zoned Numeric -> Integer	279	LONG	-1		
Coda Year	76	SMALLINT	2		Word, a year is

					represented in a number offset from the year 1900. A value of 50 would represent 1950, and a value of 103 would represent the year 2003.
Cognos JDate	43 (1041 for backward compatibility)	DATE	2		Cognos Powerhouse Julian Date (2 Bytes) converts to an ODBC Date
Cognos PHDate	33 (1040 for backward compatibility)	DATE	2		Cognos Powerhouse Date (2 bytes) converts to an ODBC Date.
Cognos PHDate 2000	239	DATE	2		Cognos Powerhouse Date (2 bytes) converts to an ODBC Date.
Comment Prefix	269	VARCHAR	-1		
Compressed String	243	VARCHAR	8		CONNX removes all spaces from the text string.
Compufast Text Date (YYYYMMDD)	435	Date	10		
CS Comment	440	VarChar	-1		
DAI History Date	106	DATE	8		Date in YYMMDD format subtracted from 999999. Site-specific.
DAI History Period	109	SMALLINT	2		Period in PP format subtracted from 99. Site-specific.
DAI History Time	107	DATE	6		Time in HHMMSS format subtracted from 999999.
DAI History Year	108	SMALLINT	4		Year in YYYY format subtracted from 9999.
Decimal (Formatted with DP)	238	DECIMAL	1-255		Number stored as text with decimal point physically stored in field.
DIBOL Date (5 bytes)	170	DATE	5		
Double Prec float (mainframe)	264	SQL_DOUBLE	2		
Double Text Formatted	19	DOUBLE	8		
EDS Date YYMMDD	242	DATE	6		Site-specific
EDS Julian Date YYDD	241	DATE	5		Site-specific
Encapsulated Date (cyMMDD)	388	SQL_DATE	6		Byte 0 -> binary number representing the most significant two digits of a four digit year. Byte 1 -> binary number

					representing the least significant two digits of a four digit year. Thus, if given the first two bytes values of 20, 03, the year for this date are 2003. Bytes 2 & 3 are the numeric characters representing the month of the year. Bytes 4 & 5 are the numeric characters representing the day of the month. Thus a value of 0x010131313131 is a date of November (month 11) 11 (day 11) in the year 101.
Fairfield Text Date (DD)	403	DATE	2		For DD, if DD were 20, then the SQL Date would be January 20, 1900. The year returned in always 1900. The month returned is always January.
Fairfield Text Date (MM)	402	DATE	2		For MM, if MM were 11, then the SQL Date would be November 1, 1900. The year returned in always 1900. The day returned is always 1.
Fairfield Text Date (MMDD)	401	DATE	4		For MMDD, if MM were 11 and DD were 20, then the SQL Date would be November 20, 1900. The year returned is always 1900.
Float IEEE 4-byte	270	REAL	4		
Float IEEE 4-byte (BE)	271	REAL	4		
Globally Unique ID (GUID)	436	Char	16		
Julian Timestamp	441	SQL_TIMESTAMP	16		
KCS Compressed Integer	161	INTEGER			Site-specific.
KCS Date (3 bytes)	167	DATE	3		Site-specific.
KCS Phone	162	CHAR	5		Site-specific.
KCS Time (2 byte)	164	TIME	2		Site-specific.
KCS Time (3 bytes)	165	TIME	3		Site-specific.
KCS Zip Code	163	CHAR	5		Site-specific.

Left Separate -> Decimal	319	DECIMAL	1-17 char		Character to SQL Decimal with leading sign
Left Separate -> Double	110	DOUBLE	1-20	Left Separate	Left Separate Double
Left Separate -> Integer	111	LONG	1-20	Left Separate	Left Separate Integer
Long Date (DDMMYY)	47 (1035 for backward compatibility.)	DATE	4		Longword Date in the specified format. Handles years between 1900 and 1999.
Long Date (DDMMYYYY)	48 (1036 for backward compatibility.)	DATE	4		Longword Date in the specified format.
Long Date (MMDDYY)	51 (1048 for backward compatibility.)	DATE	4		Longword Date in the specified format. Handles years between 1900 and 1999.
Long Date (MMDDYYYY)	52 (1049 for backward compatibility.)	DATE	4		Longword Date in the specified format.
Long Date (YYMMDD)	49 (1037 for backward compatibility.)	DATE	4		Longword Date in the specified format. Handles years between 1900 and 1999.
Long Date (YYYYMMDD)	50 (1038 for backward compatibility.)	DATE	4		Longword Date in the specified format.
Long Date 2000 (DDMMYY)	62	DATE	4		Longword Date in the specified format. Handles years between 1941 and 2040.
Long Date 2000 (MMDDYY)	64	DATE	4		Longword Date in the specified format. Handles years between 1941 and 2040.
Long Date 2000 (YYMMDD)	63	DATE	4		Longword Date in the specified format. Handles years between 1941 and 2040.
Longword	13	INTEGER	4	VMS Longword Integer PICS9(X1) COMP where X1 is between 5 and 9.	Longword Integer (4 bytes)
Longword (BE)	263	SQL_INTEGER	4		
Longword (BE) -> Double	266	DOUBLE	4		
Longword (BE)	263	SQL_INTEGER	4		
Longword Currency	71 (1032 for backward compatibility.)	CURRENCY	4		

Longword -> Decimal	311	DECIMAL	1-17 char		Character to SQL Decimal with leading sign
Longword Decimal Big Endian	267	DOUBLE	4		
Longword -> Double	14 (1023 for backward compatibility.)	DOUBLE	4	VMS Longword Integer with implied decimal PIC S9(X1)V9(X2) COMP where X1+X2 is between 5 and 9.	Longword Integer with an implied decimal place (4 bytes).
Longword JDate (BE)	437	Date	4		

CONNX Data Types - Marc BIB Decimal to Timestamp14

CONNX Data Type	Import Code	SQL Data Type	Length	VMS Equivalent Data Types	Description
Marc BIB Decimal	381	DECIMAL	-1		Customer-specific base 240 number conversion
Marc BIB Number	339	LONG	-1		
Marc BIB Tag	382	LONG	-1		Customer-specific base 240 number conversion
MAXCIM Key Date	68	DATE	4		Maxcim special key date format.
MG Word Date	408	DATE	2		
Morse Date	154	DATE	4		Custom Data Type for Morse Data
Naftha Text	439	Char	-1		
National Compressed -> Decimal	321	DECIMAL	-1		Customer-specific numeric to SQL Decimal
National Compressed Double	118	DOUBLE	-1		Compressed double data type.
Noah Date	80	DATE	4		Site-specific.
Noah Inverse Date	81	DATE	4		Site-specific.
Noah Time	82	TIME	4		Site-specific.
Numeric Data (YYMMDD)	385	DATE	3		Three-byte data field were date is taken from the decimal digits of the values. Thus a numeric value of 750824 is interpreted as August 24 1975. Note that the year field uses a century window where values < 25 are assigned century 2000, everything else 1900.

					The new function is far more customer specific: TPADateOfBirth (date, centuryCode) Inputs: date - base birth date, for example a field of Numeric Date type. centuryCode - integer value indicating century of birth, 8 = 1800, 9 = 1900, 0 == 2000, no other valid values.
Opus Date	424	DATE	6		
Opus Date Packed	425	DATE	3		
Pack Date (DDMMYY)	53	DATE	4		Packed Decimal in the specified format. Handles years between 1900 and 1999.
Pack Date (DDMMYYYY)	54	DATE	5		Packed Decimal in the specified format.
Pack Date (MMDDYY)	57	DATE	4		Packed Decimal in the specified format. Handles years between 1900 and 1999.
Pack Date (MMDDYYYY)	58	DATE	5		Packed Decimal in the specified format.
Pack Date (YYMMDD)	55	DATE	4		Packed Decimal Date in the specified format. Handles years between 1900 and 1999.

Pack Date (YYYYMMDD)	56	DATE	5		Packed Decimal in the specified format.
Pack Date 2000 (DDMMYY)	65	DATE	4		Packed Decimal Date in the specified format. Handles years between 1941 and 2040.
Pack Date 2000 (MMDDYY)	67	DATE	4		Packed Decimal Date in the specified format. Handles years between 1941 and 2040.
Pack Date 2000 (YYMMDD)	66	DATE	4		Packed Decimal Date in the specified format. Handles years between 1941 and 2040.
Packed (A) Decimal -> Decimal	316	DECIMAL	-1		Packed decimal to SQL Decimal, value of 'A' denotes negative
PACKED (A) Decimal -> Double	100	DOUBLE	-1		Packed Decimal of type (A)
PACKED (A) Decimal -> Int	101	LONG	-1		Packed Decimal of type (A)
Packed (E) Decimal -> Decimal	317	DECIMAL	-1		Packed decimal to SQL Decimal, value of 'E' denotes negative
PACKED (E) Decimal -> Double	102	DOUBLE	-1		Packed Decimal of type (E)
PACKED (E) Decimal -> Int	103	LONG	-1		Packed Decimal of type (E)
Packed (F) Decimal -> Decimal	318	DECIMAL	-1		Packed decimal to SQL Decimal, value of 'E' denotes negative

PACKED (F) Decimal -> Double	104	DOUBLE	-1		Packed Decimal of type (F)
PACKED (F) Decimal -> Int	105	LONG	-1		Packed Decimal of type (F)
Packed Decimal -> Decimal	314	DECIMAL	-1		Packed decimal converts to SQL Decimal
Packed Decimal COMP6 -> Double	224	DOUBLE	-1	VMS Packed Decimal PIC S9(X1)V9(X2) COMP-3	Packed Decimal (1-255 bytes) converts to a double precision floating point; same as Packed Decimal (Double) but unsigned.
Packed Decimal COMP6 -> Decimal	325	DECIMAL	-1		Unsigned packed decimal to SQL Decimal
Packed Decimal COMP6 -> Integer	225	LONG	-1	VMS Packed Decimal PIC S9(X1)V9(X2) COMP-3	Packed Decimal (1-255 bytes) converts to a long; same as Packed Decimal (Integer) but unsigned.
Packed Decimal -> Currency	73 (1047 for backward compatibility.)	CURRENCY	-1		The Packed Decimal Currency data type is identical to the Packed Decimal data type, except the data is converted to a SQL Decimal instead of a SQL Double. This may provide greater precision.

Packed Decimal -> Double	22 (1014 with fraction > 0 for backward compatibility.)	DOUBLE	-1	VMS Packed Decimal PIC S9(X1)V9(X2) COMP-3	Packed Decimal (1-255 bytes) converts to a double precision floating point.
Packed Decimal -> Integer	23 (1014 with fraction = 0 for backward compatibility.)	LONG	-1	VMS Packed Decimal PIC S9(X1)V9(X2) COMP-3	Packed Decimal (1-255 bytes) converts to a long.
Pinnacle Date	283	DATE	6		
PioTech Date	443	SQL_DATE	6		
PioTech Numeric	442	SQL_BIGINT	4		
Pk Centry Date (CYYMMDD)	249	DATE	4		
POISE Double	95	DOUBLE	-1		ASCII double left space padded with explicit decimal point and sign.
POISE Double (Right Padded)	259	DOUBLE	-1		ASCII double right space padded with explicit decimal point and sign. Used to convert the data type in POISE RMS key field(s).
POISE Julian Date	260	DATE	2		2 byte, Julian format (1 = 01/01/1970)
POISE Record Number	247	INTEGER	4	None	A special data type used only in the record field of any POISE key file.

POISE Text Date (MMDDYYYY)	258	DATE	8		POISE Text date in the specified format.
POISE Text Date 2000 (MMDDYY)	257	DATE	6		POISE Text date in the specified format. Handles years between 1941 and 2040.
POISE Time	261	TIME	2		2 byte field contains minutes after midnight.
POISE Txt (Right Pad) -> Decimal	324	DECIMAL	-1		Customer-specific character to SQL Decimal
POISE Validate Code	248	VARCHAR	-1	None	A short code descriptor is populated in the CDD comment field. When the field is displayed using CONNX, the code descriptor prefixes the data value.
PostgreSQL Numeric -> Decimal	406	DECIMAL	-1		
PROMIS Timesetamp	418	TIMESTAMP	4		
Quadword -> Char	15	CHAR	8		
Quadword -> Char (DP)	16	CHAR	8		
Quadword -> Currency	72 (1033 for backward compatibility.)	CURRENCY	8	DEC ODBC Quadword	The Quadword Currency data type is identical to the Quadword Decimal data type, except the

					data is converted to a SQL Decimal instead of a SQL Double. This may provide greater precision.
Quadword -> Decimal	312	DECIMAL	8		Four-byte signed value converts to a SQL Decimal
Quadword -> Double	171	DOUBLE	8	VMS Quadword Integer with implied decimal PIC S9(X1)V9(X2) COMP where X1+X2 is between 10 and 20.	Quadword Integer with an implied decimal place (8 bytes) converts to a SQL Double.
Quadword -> Char	15 (1029 for backward compatibility.)	VARCHAR	8	VMS Quadword Integer PIC S9(X1)V9(X2) COMP where X1+X2 is between 10 and 20.	Quadword Integer (8 bytes) converts to a SQL String.
Quadword (BE) -> Char	284	VARCHAR	8		
Rev Encapsulated Date (MMDDcy)	389	SQL_DATE	6		Byte 0 -> binary number representing the most significant two digits of a four digit year. Byte 1 -> binary number representing the least significant two digits of a four digit year. Thus, if given the first two bytes values of 20, 03, the year for this date are 2003. Bytes 2 & 3 are the numeric characters representing the month of the year. Bytes 4 & 5 are the numeric characters representing the day of the month. Thus a value of 0x010131313131 is a date of November (month 11) 11 (day 11) in the year 101.

Reverse CODA Year	77	SMALLINT	2		Coda Year with bytes reversed.
Reverse Long Date YYYYMMDD	245	DATE	4		Subtracts 100000000 from the date, stored physically as a Long, yielding numbers that, if sorted, result in dates in descending order.
Reverse Text Date YYYYMMDD	244	DATE	8		Subtracts 100000000 from the date, stored physically as a Long, yielding numbers that, if sorted, result in dates in descending order.
Reverse VMS 4 Byte Date	75	DATE	4		VMS 4-byte date with bytes reversed.
Reverse VMS Date	30	DATE	8	Maxcim Date	VMS Binary Date/Time (8 Bytes) converts to an ODBC Date. All bytes are stored in reverse order.
Reverse VMS Date/Time	32	TIMESTAMP	8		VMS Binary Date/Time (8 bytes) converts to an ODBC Timestamp. All bytes are stored in reverse order.
Reverse VMS Time	31	TIME	8		VMS Binary Date/Time (8 bytes) converts to an ODBC Time. All bytes are stored in reverse order.

Reverse VMS Date (6 byte)	157	DATE	6		Reversed VMS Date stored in 6 bytes (fractional second precision is truncated).
Right Separate -> Decimal	320	DECIMAL	-1		Character to SQL Decimal with trailing sign
Right Separate -> Double	112	DOUBLE	-1	Right Separate	Right Separate Double
Right Separate -> Integer	113	LONG	-1	Right Separate	Right Separate Integer
Signed Overpunch -> Decimal	315	DECIMAL	-1		Overpunched decimal converts to SQL Decimal
Signed Overpunch -> Double	24 (1011 with fraction > 0 for backward compatibility.)	DOUBLE	-1	VMS Numeric String left overpunched sign	Signed Overpunch (1-255 digits) converts to a double precision floating point.
Signed Overpunch -> Integer	25 (1011 with fraction = 0 for backward compatibility.)	LONG	-1	VMS Numeric String left overpunched sign	Signed Overpunch (1-255 digits) converts to a long
Text -> Decimal	332	DECIMAL	-1		Character to SQL Decimal
Text -> Double	18	DOUBLE	-1		
Text -> Integer	17	INTEGER	-1		
Text (Expandable)	3	VARCHAR	-1		Alphanumeric and symbols. (1-30000 chars).
Text (Formatted) -> Double	19	DOUBLE	-1		

Text (Null Terminated)	2	VARCHAR	-1		
Text (Nullable) -> Double	254	DOUBLE	-1		
Text (Nullable) -> Integer	253	INTEGER	-1		
Text (Left Space Padded)	228	CHAR	1-30000	Text Maxcim Yes/No	Alphanumeric and symbols. (1-30000 chars); same as Text but with left space padding.
Text (Right Space Padded)	1 (1000 for backward compatibility.)	CHAR	1-30000	Text Maxcim Yes/No	Alphanumeric and symbols. (1-30000 chars); same as Text but with right space padding.
Text Currency	69 (1046 for backward compatibility.)	CURRENCY	1-20		Numeric money value stored as ASCII text with implied decimal point.
Text Date (DD)	399	DATE	2		For DD, if DD = 30, the SQL date would be January 30, 2003. The year returned is the current year. The month is always January.
Text Date (DDMMYY)	41 (1042 for backward compatibility.)	DATE	6		Text Date in the specified format. Handles years between 1900 and 1999.
Text Date (DDMMYYYY)	42 (1043 for backward compatibility.)	DATE	8		Text Date in the specified format.

Text Date (MM)	398	DATE	2		For MM, if MM was 11, the SQL date would be November 1, 2003. The year returned is the current year. The day of the month is always one.
Text Date (MMDDYY)	45 (1050 for backward compatibility.)	DATE	6		Text Date in the specified format. Handles years between 1900 and 1999.
Text Date (MMDDYYYY)	46 (1051 for backward compatibility.)	DATE	8		Text Date in the specified format.
Text Date (MMYYYY)	394	DATE	6		For MMYYYY, if MMYYYY was 032002, the SQL date would be March 1, 2003. The day of the month is always one (1).
Text Date (YYMMDD)	43(1044 for backward compatibility.)	DATE	6		DATE 6 Text Date in the specified format. Handles years between 1900 and 1999.
Text Date (YYYY)	396	DATE	4		For YYYY, if YYYY was 2004, the SQL date would be January 1, 2004. The day of the month is always one (1). The month is always January.
Text Date (YYYYMM)	392	DATE	6		For YYYYMM, if YYYYMM was 200203, the SQL date would be March 1, 2003. The day of the month is always one (1).
Text Date (YYYYMMDD)	44 (1055 for backward compatibility.)	DATE	8		Text Date in the specified format.
Text Date 2000 (DDMMYY)	59	DATE	6		Text Date in the specified format. Handles years between 1941 and 2040.

Text Date 2000 (MMDDYY)	60	DATE	6		Text Date in the specified format. Handles years between 1941 and 2040.
Text Date 2000 (MMYY)	395	DATE	4		For MMYY, if MMYY was 0403, the SQL date would be April 1, 2003. The day of the month is always one (1). For MMYY, if MMYY was 0423, the SQL date would be April 1, 1923. The day of the month is always one (1). The break year = 20 (For years below 20.., add 2000; years above 20, add 1900. Tthe break year is configurable.)
Text Date 2000 (YY)	397	DATE	2		For YY, if YY was 4, the SQL date would be January 1, 2004. The day of the months is always one. The month is always January. For YY, if YY was 34, the SQL date would be January 1, 1934. The day of the months is always one. The months is always January. The break year = 20 (For years below 20.., add 2000; years above 20, add 1900. Tthe break year is configurable.)
Text Date 2000 (YYMM)	393	DATE	4		For YYMM, if YYMM was 0304, the SQL date would be April 1, 2003. The day of the month is always one (1). For YYMM, if YYMM was 2304, the SQL date would be April 1, 1923. The day of the month is always one (1). The break year = 20 (For years below 20.., add 2000; years above 20, add 1900. Tthe

					break year is configurable.)
Text Date 2000 (YYMMDD)	61	DATE	6		Text Date in the specified format. Handles years between 1941 and 2040.
Text Left Space Pad -> Dbl	228	DOUBLE	-1		
Text (Formatted) -> Double	19	DOUBLE	-1		
Text Left Space Pad -> FmtDbl	298	DOUBLE	-1		
Text Left Space Pad -> Int	227	INTEGER	-1		
Text (Space Padded) No Nulls	218	CHAR	-1		
Time (Text HHMM)	281	TIME	4		
Time (Text HHMMSS)	282	TIME	6		
Timestamp12 (YYYYMMDDHHMM)	353	TIMESTAMP	12		Character Timestamp with year, month, day, hour, and minute

Timestamp14 (YYYYMMDDHHMMSS)	354	TIMESTAMP	14		Character Timestamp with year, month, day, hour, minute, and second

CONNX Data Types - TNRD to ZZ

CONNX Data Type	Import Code	SQL Data Type	Length	VMS Equivalent Data Types	Description
TNRD Julian Date	357	Date	2		Customer-specific Julian date offset from 1/1/1975
TNRD Rev Julian Date	358	Date	2		Customer-specific reverse Julian date offset from 1/1/1975
TNRD Identification Number	359	Varchar	-1		Customer-specific suffix delimited ID Number
TNRD Surname	360	Varchar	-1		Customer-specific suffix delimited Surname
TNRD First Name and Initial	361	Varchar	-1		Customer-specific suffix delimited First name and initial
TNRD Care of	362	Varchar	-1		Customer-specific suffix delimited Care of
TNRD Street Address Number	363	Varchar	-1		Customer-specific suffix delimited Street Address Number
TNRD Street Address Name	364	Varchar	-1		Customer-specific suffix delimited Street Address Name
TNRD St Address City/Province	365	Varchar	-1		Customer-specific suffix delimited Street Address City and Province
TNRD St Address Postal Code	366	Varchar	-1		Customer-specific suffix delimited Street Address Postal Code
TNRD Home Phone	367	Varchar	-1		Customer-specific suffix delimited Home Phone
TNRD Work Phone	368	Varchar	-1		Customer-specific suffix delimited Work Phone
TNRD Alt Address Num & Street	369	Varchar	-1		Customer-specific suffix delimited Alternate Street Address Number and Name

TNRD Alt Address City & Prov	370	Varchar	-1		Customer-specific suffix delimited Alternate Street Address City and Province
TNRD Ald Address Postal Code	371	Varchar	-1		Customer-specific suffix delimited Alternate Street Address Postal Code
TNRD School	372	Varchar	-1		Customer-specific suffix delimited School
TNRD Parent or Guardian	373	Varchar	-1		Customer-specific suffix delimited Parent/Guardian
TNRD Previous Barcode	374	Varchar	-1		Customer-specific suffix delimited Previous Barcode
TNRD Next Barcode	375	Varchar	-1		Customer-specific suffix delimited Next Barcode
TNRD Xref See	376	LONG	-1		Customer-specific prefix delimited See
TNRD Xref See Also	377	LONG	-1		Customer-specific prefix delimited See Also
TNRD Xref See From	378	LONG	-1		Customer-specific prefix delimited See From
TNRD Xref See Also From	379	LONG	-1		Customer-specific prefix delimited See Also From
TNRD Notes	380	Varchar	-1		Customer-specific prefix delimited Notes
Unicode Char (UTF-8)	446	Varchar	-1		
Unicode L4 VarChar (UTF-8)	448	Varchar	-1		
Unicode LA VarChar (UTF-8)	448	Varchar	-1		
Unicode VarChar (UTF-8)	447	Varchar	-1		
Unsigned Byte	215	TINYINT	1		

Unsigned Byte Double	301	DOUBLE	1		Single unsigned byte value converts to a SQL Double
Unsigned Long	214	INTEGER	4		
Unsigned Pk(F) Dt (DDMMYY)	340	DATE	4		Packed date DDMMYY to SQL Date
Unsigned Pk(F) Dt (DDMMYYYY)	341	DATE	5		Packed date DDMMYYYY to SQL Date
Unsigned Pk(F) Dt (MMDDYY)	344	DATE	4		Packed date MMDDYY to SQL Date
Unsigned Pk(F) Dt (MMDDYYYY)	345	DATE	5		Packed date MMDDYYYY to SQL Date
Unsigned Pk(F) Dt (YYMMDD)	342	DATE	4		Packed date YYMMDD to SQL Date
Unsigned Pk(F) Dt (YYYYMMDD)	343	DATE	5		Packed date YYYYMMDD to SQL Date
Unsigned Pk(F) Dt 2000(DDMMYY)	346	DATE	4		Packed date DDMMYY date window to SQL Date
Unsigned Pk(F) Dt 2000(MMDDYY)	348	DATE	4		Packed date MMDDYY date window to SQL Date
Unsigned Pk(F) Dt 2000(YYMMDD)	347	DATE	4		Packed date YYMMDD date window to SQL Date

Unsigned Pk Date (DDMMYY)	229	DATE	3		Unsigned Packed Decimal Date in the specified format.
Unsigned Pk Date (DDMMYYYY)	230	DATE	3		Unsigned Packed Decimal Date in the specified format.
Unsigned Pk Date (YYMMDD)	231	DATE	3		Unsigned Packed Decimal Date in the specified format.
Unsigned Pk Date (YYYYMMDD)	232	DATE	4		Unsigned Packed Decimal Date in the specified format.
Unsigned Pk Date (MMDDYY)	233	DATE	3		Unsigned Packed Decimal Date in the specified format.
Unsigned Pk Date (MMDDYYYY)	234	DATE	4		Unsigned Packed Decimal Date in the specified format.
Unsigned Pk Date 2000 (DDMMYY)	235	DATE	3		Unsigned Packed Decimal Date in the specified format.
Unsigned Pk Date 2000 (MMDDYY)	237	DATE	3		Unsigned Packed Decimal Date in the specified format.
Unsigned Pk Date 2000 (YYMMDD)	236	DATE	3		Unsigned Packed Decimal Date in the specified format.
Unsigned Quadword	250	BIGINT	8		
Unsigned Word	166	INTEGER	2		Unsigned Word
Unsigned Word 2 bytes (BE)	291	SMALLINT	2		

Unsigned Word (BE) -> Double	293	DOUBLE	2		
Varbinary	207	VARBINARY	-1		
Variable Length Int -> Dbl	169	CHAR	-1		Integer data type with implied decimal precision. This type is designed to handle irregular integer sizes (3 bytes, 6 bytes, etc).
VarLen Int -> Double	290	DOUBLE	-1		
Variable Length Int -> Int (Deprecated)	168	CHAR	-1		Integer data type. This type is designed to handle irregular integer sizes (3 bytes, 6 bytes, etc).
VarLen Integer -> Decimal	323	DECIMAL	-1		Binary variable length integer to SQL Decimal
Vector Timestamp	355	TIMESTAMP	14		Unsigned short array of year, month, day, hour, minute, second, and 1000ths of a second
VISTA Date (Julian 12/31/1919)	386	DATE	2		Julian Date offset from 12/31/1919.
Vixen Date	438	Date	6		
VMS 4 Byte Date	74	DATE	4		VMS Date with time value truncated.

VMS C Date	96	TIMESTAMP	4		Date, stores number of seconds since 1/1/1970.
VMS C Reverse Date	97	TIMESTAMP	4		VMS C Date with bytes reversed.
VMS Date	27 (1002 for backward compatibility.)	DATE	8	Maxcim Date	VMS Binary Date/Time (8 Bytes) converts to an ODBC Date.
VMS Date (6 byte)	158	DATE	6		
VMS Date/Time	29	TIMESTAMP	8		VMS Binary Date/Time (8 Bytes) converts to an ODBC Timestamp.
VMS Double	6	DOUBLE	8		
VMS Double -> Currency	405	CURRENCY	8		Scale is 0 to 4. Precision = 19.
VMS G Float	114 (1018 for backward compatibility.)	DOUBLE	8	G Float	G Float - 64-bit precision
VMS H Float	115 (1019 for backward compatibility.)	DOUBLE	16	H Float	H Float-128-bit precision
VMS Single	5 (1018 for backward compatibility)	REAL	4	Maxcim Real	Single precision floating point (4 bytes)
VMS Time	28	TIME	8		VMS Binary Date/Time (8 Bytes) converts to an ODBC Time.

VMS Date (6 byte)	158	DATE	6		VMS Date stored in 6 bytes (fractional second precision is truncated).
VMS S Float	159 (1019 for backward compatibility.)	REAL	4	S Float	VMS IEEE Single-precision floating point
VMS T Float	160	DOUBLE	8	T Float	VMS IEEE Double-precision floating point
VMS X Float	156	DOUBLE	16	X Float	VMS IEEE X Float
Winery Long Date	93	DATE	2		Special date for customer.
Winery Plus Long Date	94	DATE	2		Special date for customer.
Winery Plus Word Date	92	DATE	2		Special date for customer.
Winery Word Date	91	DATE	2		Special date for customer.
Word (BE)	262	SMALLINT	2		
Word (BE) -> Double	265	SQL_DOUBLE	2		
Word -> Currency	70 (1031 for backward compatibility.)	CURRENCY	2	DEC ODBC Word	The Word Currency data type is identical to the Word Decimal data type, except the data is converted to a SQL Decimal instead of a SQL Double. This may provide greater precision.
Word -> Double (2)	98	DOUBLE	2		Word converted to double with implied decimal point.

Word -> Float	12	REAL	2		
Word	11 (1027 for backward compatibility.)	SMALLINT	2	VMS Word IntegerPIC S9(X1) COMPwhere X1 is between 1 and 4.	Word Integer (2 bytes)
Word Numeric -> Decimal	310	DECIMAL	2		Two-byte signed value converts to a SQL Decimal
Zero Filled Text	4 (1039 for backward compatibility.)	CHAR	-1	Maxcim Zero-filled String	A zero-filled string is NOT filled if it contains an alpha character, otherwise, it is a zero-filled numeric string. (1-30000 chars)
Zoned Numeric -> Decimal	313	DECIMAL	-1		String of digits with sign overpunched in lowest (right) digit converts to SQL Decimal
ZONED NUMERIC -> Double	20 (1015 with fraction > 0 for backward compatibility.)	DOUBLE	-1	VMS Dibol	Converts to a double precision floating point.
ZONED NUMERIC -> Integer	21 (1015 with fraction = 0 for backward compatibility.)	LONG	-1	VMS Dibol	Converts to a long.
ZZ[deprecated] Adabas Natural Date Unpacked	409	DATE	6		
ZZ[deprecated] Adabas Natural Time Unpacked	410	TIME	7		

ZZ[deprecated] Adabas Natural Timestamp	419	TIMESTAMP	7		
ZZ[deprecated] VarLen Int	328	DECIMAL	-1		

CONNX Data Type Import Codes

Data Type Length Information

The negative numbers used in the Length and SQL Length columns have the following meanings:

Length Value	Meaning
-1	Variable length
-2	Variable length
-3	Variable length
-4	Variable length
-5	Variable length
-6	Variable length
-7	Variable length
-8	Special POISE data type - Variable length
-9	Variable length
-10	Variable length
-11	Variable length
-12	Variable length
-13	Variable length
-14	Variable length
-15	Variable length

CONNX Data Types - Import Codes 1-42

The following table contains the available data types for databases supported by CONNX, organized by import code number. Several of the data types have more than one import code, supplied for compatibility with earlier versions of CONNX.

Import Code	CONNX Data Type	SQL Data Type	SQL Length	VMS Equivalent Data Types	Description
1 (1000)	Text (Right Space Padded)	CHAR	-1	Text Maxcim Yes/No	Alphanumeric and symbols. (1-30000 chars); same as Text but with right space padding.
2	Text (Null Terminated)	VARCHAR	-1		
3	Text (Expandable)	VARCHAR	-1		Alphanumeric and symbols. (1-30000 chars).
4 (1039)	Zero Filled Text	CHAR	-1	Maxcim Zero-filled String	A zero-filled string is NOT filled if it contains an alpha character, otherwise, it is a zero-filled numeric string. (1-30000 chars)
5 (1018 for backward compatibility)	VMS Single	REAL	4	VMS Single Real Maxcim Single Real	Single precision floating point (4 bytes).
6 (1017)	VMS Double	DOUBLE	8	VMS Double Real Maxcim Double Real	Double precision floating point (8 bytes).
7 (1026)	Byte	TINYINT	1	VMS Byte Integer Maxcim Byte Integer	Byte Integer (1 byte).
8 (1021)	Byte -> Float	REAL	4	VMS Byte Integer with implied decimal.	Byte Integer with an implied decimal place (1 byte).
9 (1001 for backward compatibility)	Byte Bit (1/0)	BIT	1		Evaluates Byte to a True or False value, and represents it as a bit.
10	Byte Bit Text(Yes/No)	CHAR	3		Evaluates Byte to a True or False value, and represents it as Yes or No.
11 (1027)	Word	SMALLINT	2	VMS Word Integer PIC S9(X1) COMP where X1 is between 1 and 4.	Word Integer (2 bytes)
12 (1022)	Word -> Float	REAL	4	VMS Word Integer with implied decimal	Word Integer with an implied decimal place (2 bytes).

				PIC S9(X1)V9(X2) COMP where X1+X2 is between 1 and 4.	
13 (1028)	Longword	INTEGER	4	VMS Longword Integer PIC S9(X1) COMP where X1 is between 5 and 9.	Longword Integer (4 bytes)
14 (1023)	Longword -> Double	DOUBLE	8	VMS Longword Integer with implied decimal PIC S9(X1)V9(X2) COMP where X1+X2 is between 5 and 9.	Longword Integer with an implied decimal place (4 bytes).
15 (1029)	Quadword -> Char	VARCHAR	20	VMS Quadword Integer PIC S9(X1)V9(X2) COMP where X1+X2 is between 10 and 20.	Quadword Integer (8 bytes) converts to a SQL String.
16 (1024)	Quadword -> Char (DP)	VARCHAR	20	VMS Quadword Integer with implied decimal PIC S9(X1)V9(X2) COMP where X1+X2 is between 10 and 20.	Quadword Integer with an implied decimal place (8 bytes) converts to a SQL String.
17	Text -> Integer	INTEGER	4		
18	Text -> Double	DOUBLE	8		
19	Text (Formatted) -> Double	DOUBLE	8		
20	Zoned Numeric -> Double	DOUBLE	8		
21	Zoned Numeric -> Integer	LONG	4		
22 (1014 with fraction > 0)	Packed Decimal -> (Double)	DOUBLE	8	VMS Packed Decimal PIC S9(X1)V9(X2) COMP-3	Packed Decimal (1-255 bytes) converts to a double precision floating point.
23 (1014 with fraction = 0)	Packed Decimal -> (Integer)	LONG	4	VMS Packed Decimal PIC S9(X1)V9(X2) COMP-3	Packed Decimal (1-255 bytes) converts to a long.
24 (1011 with	Signed Overpunch -> Double	DOUBLE	8	VMS Numeric String left	Signed Overpunch (1-255 digits)

fraction > 0)				overpunched sign	converts to a double precision floating point.
25 (1011 with fraction = 0)	Signed Overpunch -> Integer	LONG	4	VMS Numeric String left overpunched sign	Signed Overpunch (1-255 digits) converts to a long
26	(TXT) VMS Date/Time	VARCHAR	23		
27 (1002)	VMS Date	DATE	6	Maxcim Date	VMS Binary Date/Time (8 Bytes) converts to an ODBC Date.
28	VMS Time	TIME	6		VMS Binary Date/Time (8 Bytes) converts to an ODBC Time.
29	VMS Date/Time	TIMESTAMP	16		VMS Binary Date/Time (8 Bytes) converts to an ODBC Timestamp.
30 (1052)	Reverse VMS Date	DATE	6	Maxcim Date	VMS Binary Date/Time (8 Bytes) converts to an ODBC Date. All bytes are stored in reverse order.
31 (1053)	Reverse VMS Time	TIME	6		VMS Binary Date/Time (8 bytes) converts to an ODBC Time. All bytes are stored in reverse order.
32 (1054)	Reverse VMS Date/Time	TIMESTAMP	16		VMS Binary Date/Time (8 bytes) converts to an ODBC Timestamp. All bytes are stored in reverse order.
33 (1040)	Cognos PHDate	DATE	6		Cognos Powerhouse Date (2 bytes) converts to an ODBC Date.
34 (1041)	Cognos JDate	DATE	6		Cognos Powerhouse Julian Date (2 Bytes) converts to an ODBC Date
35	Binary	BINARY	-1	Maxcim RFA Maxcim Byte Array	The binary field is not converted. It provides access to raw, unaltered data in an RMS file.
36	Binary (Text)	CHAR	-2		Each byte of data is returned as a 2-character hexadecimal representation of the original byte of data.
37	Basic Plus Single	REAL	4	Basic Plus Single	The Basic Plus Single data type is

					identical to the Single data type, with all bytes in reverse order.
38	Basic Plus Double	DOUBLE	8	Basic Plus Double	The Basic Plus Double Datatype is identical to the Double Datatype, with all bytes in reverse order.
39	Basic Plus Word	SMALLINT	2	Basic Plus Word Integer	The Basic Plus Word Integer data type is identical to the Word Integer data type, with all bytes in reverse order.
40	Basic Plus Long	INTEGER	4		
41 (1042)	Text Date (DDMMYY)	DATE	6		Text Date in the specified format. Handles years between 1900 and 1999.
42 (1043)	Text Date (DDMMYYYY)	DATE	6		Text Date in the specified format.

CONNX Data Types - Import Codes - 43 - 79

Import Code	CONNX Data Type	SQL Data Type	SQL Length	VMS Equivalent Data Types	Description
43 (1044 for backward compatibility)	Text Date (YYMMDD)	DATE	6		Text Date in the specified format. Handles years between 1900 and 1999.
44 (1055 for backward compatibility)	Text Date (YYYYMMDD)	DATE	6		Text Date in the specified format.
45 (1050 for backward compatibility)	Text Date (MMDDYY)	DATE	6		Text Date in the specified format. Handles years between 1900 and 1999.
46 (1051 for backward compatibility)	Text Date (MMDDYYYY)	DATE	6		Text Date in the specified format.
47 (1035 for backward compatibility)	Long Date (DDMMYY)	DATE	6		Longword Date in the specified format. Handles years between 1900 and 1999.
48 (1036 for backward compatibility)	Long Date (DDMMYYYY)	DATE	6		Longword Date in the specified format.
49 (1037 for backward compatibility)	Long Date (YYMMDD)	DATE	6		Longword Date in the specified format. Handles years between 1900 and 1999.
50 (1038 for backward compatibility)	Long Date (YYYYMMDD)	DATE	6		Longword Date in the specified format.
51 (1048 for backward compatibility)	Long Date (MMDDYY)	DATE	6		Longword Date in the specified format. Handles years between 1900 and 1999.
52 (1049 for backward compatibility)	Long Date (MMDDYYYY)	DATE	6		Longword Date in the specified format.
53	Pack Date (DDMMYY)	DATE	6		Packed Decimal in the specified format. Handles years between 1900 and 1999.
54	Pack Date (DDMMYYYY)	DATE	6		Packed Decimal in the specified format.
55	Pack Date (YYMMDD)	DATE	6		Packed Decimal in the specified format. Handles

					years between 1900 and 1999.
56	Pack Date (YYYYMMDD)	DATE	6		Packed Decimal in the specified format.
57	Pack Date (MMDDYY)	DATE	6		Packed Decimal in the specified format. Handles years between 1900 and 1999.
58	Pack Date (MMDDYYYY)	DATE	6		Packed Decimal in the specified format.
59	Text Date 2000 (DDMMYY)	DATE	6		Text Date in the specified format. Handles years between 1941 and 2040.
60	Text Date 2000 (YYMMDD)	DATE	6		Text Date in the specified format. Handles years between 1941 and 2040.
61	Text Date 2000 (MMDDYY)	DATE	6		Text Date in the specified format. Handles years between 1941 and 2040.
62	Long Date 2000 (DDMMYY)	DATE	6		Longword Date in the specified format. Handles years between 1941 and 2040.
63	Long Date 2000 (YYMMDD)	DATE	6		Longword Date in the specified format. Handles years between 1941 and 2040.
64	Long Date 2000 (MMDDYY)	DATE	6		Longword Date in the specified format. Handles years between 1941 and 2040.
65	Pack Date 2000 (DDMMYY)	DATE	6		Packed Decimal Date in the specified format. Handles years between 1900 and 1999.
66	Pack Date 2000 (YYMMDD)	DATE	6		Packed Decimal Date in the specified format. Handles years between 1900 and 1999.
67	Pack Date 2000 (MMDDYY)	DATE	6		Packed Decimal Date in the specified format. Handles years between 1900 and 1999.
68	Maxcim Key Date	DATE	6		Maxcim special key date format.
69 (1046 for backward compatibility)	Text Currency	CURRENCY	21		Numeric money value stored as ASCII text with implied decimal point.
70 (1031 for backward compatibility)	Word -> Currency	CURRENCY	21	DECODBC Word	The Word Currency data type is identical to the Word Decimal data type, except the data is converted to a SQL Decimal instead of a SQL Double. This may provide greater precision.
71 (1032 for	Longword Currency	CURRENCY	21		

backward compatibility)					
72 (1033 for backward compatibility)	Quadword -> Currency	CURRENCY	21	DEC ODBC Quadword	The Quadword Currency data type is identical to the Quadword Decimal data type, except the data is converted to a SQL Decimal instead of a SQL Double. This may provide greater precision.
73 (1047 for backward compatibility)	Packed Decimal -> Currency	CURRENCY	21		The Packed Decimal Currency data type is identical to the Packed Decimal data type, except the data is converted to a SQL Decimal instead of a SQL Double. This may provide greater precision.
74	VMS 4 Byte Date	DATE	6		VMS Date with truncated Time value.
75	Reverse VMS 4 Byte Date	DATE	6		VMS 4-byte Date with bytes reversed.
76	CODA Year	SMALLINT	2		Word, a year is represented in a number offset from the year 1900. A value of 50 would represent 1950, and a value of 103 would represent the year 2003.
77	Reverse CODA Year	SMALLINT	2		Coda Year with bytes reversed.
78	BMS Date	DATE	6		3-byte Date for BMS.
79	BMS Reverse Date	DATE	6		3-byte Reverse Date for BMS.

CONNX Data Types - Import Codes - 80-118

Import Code	CONNX Data Type	SQL Data Type	SQL Length	VMS Equivalent Data Types	Description
80	Noah Date	DATE	6		
81	Noah Inverse Date	DATE	6		Site-specific.
82	Noah Time	TIME	4		Site-specific.
83	Byte Bit#0 (2^0)	BIT	1		Evaluates Byte to a True or False value, and represents it as a bit.
84	Byte Bit#1 (2^1)	BIT	1		This represents bit #2 (2^0) from the byte.
85	Byte Bit#2 (2^2)	BIT	1		This represents bit #3 (2^0) from the byte.
86	Byte Bit#3 (2^3)	BIT	1		This represents bit #4 (2^0) from the byte.
87	Byte Bit#4 (2^4)	BIT	1		This represents bit #5 (2^0) from the byte.
88	Byte Bit#5 (2^5)	BIT	1		This represents bit #6 (2^0) from the byte.
89	Byte Bit#6 (2^6)	BIT	1		This represents bit #7 (2^0) from the byte.
90	Byte Bit#7 (2^7)	BIT	1		This represents bit #8 (2^0) from the byte.
91	Winery Word Date	DATE	6		Site-specific.
92	Winery Plus Word Date	DATE	6		Site-specific. Bytes in reverse order.
93	Winery Long Date	DATE	6		Site-specific.
94	Winery Plus Long Date	DATE	6		Site-specific. Bytes in reverse order.
95	POISE Double	DOUBLE	8		ASCII double left space padded with explicit decimal point and sign.
96	VMS C Date	TIMESTAMP	16		Date, stores number of seconds since 1/1/1970.
97	VMS C Reverse Date	TIMESTAMP	16		VMS C Date with bytes reversed.
98	Word -> Double (2)	DOUBLE	8		Word converted to double with implied decimal point.
99	24bit Pascal Integer	INTEGER	4		24-bit Pascal Integer - 3 bytes.
100	PACKED (A) Decimal -> Double	DOUBLE	8		Packed Decimal of type (A).
101	PACKED (A) Decimal -	LONG	4		Packed Decimal of

	> Integer				type (A).
102	PACKED (E) Decimal -> Double	DOUBLE	8		Packed Decimal of type (E).
103	PACKED (E) Decimal -> Integer	LONG	4		Packed Decimal of type (E).
104	PACKED (F) Decimal -> Double	DOUBLE	8		Packed Decimal of type (F).
105	PACKED (F) Decimal -> Integer	LONG	4		Packed Decimal of type (F).
106	DAI History Date	DATE	6		Date in YYMMDD format subtracted from 999999.
107	DAI History Time	DATE	6		Time in HHMMSS format subtracted from 999999.
108	DAI History Year	SMALLINT	2		Year in YYYY format subtracted from 9999.
109	DAI History Period	SMALLINT	2		Period in PP format subtracted from 99.
110	Left Separate -> Double	DOUBLE	8		Left Separate Left Separate Double
111	Left Separate -> Integer	LONG	4		Left Separate
112	Right Separate -> Double	DOUBLE	8		Right Separate
113	Right Separate -> Integer	LONG	4		Right Separate
114 (1018 for backward compatibility)	VMS G Float	DOUBLE	8	G Float	G Float - 64-bit precision
115 (1019 for backward compatibility)	VMS H Float	DOUBLE	8	H Float	H Float - 128-bit precision
116	Allport Julian Date (4 byte)	DATE	6		Site-specific.
117	Allport Time	TIME	6		Site-specific.
118	National Compressed Double	DOUBLE	8		Compressed Double data type.

CONNX Data Types - Import Codes - 119-225

Import Code	CONNX Data Type	SQL Data Type	SQL Length	VMS Equivalent Data Types	Description
119	ADL Math	DOUBLE	8		Site-specific.
120	ADL MMIS Number	CHAR	23		Site-specific.
121	ADL Date	DATE	6		Site-specific.
122	ADL GL Number	DOUBLE	8		Site-specific.
123	ADL Type Code	VARCHAR	5		Site-specific.
124	ADL Zip Code	LONG	4		Site-specific.
125	ADL Old Math	VARCHAR	16		Site-specific.
126	Flex Text	CHAR	-1		
127	Flex Double	DOUBLE	8		
128	Flex Longword	INTEGER	4		
129	Flex Date	DATE	6		
130	Flex Binary	BINARY	-1		
131	String Tinyint	TINYINT	1		
132	String Smallint	SMALLINT	2		
133	String Integer	INTEGER	4		
134	String Quadword	VARCHAR	20		
135	String Single	REAL	4		
136	String Double	DOUBLE	8		
137	RDB Date VMS	TIMESTAMP	16		
138	RDB Date ANSI	DATE	6		
139	RDB Time	TIME	6		
140	RDB Timestamp	TIMESTAMP	16		
141	Interval Day	DOUBLE	8		
142	Interval Hour	DOUBLE	8		
143	Interval Minute	DOUBLE	8		
144	Interval Date Day/Second	DOUBLE	8		
145	Interval Date Hour	DOUBLE	8		
146	Interval Date Hour/Minute	DOUBLE	8		

147	Interval Date Hour/Second	DOUBLE	8		
148	Interval Date Minute	DOUBLE	8		
149	Interval Date Minute/Second	DOUBLE	8		
150	Interval Date Month	DOUBLE	8		
151	Interval Date Second	DOUBLE	8		
152	Interval Date Year	DOUBLE	8		
153	Interval Date Year/Month	DOUBLE	8		
154	Morse Date	DATE	6		Site-specific.
155	Allport Julian Date (2 byte)	DATE	6		Site-specific.
156	VMS X Float	DOUBLE	8		X Float VMS IEEE X Float
157	Reverse VMS Date (6 byte)	DATE	6		Reversed VMS Date stored in 6 bytes (fractional second precision is truncated).
158	VMS Date (6 byte)	DATE	6		VMS Date stored in 6 bytes (fractional second precision is truncated).
159	VMS S Float	REAL	4	S Float	VMS IEEE single-precision floating point.
160	VMS T Float	DOUBLE	8	T Float	VMS IEEE double-precision floating point.
161	KCS Compressed Integer	INTEGER	4		Site-specific.
162	KCS Phone	CHAR	10		Site-specific.
163	KCS Zip Code	CHAR	9		Site-specific.
164	KCS Time (2 byte)	TIME	6		Site-specific.
165	KCS Time (3 bytes)	TIME	6		Site-specific.
166	Unsigned Word	INTEGER	4		Unsigned Word
167	KCS Date (3 bytes)	DATE	6		Site-specific.
168	Variable Length Int -> intr	CHAR	20		Integer data type. This type is designed to handle irregular integer sizes (3 bytes, 6 bytes, etc.)
169	Variable Length Int -> Dbl Decimal	CHAR	20		Integer date type with implied decimal precision. This type is designed to handle irregular integer sizes (3 bytes, 6 bytes,

					etc.)
170	DIBOL Date (5 bytes)	DATE	6		
171	Quadword -> Double	DOUBLE	8		
172	ZZ(deprecated) DB2 Binary	BINARY	-1		
173	ZZ(deprecated DB2 Variable	VARBINARY	-1		
174	ZZ(deprecated) DB2 Long	LONG VARBINARY	-1		
175	ZZ*deprecated) DB2 CHAR	CHAR	-1		
176	ZZ(deprecated) DB2 VARCHAR	VARCHAR	-1		
177	ZZ(deprecated) DB2 LONGVARCHAR	LONGVARCHAR	-1		
178	Word 2 Bytes	SMALLINT	2		
179	DB2 Word Big Endian	SMALLINT	2		
180	Longword 4 bytes	INTEGER	4		
181	DB2 Longword Big Endian	INTEGER	4		
182	DB2 Numeric	NUMERIC	-6		
183	DB2 Numeric -> Integer	LONG	4		
184	DB2 Packed Decimal	DECIMAL	-7		
185	DB2 Packed Decimal -> Integer	INTEGER	4		
186	DB2 ISO Date (YYYY-MM-DD)	DATE	6		
187	DB2 USA Date (mm/dd/yyyy)	DATE	6		
188	DB2 Eur Date (dd.mm.yyyy)	DATE	6		
189	DB2 JIS Date (YYYY-MM-DD)	DATE	6		
190	DB2 ISO Time (hh.mm.ss)	TIME	6		
191	DB2 USA Time (hh:mm xM)	TIME	6		
192	DB2 Eur Time (hh.mm.ss)	TIME	6		
193	DB2 JIS Time (hh:mm:ss)	TIME	6		
194	DB2 Timestamp	TIMESTAMP	16		
195	DB2 IEEE 4-byte float	REAL	4		
196	DB2 IEEE Big E 4-byte	REAL	4		

	float				
197	DB2 IEEE 8-byte float	DOUBLE	8		
198	DB2 IEEE Big E 8-byte float	DOUBLE	8		
199	DB2 Mainframe 4-byte float	REAL	4		
200	DB2 Mainframe 8-byte float	DOUBLE	8		
201	Decimal	DECIMAL	-6		
202	CLOB	LONG VARCHAR	-1		
203	BLOG	LONG VARBINARY	-1		
204	Oracle Date	TIMESTAMP	16		
205	Oracle Rowid	BINARY	-1		
206	Oracle MLSLabel	BINARY	-1		
207	Varbinary	VARBINARY	-1		
208	OLE DB Date	DATE	6		
209	OLE DB Time	TIME	6		
210	OLE DB Timestamp	TIMESTAMP	16		
211	OLE DB Number	NUMERIC	-6		
212	OLE DB WSTR	UNICODE	-1		
213	OLE DB Decimal	DECIMAL	-6		
214	Unsigned Long	INTEGER	4		
215	Unsigned Byte	TINYINT	1		
216	OLE DB real	REAL	4		
217	OLE DB double	DOUBLE	8		
218	Text (Space Padded) No Nulls	CHAR	-1		
219	Big Double	DOUBLE	8		
220	Big Double Text	CHAR	48		
221	BOSS Date CCYY	DATE	6		Site-specific, 6-byte, Y2K-compliant date.
222	BOSS Year CCYY	SMALLINT	2		Site-specific, 4-byte, Y2K-compliant year.
223	BOSS Julian CCYY	DATE	6		Site-specific, 5-byte, Y2K-compliant year.
224	Packed Decimal COMP6 -> Double	DOUBLE	8	VMS Packed Decimal PIC S9(X1)V9(X2) COMP-3	Packed Decimal (1-255 bytes) converts to a double precision floating point; same as Packed Decimal (Double) but

					unsigned.
225	Packed Decimal COMP6 -> Integer	LONG	4	VMS Packed Decimal PIC S9(X1)V9(X2) COMP-3	Packed Decimal (1-255 bytes) converts to a Long; same as Packed Decimal (Integer) but unsigned.

CONNX Data Types - Import Codes - 226-265

Import Code	CONNX Data Type	SQL Data Type	SQL Length	VMS Equivalent Data Types	Description
226	Text (Left Space Padded)	CHAR	-1		
227	Text Left Space Pad -> Int	INTEGER	4		
228	Text Left Space Pad -> Dbl	DOUBLE	8	TextMaxcim Yes/No	Alphanumeric and symbols. (1-30000 chars); same as Text but with left space padding.
229	Unsigned Pk Date (DDMMYY)	DATE	6		Unsigned Packed Decimal Date in the specific format.
230	Unsigned Pk Date (DDMMYYYY)	DATE	6		Unsigned Packed Decimal Date in the specific format.
231	Unsigned Pk Date (YYMMDD)	DATE	6		Unsigned Packed Decimal Date in the specific format.
232	Unsigned Pk Date (YYYYMMDD)	DATE	6		Unsigned Packed Decimal Date in the specific format.
233	Unsigned Pk Date (MMDDYY)	DATE	6		Unsigned Packed Decimal Date in the specific format.
234	Unsigned Pk Date (MMDDYYYY)	DATE	6		Unsigned Packed Decimal Date in the specific format.
235	Unsigned Pk Date 2000 (DDMMYY)	DATE	6		Unsigned Packed Decimal Date in the specific format.
236	Unsigned Pk Date 2000 (YYMMDD)	DATE	6		Unsigned Packed Decimal Date in the specific format.
237	Unsigned Pk Date 2000 (MMDDYY)	DATE	6		Unsigned Packed Decimal Date in the specific format.
238	Decimal (Formatted with DP)	DECIMAL	-1		
239	Cognos PHDate 2000	DATE	6		Cognos Powerhouse Date (2 bytes) converts to an ODBC Date.
240	BOSS Julian YYYYJJJ	DATE	6		Site-specific, 5-byte, Y2K-compliant Julian date.
241	EDS Julian Date YYDDD	DATE	6		Site-specific.

242	EDS Date YYMMDD	DATE	6		Site-specific.
243	Compressed String	VARCHAR	-1		All spaces removed from the text field.
244	Reverse Text Date YYYYMMDD	DATE	6		Subtracts 10000000 from the date, stored physically as a character field, yielding strings that, if sorted, result in dates in descending order.
245	Reverse Long Date YYYYMMDD	DATE	6		Subtracts 10000000 from the date, stored physically as a Long, yielding strings that, if sorted, result in dates in descending order.
246	Text Null Terminated (I)	VARCHAR	-1		
247	POISE Record Number	INTEGER	4		A special data type used only in the record field of any POISE key file.
248	POISE Validate Code	VARCHAR	-8	none	A short code descriptor is populated in the CDD comment field. When the field is displayed using CONNX. the code descriptor prefixes the data value.
249	Pk Centry Date (CYYMMDD)	DATE	6		
250	Unsigned Quadword	BIGINT	8		
251	CISAM Packed Decimal	DECIMAL	-9		
252	Access Currency	CURRENCY	21		
253	Text (Nullable) -> Integer	INTEGER	4		
254	Text (Nullable) -> Double	DOUBLE	8		
255	Text DB2 (Right Space Padded)	CHAR	-1		
256	Text DB2 (Null Terminated)	VARCHAR	-1		
257	POISE Text Date 2000 (MMDDYY)	DATE	6		POISE Text Date in the specified format. Handles years between 1941 and 2040.
258	POISE Text Date (MMDDYYYY)	DATE	6		POISE Text Date in the specified format.
259	POISE Double (Right Padded)	DOUBLE	8		ASCII double left space padded with explicit decimal point

					and sign. Used to convert the data type in POISE RMS key field(s).
260	POISE Julian Date	DATE	6		2 byte, Julian format (1=01/01/1970)
261	POISE Time	TIME	6		2-byte field contains minutes after midnight.
262	Word (big endian)	SQL_SMALLINT	2		
263	Longword	SQL_INTEGER	4		
264	Single Prec float (Mainframe)	REAL	4		
265	Double Prec float (Mainframe)	DOUBLE	8		

CONNX Data Types - Import Codes - 266-356

Import Code	CONNX Data Type	SQL Data Type	SQL Length	VMS Equivalent Data Types	Description
266	Word (BE) -> Double	DOUBLE	8		
267	Longword Decimal Big Endian	DOUBLE	8		
268	Binary (Text) Reversed	CHAR	-2		
269	Comment Prefix	VARCHAR	-8		
270	Float IEEE 4-byte	REAL	4		
271	Float IEEE 4-byte (BE)	REAL	4		
272	Double IEEE 8-byte	DOUBLE	8		
273	Double IEEE 8-byte (BE)	DOUBLE	8		
274	CISAM Packed Decimal 1	DOUBLE	8		
275	CISAM Packed Decimal 2	VARCHAR	128		
276	PROIV Date	DATE	6		
277	Astrazeneca Interval	VARCHAR	18		
278	CISAM Zoned Numeric -> Double	DOUBLE	8		
279	CISAM Zoned Numeric -> Integer	LONG	4		
280	CISAM DIBOL Date (5 bytes)	DATE	6		
281	Time (Text HHMM)	TIME	6		
282	Time (Text HHMMSS)	TIME	6		
283	Pinnacle Date	DATE	6		
284	Quadword (BE) -> Char	VARCHAR	20		
285	Quadword (BE) -> Char (DP)	VARCHAR	20		
286	VarLen Int (BE) -> Char	CHAR	20		
287	VarLen Int (BE) -> Char DP	CHAR	20		
288	Quadword (BE) -> Double	DOUBLE	8		
289	VarLen Int (BE) -> Double	DOUBLE	8		
290	VarLen Int -> Double	DOUBLE	8		
291	Unsigned Word 2 bytes (BE)	SMALLINT	2		
292	Unsigned Long 4 bytes	INTEGER	4		

	(BE)				
293	Unsigned Word Decimal (BE)	DOUBLE	8		
294	Unsigned Long (BE) -> Double	DOUBLE	8		
295	Unsigned Quad (BE) -> Char	VARCHAR	20		
296	Unsigned Quad (BE) -> Char(DP)	VARCHAR	20		
297	Unsigned VarLen Int(BE)->Char	CHAR	20		
298	Unsigned VarLen Int(BE)->Char(DP)	CHAR	20		
299	Unsigned Quadword Double (BE)	DOUBLE	8		
300	Unsigned VarLen Int (BE) -> Dbl	DOUBLE	8		
301	Unsigned Byte Double	DOUBLE	8		Single unsigned byte value converts to a SQL Double
302	PostgreSQL Timestamp	TIMESTAMP	16		
303	PostgreSQL Date	DATE	6		
304	PostgreSQL Time	TIME	8		
305	CA Window	VARCHAR	155		
306	CA Bit Flag	INTEGER	4		Customer-specific (1 byte of individual bit flags)
307	4C Date	DATE	6		Date in the form of days since 12/31/1799 converts to SQL Date
308	Gestepargne Date	DATE	6		
309	Byte Numeric -> Decimal	DECIMAL	5		Single signed byte value converts to a SQL Decimal
310	Word Numeric -> Decimal	DECIMAL	7		Two-byte signed value converts to a SQL Decimal
311	Longword -> Decimal	DECIMAL	12		Three-byte signed value converts to a SQL Decimal
312	Quadword -> Decimal	DECIMAL	21		Four-byte signed value converts to a SQL Decimal
313	Zoned Numeric-> Decimal	DECIMAL	-6		String of digits with sign overpunched in lowest (right) digit converts to SQL Decimal
314	Packed Decimal -> Decimal	DECIMAL	-7		Packed decimal converts to SQL

					Decimal
315	Signed Overpunch -> Decimal	DECIMAL	-6		Overpunched decimal converts to SQL Decimal
316	PACKED (A) Decimal -> Decimal	DECIMAL	-7		Packed decimal to SQL Decimal, value of 'A' denotes negative
317	PACKED (E) Decimal -> Decimal	DECIMAL	-7		Packed decimal to SQL Decimal, value of 'E' denotes negative
318	PACKED (F) Decimal -> Decimal	DECIMAL	-7		Packed decimal to SQL Decimal, value of 'F' denotes negative
319	Left Separate -> Decimal	DECIMAL	-10		Character to SQL Decimal with leading sign
320	Right Separate -> Decimal	DECIMAL	-10		Character to SQL Decimal with trailing sign
321	National Compressed -> Decimal	DECIMAL	-11		Customer-specific numeric to SQL Decimal
322	Flex Numeric	DECIMAL	-1		
323	VarLen Integer -> Decimal	DECIMAL	-11		Binary variable length integer to SQL Decimal
324	POISE Txt (Right Pad) -> Decimal	DECIMAL	-1		Customer-specific character to SQL Decimal
325	PACKED Decimal COMP6 -> Decimal	DECIMAL	-13		Unsigned packed decimal to SQL Decimal
326	CISAM Zoned Numeric -> Decimal	DECIMAL	-6		
327	Quadword (BE) -> Decimal	DECIMAL	21		
328	VarLen Int -> Decimal	DECIMAL	-11		Integer (2, 4, 8 bytes) to SQL Decimal (string) (DEPRECATED)
329	Unsigned Quadword(BE) -> Dec	DECIMAL	21		
330	VarLen Int (BE) -> Decimal	DECIMAL	-11		
331	Unsigned VarLen Int(BE)->Dec	DECIMAL	-12		
332	Text -> Decimal	DECIMAL	-6		Character to SQL Decimal
333	Unsigned Byte -> Decimal	DECIMAL	5		Unsigned byte number to SQL

					Decimal
334	Unsigned Word (BE) -> Decimal	DECIMAL	7		
335	Unsigned Long (BE) -> Decimal	DECIMAL	12		
336	Word (BE) -> Decimal	DECIMAL	7		
337	Longword (BE) -> Decimal	DECIMAL	12		
338	RDB Date VMS (String)	TIMESTAMP	16		
339	Mark BIB Number	LONG	4		
340	Unsigned PK(F) Dt (DDMMYY)	DATE	6		Packed date DDMMYY to SQL Date
341	Unsigned PK(F) Dt (DDMMYYYY)	DATE	6		Packed date DDMMYY to SQL Date
342	Unsigned PK(F) Dt (YYMMDD)	DATE	6		Packed date YYMMDD to SQL Date
343	Unsigned PK(F) Dt (YYYYMMDD)	DATE	6		Packed date YYYYMMDD to SQL Date
344	Unsigned PK(F) Dt (MMDDYY)	DATE	6		Packed date MMDDYY to SQL Date
345	Unsigned PK(F) Dt (MMDDYYYY)	DATE	6		Packed date MMDDYYYY to SQL Date
346	Unsigned PK(F) Dt 2000(DDMMYY)	DATE	6		Packed date DDMMYY date window to SQL Date
347	Unsigned PK(F) Dt 2000(YYMMDD)	DATE	6		Packed date YYMMDD date window to SQL Date
348	Unsigned PK(F) Dt 2000(MMDDYY)	DATE	6		Packed date MMDDYY date window to SQL Date
349	CISAM Zoned Decimal	DECIMAL	-6		
350	Word Big Endian Date (2 bytes)	DATE	6		
351	Word Big Endian Time (4 bytes)	TIME	6		
352	Word Big Endian Timestamp	TIMESTAMP	16		
353	Timestamp12 (YYYYMMDDHHMM)	TIMESTAMP	16		Character Timestamp with year, month, day, hour, and minute
354	Timestamp14 (YYYYMMDDHHMMSS)	TIMESTAMP	16		Character Timestamp with year, month, day, hour, minutes, and second

355	Vector Timestamp	TIMESTAMP	16		Unsigned short of year, month, day, hour, minute, second, and 1000ths of a second
356	Allport Julian Date String	DATE	6		Character string Julian (jumeric) date offset from 1/1/1970

CONNX Data Types - Import Codes - 357-406

Import Code	CONNX Data Type	SQL Data Type	SQL Length	VMS Equivalent Data Types	Description
357	TNRD Julian Date	Date	6		Customer-specific Julian data offset from 1/1/1975
358	TNRD Rev Julian Date	Date	6		Customer-specific reverse Julian data offset from 1/1/1975
359	TNRD Identification number	Varchar	-1		Customer-specific suffix delimited ID number
360	TNRD Surname	Varchar	-1		Customer-specific suffix delimited Surname
361	TNRD First Name and Initial	Varchar	-1		Customer-specific suffix delimited First name and initial
362	TNRD Care of	Varchar	-1		Customer-specific suffix delimited Care of
363	TNRD Street Address Number	Varchar	-1		Customer-specific suffix delimited Street Address Number
364	TNRD Street Address Name	Varchar	-1		Customer-specific suffix delimited Street Address Name
365	TNRD St Address City/Province	Varchar	-1		Customer-specific suffix delimited Street Address City and Province
366	TNRD St Address Postal Code	Varchar	-1		Customer-specific suffix delimited Street Address Postal Code
367	TNRD Home Phone	Varchar	-1		Customer-specific suffix delimited Home Phone
368	TNRD Work Phone	Varchar	-1		Customer-specific suffix delimited Work Phone
369	TNRD Alt Address Num & Street	Varchar	-1		Customer-specific suffix delimited Alternate Street Address Number and Name
370	TNRD Alt Address City & Prov	Varchar	-1		Customer-specific suffix delimited Alternate Address City and Province
371	TNRD Alt Address Postal Code	Varchar	-1		Customer-specific suffix delimited Alternate Address

					Postal Code
372	TNRD School	Varchar	-1		Customer-specific suffix delimited School
373	TNRD Parent or Guardian	Varchar	-1		Customer-specific suffix delimited Parent or Guardian
374	TNRD Previous Barcode	Varchar	-1		Customer-specific suffix delimited Previous Barcode
375	TNRD Next Barcode	Varchar	-1		Customer-specific suffix delimited Next Barcode
376	TNRD Xref See	Long	4		Customer-specific prefix delimited See
377	TNRD Xref See Also	Long	4		Customer-specific prefix delimited See Also
378	TNRD Xref See From	Long	4		Customer-specific prefix delimited See From
379	TNRD Xref See Also From	Long	4		Customer-specific prefix delimited See Also From
380	TNRD Notes	Varchar	-1		Customer-specific prefix delimited Notes
381	Marc BIB Decimal	Decimal	21		Customer-specific base 240 number conversion
382	Marc BIB Tag	Long	4		Customer-specific base 240 number conversion
383	Adonix Date	Date	6		Customer-specific Julian date offset from 12/31/1599
384	Text Left Space Pad -> FmtDbl	DOUBLE	8		
385	Numeric Data (YYMMDD)	Date	6		Three- byte date field where the date is taken from the decimal digits of the values. Thus a numeric value of 750824 is interpreted as August 24 1975. Note that the year field uses a century window where values < 25 are assigned century 2000, everything else 1900. The new function is far more customer specific: TPADateOfBirth(date, centuryCode) Inputs: date - base birth date,

					for example a field of Numeric Date type. centuryCode - integer value indicating century of birth, 8 = 1800, 9 = 1900, 0 == 2000, no other valid values.
386	VISTA Date (Julian 12/31/1919)	Date	6		Julian Date offset from 12/31/1919.
387	DBMS Key	CHAR	22		
388	Encapsulated Date (cyMMDD)	DATE	6		Byte 0 -> binary number representing the most significant two digits of a four digit year. Byte 1 -> binary number representing the least significant two digits of a four digit year. Thus, if given the first two bytes values of 20, 03, the year for this date are 2003. Bytes 2 & 3 are the numeric characters representing the month of the year. Bytes 4 & 5 are the numeric characters representing the day of the month. Thus a value of 0x010131313131 is a date of November (month 11) 11 (day 11) in the year 101.
389	Rev Encapsulated Date (MMDDcy)	DATE	6		Byte 0 -> binary number representing the most significant two digits of a four digit year. Byte 1 -> binary number representing the least significant two digits of a four digit year. Thus, if given the first two bytes values of 20, 03, the year for this date are 2003. Bytes 2 & 3 are the numeric characters representing the month of the year. Bytes 4 & 5 are the numeric characters representing the day of the month. Thus a value of 0x010131313131 is a

					date of November (month 11) 11 (day 11) in the year 101.
390	Text (Right Space Padded)	CHAR	-1		
391	Text (Expandable)	VARCHAR	-1		
392	Text Date (YYYYMM)	DATE	6		For YYYYMM , if YYYYMM was 200203, the SQL date would be March 1, 2003. The day of the month is always one (1).
393	Text Date 2000 (YYMM)	DATE	6		For YYMM , if YYMM was 0304, the SQL date would be April 1, 2003. The day of the month is always one (1). For YYMM , if YYMM was 2304, the SQL date would be April 1, 1923. The day of the month is always one (1). The break Year = 20 (For years below 20, add 2000; for years above 20, add 1900. The break year is configurable.)
394	Text Date (MMYYYY)	DATE	6		For MMYYYY , if MMYYYY was 032002, the SQL date would be March 1, 2003. The day of the month is always one (1).
395	Text Date 2000 (MMYY)	DATE	6		For MMYY , if MMYY was 0403, the SQL date would be April 1, 2003. The day of the month is always one (1). For MMYY , if MMYY was 0423, the SQL date would be April 1, 1923. The day of the month is always one (1). The break Year = 20 (For years below 20, add 2000; for years above 20, add 1900. The break year is

					configurable.)
396	Text Date (YYYY)	DATE	6		For YYYY, if YYYY was 2004, the SQL Date would be January 1, 2004. The day of the month is always one (1). The month is always January.
397	Text Date 2000 (YY)	DATE	6		for YY, if YY was 04, the SQL Date would be January 1, 2004. The day of the month is always one. The month is always January. For YY, if YY was 34, the SQL Date would be January 1, 1934. The day of the month is always one (1). The month is always January. The break Year = 20 (For years below 20, add 2000; for years above 20, add 1900. The break year is configurable.)
398	Text Date (MM)	DATE	6		For MM, if MM was 11, the SQL Date would be November 1, 2003. The year returned is the current year. The day of the month is always one (1).
399	Text Date (DD)	DATE	6		For DD, if DD = 30, the SQL Date would be January 30, 2003. The year returned is the current year. The month is always January.
400	Fairfield Text Date (MMDD)	DATE	6		For MMDD, if MM were 11 and DD were 20, then the SQL Date would be November 20, 1900. The year returned is always 1900.
401	Fairfield Text Date (MM)	DATE	6		For MMDD, if MM were 11 and DD were 20, then the SQL Date would be November 20, 1900. The year returned is always 1900.

402	Fairfield Text Date (DD)	DATE	6		For DD, if DD were 20, then the SQL Date would be January 20, 1900. The year returned in always 1900. The month returned is always January.
403	Adabas Natural Date	DATE	6		
404	Adabas Natural Time	TIME	6		
405	VMS Double -> Currency	Currency	21		
406	PostgreSQL Numeric -> Decimal	DECIMAL	-15		

CONNX Data Types - Import Codes 407-443

Import Code	CONNX Data Type	SQL Data Type	SQL Length	VMS Equivalent Data Types	Description
407	OLEDB VAR-WSTR	CHAR	-1		
408	MG Word Date	DATE	6		
409	ZZ[deprecated] Adabas Natural Date Unpacked	DATE	6		
410	ZZ[deprecated] Adabas Natural Time Unpacked	TIME	6		
411	Adabas Unpacked Decima->Integer	LONG	4		
412	Adabas Unpacked->Decimal	DECIMAL	-6		
413	Adabas Unpacked EBCDIC(Integer)	LONG	4		
414	Adabas Unpacked EBCDIC(Numeric)	DECIMAL	-6		
415	Cognos PHDate (BE)	DATE	6		
416	Cognos PHDate 2000 (BE)	DATE	6		
417	Adabas Text (VarChar)	VARCHAR	-1		
418	PROMIS Timestamp	Timestamp	16		
419	ZZ[deprecated] Adabas Natural Timestamp	TIMESTAMP	16		
420	Adabas Natural Timestamp	TIMESTAMP	16		
421	Adabas PACKED Decimal -> Integer	Long	4		
422	Adabas PACKED Decimal -> Decimal	Decimal	-7		
423	Adabas Natural Timestamp -> Date	Date	6		
424	Opus Date	Date	6		
425	Opus Date Packed	Date	6		
426	PostgreSQL Timestamp	Timestamp	16		
427	PostgreSQL Date	Date	6		
428	PostgreSQL Time	Time	6		
429	PostgreSQL Numeric (BE) -> Decimal	Decimal	-15		
430	PostgreSQL Numeric (7.4+) -> Decimal	Decimal	-15		

431	PostgreSQL Numeric (7.4+ BE) -> Decimal	Decimal	-15		
432	OPUS Unsigned PACKED -> Decimal	Decimal	21		
433	HCHD Serial Date	Date	6		
434	ANSI/ISO Timestamp	Timestamp	16		
435	Compufast Text Date (YYYYMMDD)	Date	6		
436	Globally Unique ID (GUID)	Char	38		Meant for SQL Server databases, but can be used with any database that supports GUID types.
437	Longword JDate (BE)	Date	6		
438	VIXEN Date	Date	6		The VIXEN Date is in the format DDMMYY. The year value is stored in such a way that if the first digit is 0-9, the decade is in 1900. If the first "digit" is from A-Z, then the decade is offset from 2000, i.e., 99 would be 1999, A9 would be 2009, and B9 would be 2019, etc.
439	Naftha Text	Char	-1		The Naftha Text is a custom datatype that does some character conversions for Arabic data (customer -specific).
440	CS Comment	Varchar	-1		Designed specifically for the Comment fields in Caldwell Spartan software.
441	Julian Timestamp (BE	TIMESTAMP	16		
442	PioTech Numeric	BIGINT	8		
443	PioTech Date	DATE	6		

CONNX Data Types - Import Codes 444-500

Import Code	CONNX Data Type	SQL Data Type	SQL Length	VMS Equivalent Data Types	Description
444	Adabas Text (LA VarChar)	Varchar	-1		
445	Adabas Text (L4 VarChar)	Varchar	-1		
446	Unicode Char (UTF-8)	Unicode	-2		
447	Unicode Char (UTF-8)	VarUnicode	-2		
448	Unicode LA VarChar (UTF-8)	VarUnicode	-2		
449	Unicode L4 VarChar (UTF-8)	VarUnicode	-2		
450	ODBC Bit (not used)	Bit	1		
451	ODBC Tiny Integer (not used)	TinyInt	1		
452	ODBC Date	Date	6		
453	ODBC Time	Time	6		
454	ODBC Timestamp	Timestamp	16		
455	Maestro Date (DD/MM/YYYY)	Date	6		
456	PRO IV Text Integer -> Date	Date	6		
457	TextDBMS Variable Text	VarChar	-1		
458	VMS Time with Precision	VarChar	18		
459	OLEDB Variable Length WSTR	VarUnicode	-1		
460	Unicode CLOB	VarUnicode	-1		
461	OLEDB WSTR (BE)	Unicode	-1		
462	OLEDB Variable Length WSTR(BE)	VarUnicode	-1		
463	Unicode CLOB (BE)	VarUnicode	-1		
464	ODBC Char	Char	-1		
465	Adabas Natural Date [Iterative]	Date	6		

466	ZZ [deprecated] Adabas Natural Date Unpacked [Iterative]	Date	6		
467	ZZ [deprecated] Adabas Natural Timestamp Unpacked [Iterative]	Timestamp	16		
468	Adabas Natural Timestamp [Iterative]	Timestamp	16		
469	Adabas Natural Timestamp -> Date [Iterative]	Date	6		
470	Text seconds since midnight (SSSSS)	Time	6		
471	PACKED Decimal 3.5 -> double	Double	8		
472	CONNXStore TID	VarChar	18		
473	CONNXStore TID (BE)	VarChar	18		
474	ODBC Small Integer	SmallInt	2		
475	ODBC Unsigned Small Integer	SmallInt	2		
476	ODBC Integer	Integer	4		
477	ODBC Unsigned Integer	Integer	4		
478	ODBC Big Integer	Bigint	8		
479	ODBC Unsigned Big Integer	Bigint	8		
480	ODBC Real	Double	8		
481	ODBC Double	Double	8		
482	Quadword (BE)	Bigint	4		
483	Quadword (BE) Unsigned	Bigint	4		
484	Quadword	Bigint	4		
485	FHP Julian Day Integer	Date	6		
486	FHP Julian Day String	Date	6		
487	APCO Small Date (3 bytes)	Date	6		
488	APCO Hex Nybbles	Integer	4		

489	APCO Small Date (4 bytes)	Date	6		
490	APCO 4C Date	Date	6		
491	Word Big Endian Date	Date	6		
492	DB2 Bigint	Bigint	4		
493	DB2 Bigint Big Endian	Bigint	4		

Chapter 17 - CONNX Reserved Keywords and Symbols

Reserved Keywords and Symbols

The list of CONNX reserved keywords and symbols begins below. If one of these keywords or symbols is used as a database object or field name, you must select the Use Quoted Delimiters option in the CONNX Data Dictionary Manager window.

Note: If a reserved keyword is used as a field or table name in a database object, CONNX automatically appends the names with the characters _col.

Important: *When the* ***Use Quoted Delimiters*** *option is selected, the keyword or symbol must be surrounded by quotation marks.*

Reserved Symbols and Reserved Keywords beginning with Symbols

-	/	<
(	;	<=
)	?	<>
*	@@IDENTITY	=
--*(	{	>
--*)	||	>=
*=	}	
,	+	

"A" Reserved Keywords

abs	adabas_reuse	any	authorization
acos	adabas_uisize	arraysearch	autocommit
adabas	adabas_uiunit	as	autocounter
adabas_dssize	add	asc	avedevmean
adabas_dsunit	all	ascii	avedevmedian
adabas_maxisn	all~cols	asin	avg
adabas_nisize	alter	atan	avg_distinct
adabas_niunit	and	atan2	

"B" Reserved Keywords

begin	bibdata	bin	bit_length
begintrans	bibxref	binary	block
between	bigint	bit	by

"C" Reserved Keywords

call	cnxforcechar	connxext	cot
cascade	cnxmemory	connxlength	count
case	cnxname	connxnullableindex	count_distinct
cast	cnxpreference	connxnullspace	create
catalog	cnxrpc	connxoffset	curdate
ceiling	cnxsleep	connxoption	current
char	coefvarpct	connxprecision	current_catalog
char_length	coefvarpctp	connxprefix	current_date
character	column	connxscale	current_schema
character_length	commit	connxtype	current_time
chr	concat	constraint	current_timestamp
cluster	connx_hash	convert	curtime
cnxforcebinary	connx_version	cos	

"D" Reserved Keywords

database	datepart	dayofweek	decode	description
date	datetime	dayofyear	default	difference
dateadd	day	dbid	degrees	distinct
datediff	dayname	dec	delete	double
datename	dayofmonth	decimal	desc	drop

"E" through "J" Reserved Keywords

else	float	group	index
encrypt	floor	having	inner
end	fn	hex	insert
execute	for	host	int
exists	foreign	hour	integer
exp	from	if	into
extract	getcursorname	ifempty	is
file	getdate	ifnull	is not
first	getvmsdate	ilike	join
fixed	grant	in	

"K" through "L" Reserved Keywords

key	last	locate	longvarbinary
kthlargest	lcase	log	lower
kthsmallest	left	log10	ltrim
kurtosis	length	long	
kurtosisp	like	longnvarchar	

"M" through "N" Reserved Keywords

max	minute	multimodalcount	nclob	ntusername
median	mod	multimodaloccur	netrics_match	null
microsoft	mode	national	not	number
middle	month	natural	now	numeric
min	monthname	nchar	nowait	nvarchar

"O" through "P" Reserved Keywords

octet_length	order	pow
odbc	outer	power
of	output	precision
oj	password	primary
on	pi	privileges
option	port	product
or	position	public

"Q" through "R" Reserved Keywords

quantile	range	replace	rmserrorp
quarter	real	restrict	rollback
quartile1	references	reverse	rotate
quartile3	remoteport	revoke	round
radians	remoteserver	right	rowcount
rand	repeat	rmserror	rtrim

"SA" through "SP" Reserved Keywords

schema	setrealtimedatabuffer	skewness	sortfirst
second	shortname	skewnessp	sortlast

select	sign	slike	sortmiddle
seqno	sin	smallint	soundex
set	size	some	space

"SQL_" Reserved Keywords

sql_binary	sql_integer	sql_timestamp	sql_tsi_quarter
sql_bit	sql_longvarbinary	sql_tinyint	sql_tsi_second
sql_char	sql_longvarchar	sql_tsi_day	sql_tsi_week
sql_date	sql_numeric	sql_tsi_frac_second	sql_tsi_year
sql_decimal	sql_real	sql_tsi_hour	sql_user_name
sql_double	sql_smallint	sql_tsi_minute	sql_varbinary
sql_float	sql_time	sql_tsi_month	sql_varchar

"SQ" through "SZ" Reserved Keywords

sqrt	strcmp	substring	suppression
stddev	stuff	sum	switch
stddevp	substr	sum_distinct	

"T" through "U" Reserved Keywords

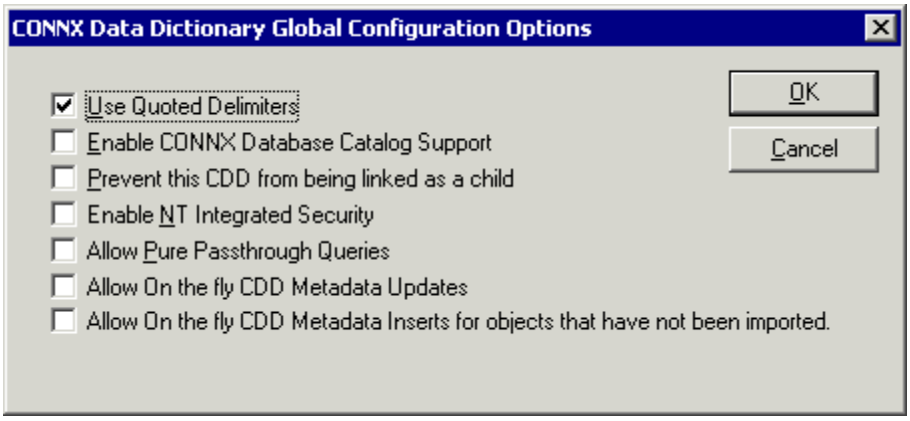
table	timestampadd	trimean	update
tablephysicallyexists	timestampdiff	truncate	upper
tan	tinyint	ucase	uqindex
then	to	union	use
time	tpadateofbirth	unique	user
timestamp	trim	unsigned	

"V" through "Z" Reserved Keywords

values	variancep	virtual	with
varbinary	varying	week	work
varchar	vendor	when	xpusername
variance	view	where	year

To enable the Use Quoted Delimiters option

1. Click the **Start** button, and then point to **All Programs.** Point to **CONNX Driver** and then click **CONNX Data Dictionary.**
2. Select the **Options** menu, and then click **CONNX SQL Options.** Select the **Use Quoted Delimiters** check box, and then click **OK.**



Related Topics

Creating CDDs
To create a CDD
Importing Existing Table Definitions

Connecting to multiple servers

Support for multiple JDBC Servers and/or Data servers is available on the CDD server name fields and the JDBC connection string.

<servername>[:<port>][!<servername>[:<port>]][!<servername>[:<port>]]...[!DISTRIBUTE]

Note: Each instance of <servername>[:<port>] is spearated by an exclaimation (!) mark.

For example - if you have two JDBC servers set up for redundancy - one on a server called PRIMARY, listening on port 7323

and another server called SECONDARY, listening on port 8754, the gateway server name would be:

Example:

PRIMARY:7323!SECONDARY:8754

When a list of servers is provided - CONNX will start with the first server listed. If the connection fails for that server, it will then serially loop through

all the servers from that point forward, wrapping back around to the beginning if necessary, until all options are exhausted, or a successful connection is established..

The DISTRIBUTE option may be added to the list of JDBC Servers and/or Data Servers. By using this option, CONNX will randomly choose a server from the list initially.

If the connection fails for that server, it will then serially loop through all the servers from that point forward, wrapping back around to the beginning if necessary, until all options
are exhausted, or a successful connection is established. This is used for load balancing purposes.

Example:

PRIMARY:7323!SECONDARY:8754!DISTRIBUTE

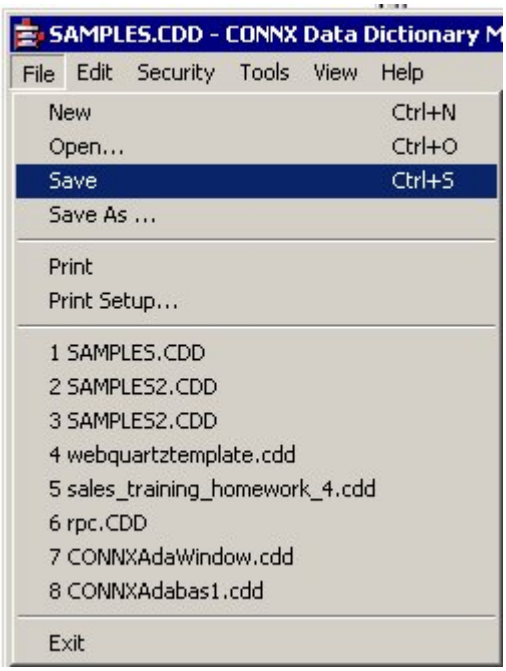
Note: The DISTRIBUTE keyword must be in all upper case letters.

Migrating to the new CONNX CDD format

The CONNX CDD has been converted to a new format in order to increase performance levels. If you are currently running a version of CONNX that is earlier than 10, you may find that you need to convert your CDDs to the new format (created in and added to version 9.0) for consistency and security.

To convert your CDDs to the new CONNX CDD format

Open the **CONNX Data Dictionary Manager.** Select a **CDD,** then under **File,** click **Save.**



Should a situation arise when you might not wish to change your CDD format, create a backup copy of your production CDD, and then create a copy that can be used by CONNX. For identification purposes, the file extension on converted CONNX CDDs is .vn9.

CONNX FTL™ (Fast Tuning Logic)

CONNX FTL™ automatically simplifies WHERE clause expressions used in SQL queries by eliminating elements deemed unnecessary to producing query results or by producing a more efficient expression that creates identical results. For example, for an SQL statement such as the following:

```
SELECT * FROM customers_dataflex
WHERE (((customers_dataflex.CUSTOMERCITY)='Seattle')
   AND ((customers_dataflex.CUSTOMERSTATE)='WA'))
   OR (((customers_dataflex.CUSTOMERCITY)="Bellevue")
   AND ((customers_dataflex.CUSTOMERSTATE)="WA"))
```

CONNX FTL is able to produce this simplified SQL statement:

```
SELECT * FROM Customers_dataflex t1
   WHERE ( t1.CUSTOMERSTATE ) = 'WA'
   AND
   ( ( t1.CUSTOMERCITY ) = 'Bellevue'
   OR
   ( t1.CUSTOMERCITY ) = 'Seattle' )
```

To prevent CONNX FTL from performing such optimizations, queries can be altered by including a SQL extended function tag that tells CONNX FTL not to optimize the query, as in the following example:

```
SELECT * FROM customers_dataflex
WHERE (((customers_dataflex.CUSTOMERCITY)='Seattle') AND
   ((customers_dataflex.CUSTOMERSTATE)='WA')) OR
   (((customers_dataflex.CUSTOMERCITY)="Bellevue") AND
   ((customers_dataflex.CUSTOMERSTATE)="WA")) {nosqloptimize}
```

The {nosqloptimize} tag, added at the end of the SQL statement, forces the query to process exactly as written.

CONNX FTL takes advantage of a technology called Sequential Interactive Synthesis (SIS), developed by the University of California at Berkeley. (See http://www-cad.eecs.berkeley.edu/~polis/ for a look at POLIS, A Framework for Hardware-Software Co-Design of Embedded Systems, which includes SIS.)

SQL View Clause Text Box

The SQL View Clause text box in the CONNX Data Dictionary Manager window uses the same syntax as the WHERE clause in an SQL statement.

Example:

```
SELECT * FROM testtable WHERE recordtype = "O"
```

Using this example, you would enter only:

```
recordtype = "O"
```

in the SQL View Clause text box. You do not need to enter the word WHERE in the SQL View Clause text box, as it is implied. If you enter the word WHERE, you will receive a syntax error when attempting to view the data for the table.

Related Topics

Creating CDDs

Importing Existing Table Definitions

CONNX Views

A CONNX View is a logical CONNX Data Dictionary table that is in essence a SQL statement. Views enhance the usability of CONNX by end-users through field aliases ('friendly' field names) and limiting the quantity of fields to only those that the end-user is concerned with. Views can specify filters for the returned result set. Views can define relationships to data from multiple tables, even multiple databases and other CONNX Views. Views are used to specify record-level security. A user may be granted access to a view without being granted access to the underlying table(s).
Important: *CONNX Views are updateable through front-end applications that use ADO, RDO, or DAO. However, they are not updateable through Microsoft Access unless you use a Microsoft Access passthrough query to construct the view. If you are going to be using Microsoft Access and want to be able to update CONNX Views, you must create a Microsoft Access passthrough query. Consult your Microsoft Access documentation for more information.*
NOTE: *CONNX Views support standard SQL functionality with the exception of aggregate functions (Group by, Sum, Avg, etc.), distinct queries, Union, and sub-queries. Insert, Update, and Delete statements are only permitted against CONNX views that are composed of a single table.*

The following is a sample CONNX View. Notice that SQL comments have been added to document the view. The view logic is as follows:

The CONNX sample ORDER and CUSTOMER tables are joined together, their relationship is defined in the where clause. Only specific fields from each table are chosen for display, including a new computed field with the alias "Ext Price". Only order records for customers whose CUSTOMERSTATE field contains an abbreviation for a state in the Northwest Territory --WA, OR, MT, ID, or CA -- are selected.

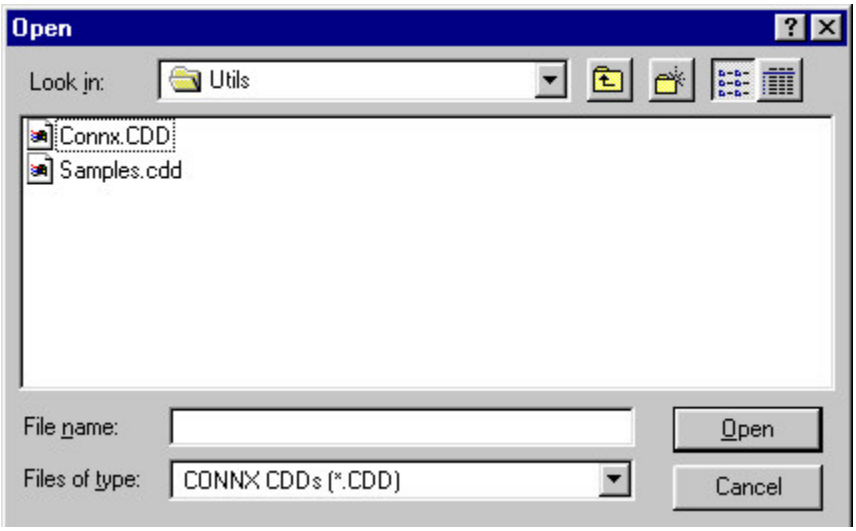
```
/*This view was requested by Johnathon Jones on 3/1/2001. He executes
   this view daily to see orders for the Northwest Territory. */
SELECT
ORDERS_RMS.orderid as 'Order' /* Order Number */,
ORDERS_RMS.customerid as 'Cust Id' /* Customer Identification */,
CUSTOMERS_RMS.customername as 'Name' /* Name of Customer*/,
CUSTOMERS_RMS.customerstate as 'ST' /* State Ordered by */,
ORDERS_RMS.orderdate as 'Ord Date' /* Date Ordered */,
ORDERS_RMS.productid as 'Product' /* Product number */,
PRODUCTS_RMS.productname as 'Description' /* Product Description */,
ORDERS_RMS.productquantity as 'Qty' /* order quantity */,
PRODUCTS_RMS.productprice as 'Price' /* price per unit */,
(ORDERS_RMS.productquantity * PRODUCTS_RMS.productprice) as 'Ext Price'
   /* Calculate extended price) */
FROM ORDERS_RMS, CUSTOMERS_RMS, PRODUCTS_RMS /* Tables included in view
   */
WHERE ORDERS_RMS.customerid=CUSTOMERS_RMS.customerid AND
ORDERS_RMS.productid=PRODUCTS_RMS.productid and
   CUSTOMERS_RMS.customerstate in ('WA', 'OR', 'MT', 'ID', 'CA') /*
   Join tables together and select only Northwest states */
```

Related Topics

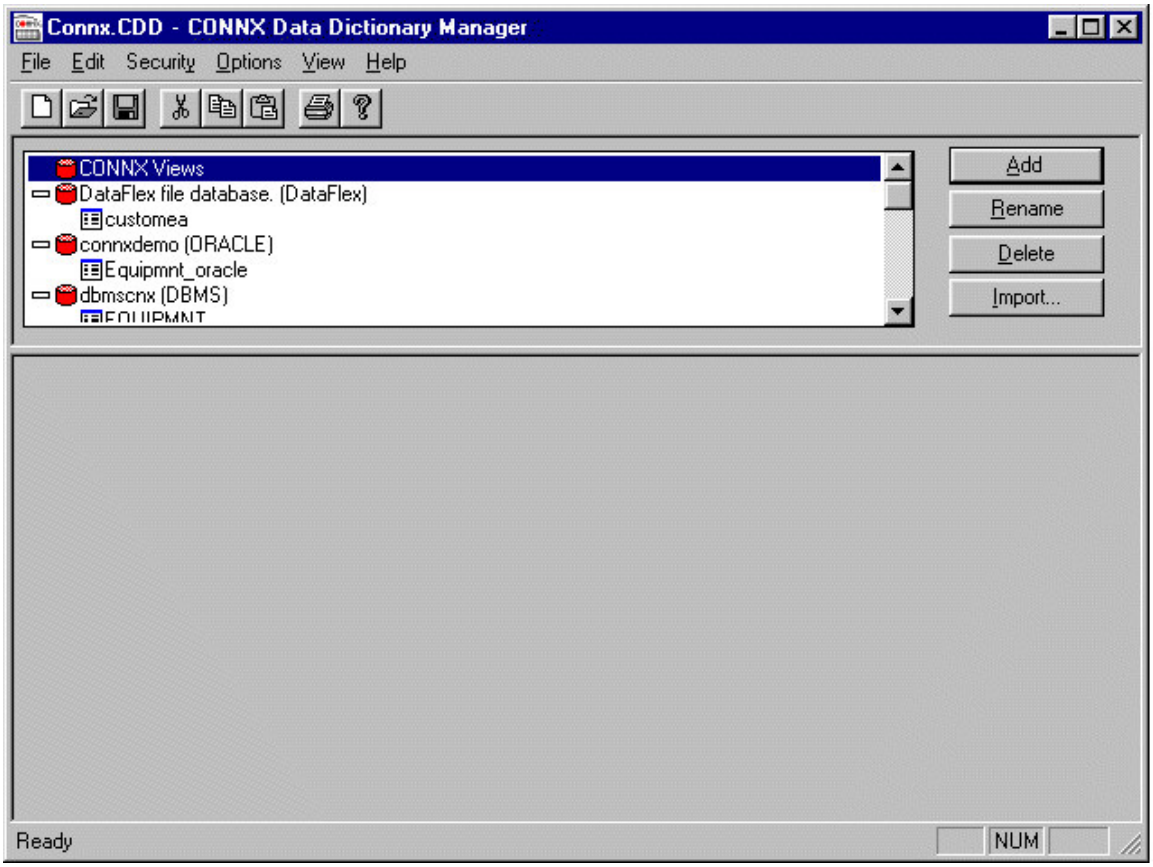
- To create a CONNX View manually in the CONNX Data Dictionary Manager
- CONNX Security Overview
- CREATE VIEW
- RMS View Text File Import Specification

To create a CONNX View manually in the CONNX Data Dictionary Manager

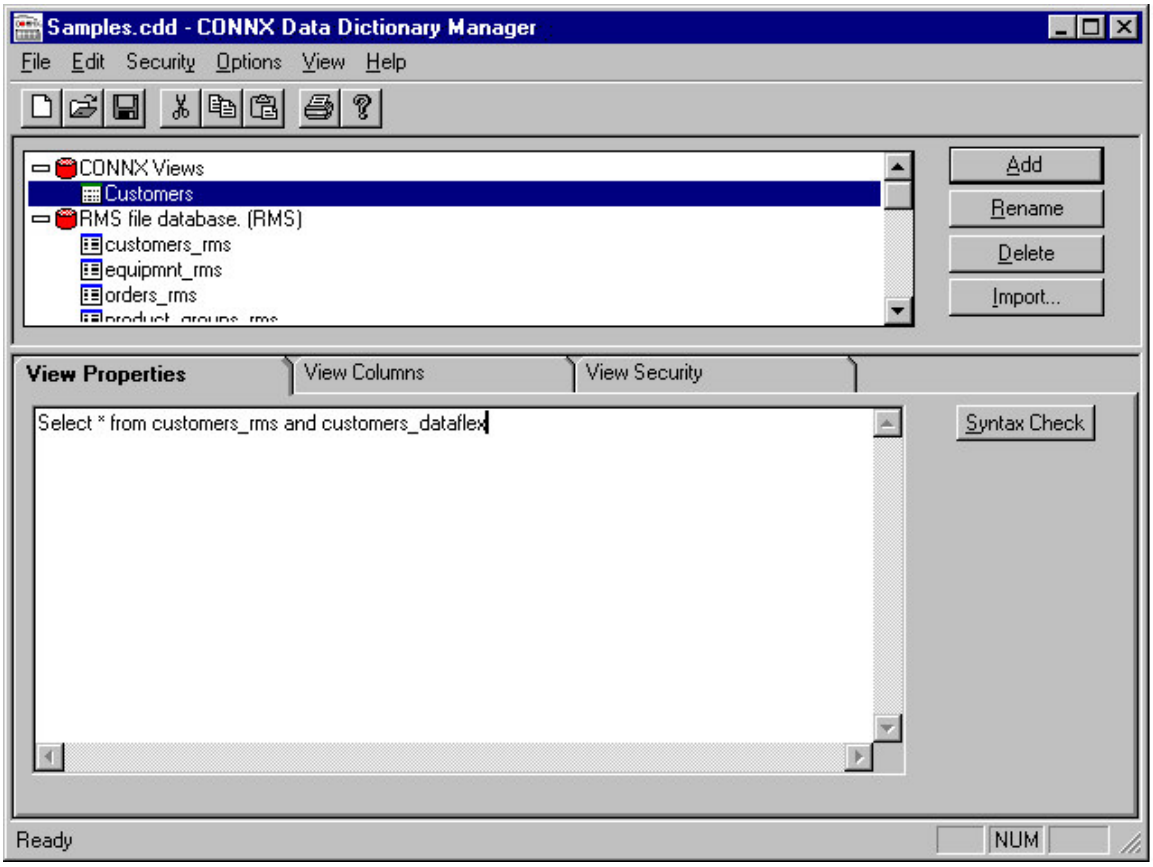
1. Click the **Start** button, and then point to **All Programs.** Point to **CONNX Driver** and then click **CONNX Data Dictionary.**
2. The **Open** dialog box appears.



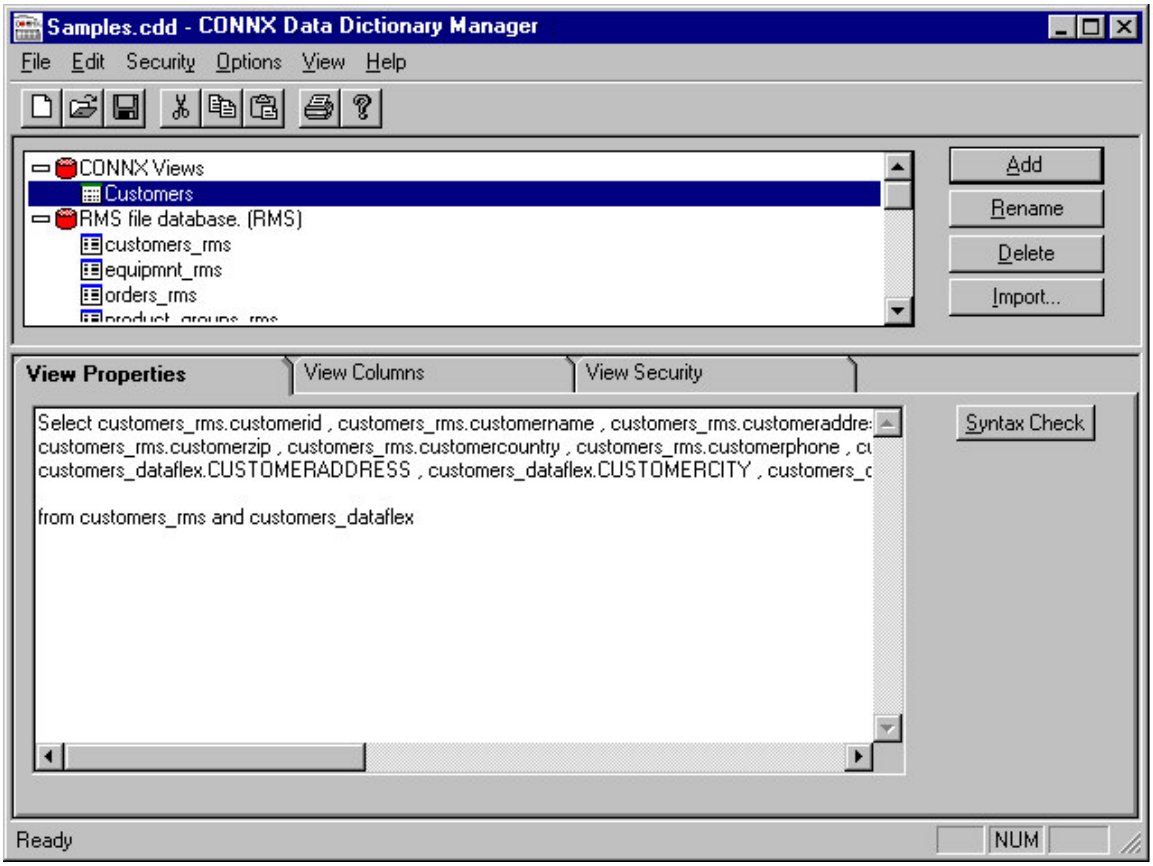
3. Select a CDD from the list, and then click the **Open** button.
4. The CONNX Data Dictionary Manager window appears. Click the **Add** button.



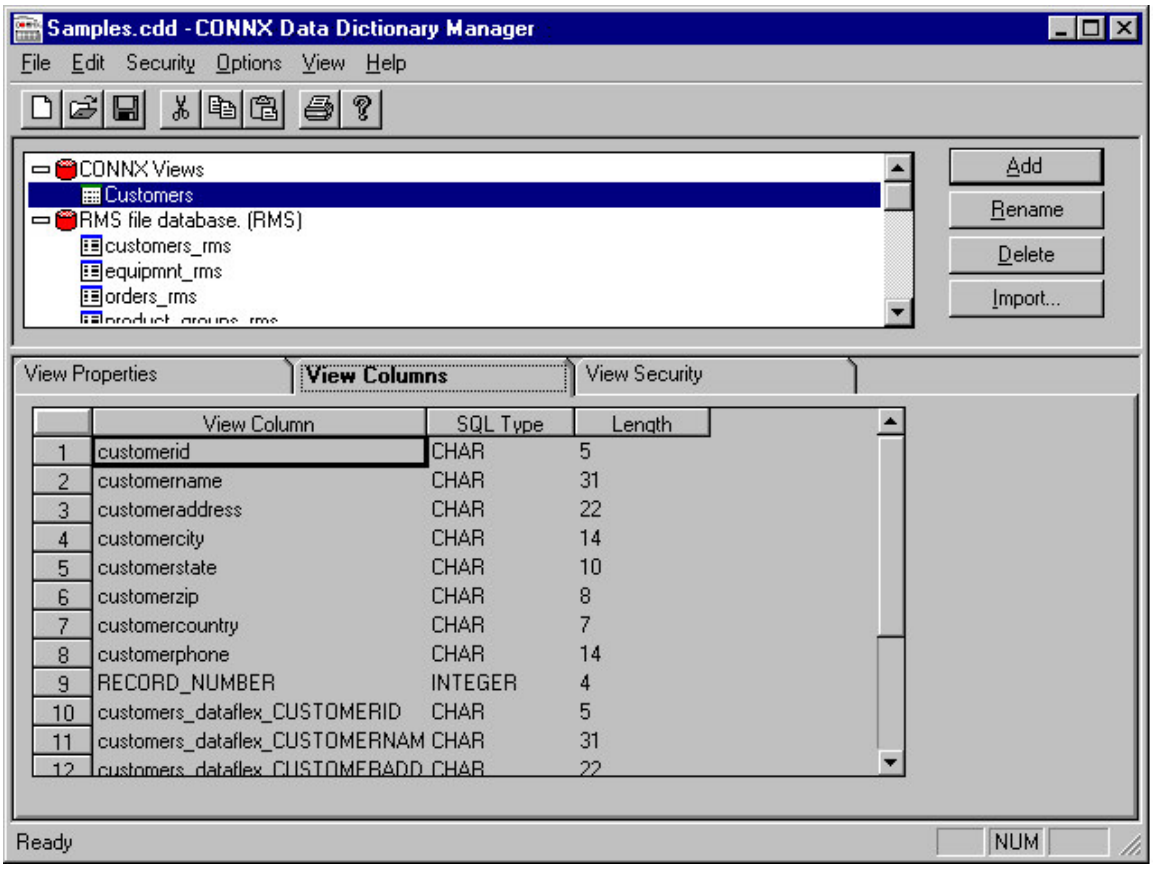
5. Type a name for the view in the **SQL Object Name** text box in the **Enter the Name of the New Table or View** dialog box. Do not use spaces or symbols in the object name. Select **View** in the **Object Type** list box, and then click the **OK** button.
6. The CONNX Data Dictionary Manager window appears. There are three tabs in the lower pane: View Properties, View Columns, and View Security. Type a SQL statement in the **View Properties** pane, and then click the **Syntax Check** button.



7. The results of the SQL statement appear on the View Properties tab.



8. Select the **View Columns** tab. The selected columns are listed along with their SQL data type and column length.


Samples.cdd - CONNX Data Dictionary Manager
File Edit Security Options View Help
CONNX Views
Customers
RMS file database. (RMS)
customers_rms
equipmnt_rms
orders_rms
Add
Rename
Delete
Import...
View Properties
View Columns
View Security
View Column
SQL Type
Length
1 customerid CHAR 5
2 customername CHAR 31
3 customeraddress CHAR 22
4 customercity CHAR 14
5 customerstate CHAR 10
6 customerzip CHAR 8
7 customercountry CHAR 7
8 customerphone CHAR 14
9 RECORD_NUMBER INTEGER 4
10 customers_dataflex_CUSTOMERID CHAR 5
11 customers_dataflex_CUSTOMERNAM CHAR 31
Ready
NUM

Adding security to a CONNX View

The security levels in all CDD entries can be modified to protect specific types of data. The access rights of individuals or groups can also be modified within the CDD. Users or groups can be added or removed, passwords can be changed, and security levels can be added to specific views.

1. Select a view from the list box in the CONNX Data Dictionary Manager window.
2. Click the **View Security** tab in the lower pane and then click the **Add Restriction** button.
3. To add a restriction to a view, select the name of the user or group to restrict in the **User** list box.
4. To add a restriction to a column, select the name of the column to restrict access to in the **SQL Column** list box. Select **<All Columns>** if the security entry is to apply to the entire table. Select **<Everyone>** and **<All Columns>** on all tables in order to maintain the highest levels of security.
5. Select the check boxes in the remaining columns to define access rights for each user or group as described in the following table:

Definition of Access Rights

Access	Definition
Select	Read only - can run Select queries.
Update	Can update queries and can modify existing data but not add new data.
Insert	Can insert new data records.
Delete	Can delete data records.
Drop	Can drop tables and keys to remove completely.
Execute	Can execute stored procedures.
RFI	Can enable referential integrity. (Available in a future release.)

Important: Individual security entries take precedence over group security entries.

Related Topics

- CONNX Security Overview
- CONNX Views
- To create a CONNX View

Rotated Arrays (RMS and VSAM only)

Rotated Arrays enable the creation of a logical view of repeating data so that each column in the repeating segment appears as a unique row. A benefit to using rotated arrays is that aggregate functions, such as SUM, can be applied. The following is an example of a rotated array.

The record shown is an order summary.

Sales Order Summary

CustomerID	Jan Sales	Feb Sales	Mar Sales	April Sales	May Sales	June Sales
CUST876762	$400	$50	$20	$90	$98	$85

By treating the sales_month array as a rotated array, results are returned as follows:

Sales Order Summary Rotated Array

CustomerID	Month	Sales	CNXARRAYCOLUMN
CUST876762	Jan	$400	0
CUST876762	Feb	$50	1
CUST876762	Mar	$20	2
CUST876762	April	$90	3
CUST876762	May	$98	4
CUST876762	June	$85	5

Returning data in this form provides two benefits. First, many tools, including Microsoft Access and Visual Basic, cannot use ODBC tables containing more than 255 columns. If a table contains a 400-element array, it cannot be imported into Microsoft Access. However, by rotating the array, Access reads the array as a single column, thereby allowing manipulation of data. This has the effect of normalizing nonrelational data.

The pseudo-column `CNXARRAYCOLUMN` displayed in the above example is returned whenever the rotated array feature is used. This column represents the element number of the array for this row of data.

Secondly, mathematical functions such as SUM, AVG, MIN, and MAX are easily applied to numeric data in column format. In the following example, trying to return the sales average would be much more difficult without the rotated array.
Here is an example of an SQL statement used for a non-rotated array:

```
SELECT (JanSales + FebSales + MarSales + AprSales + MaySales +
   JuneSales + JulySales + AugSales + SeptSales + OctSales + NovSales +
   DecSales)/12 as SalesAverage from ORDER_SUMMARY
```

Here is the same example of an SQL statement used as a rotated array:

```
SELECT AVG(MonthSales) as SalesAverage from ORDER_SUMMARY
```

Related Topics

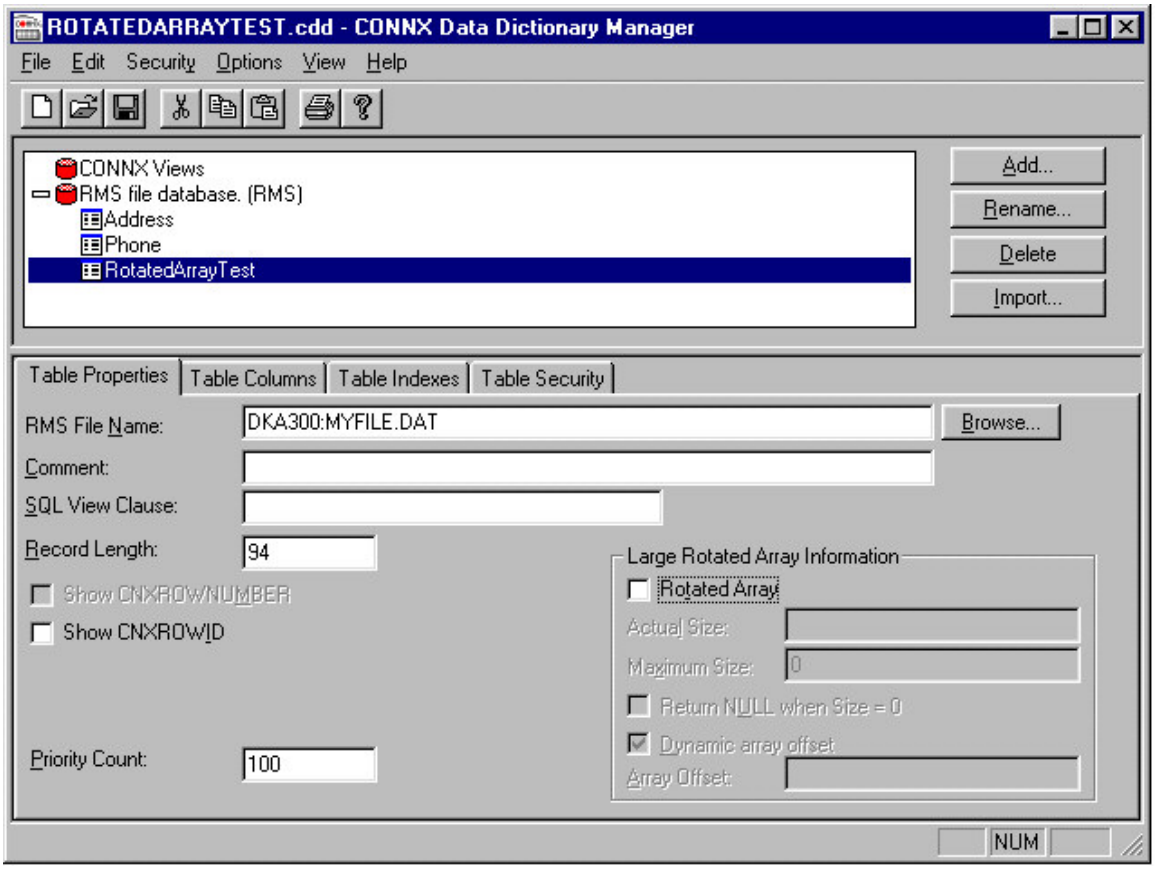
Configuring a rotated array

Using the Rotated Array Assistant

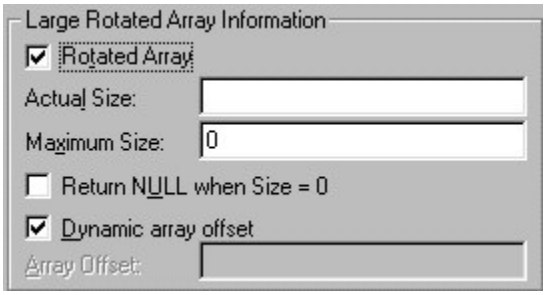
SCT-specific Non-standard Rotated Arrays

Configuring a rotated array

1. Select a table in the upper pane of the CONNX Data Dictionary Manager window.



2. Select the **Rotated Array** check box under **Large Rotated Array Information.**



3. Specify the actual size of the array. For fixed array sizes, type in the number of elements in the array. If the array size changes depending on another field in the record (a dynamic array), specify that field, or any valid SQL Expression that will determine the size of the array. You may use the following expression to determine the size of the array:

   header length + (array element counter (array_CTR) x array element length)

   For variable length arrays, the counter field (possibly ending in _CTR, depending on how it was initially set up in RMS and VSAM) appears in the CDD, listing the number of occurrences of the array.

4. The **Maximum Size,** which equals the total number of elements in the array, is entered automatically, but it can be adjusted. In fixed-length arrays, this number is the same as the actual size.
5. If a Null record is to be returned even when the array size is zero for a particular row, select the **Return NULL when Size = 0** check box.
6. If the record layout contains more than one variable length array, specify a **Dynamic array offset**, which is the starting position in the record for this array. This is usually a SQL calculation that adds the length of the fixed portion of the record layout to the size of the preceding variable length arrays. The sizes of the preceding arrays are calculated by multiplying the size of one instance of the array by the number of occurrences of the array.

Example 1 (An array with one element):

CustomerName	Phone	CNXARRAYCOLUMN
Nick Delmonico	(555) 333-4444	0
Nick Delmonico	(555) 333-5555	1
Nick Delmonico	(555) 333-6666	2

Length Array
10 Phone_0001
10 Phone_0002
10 Phone_0003

1 element = 10 bytes

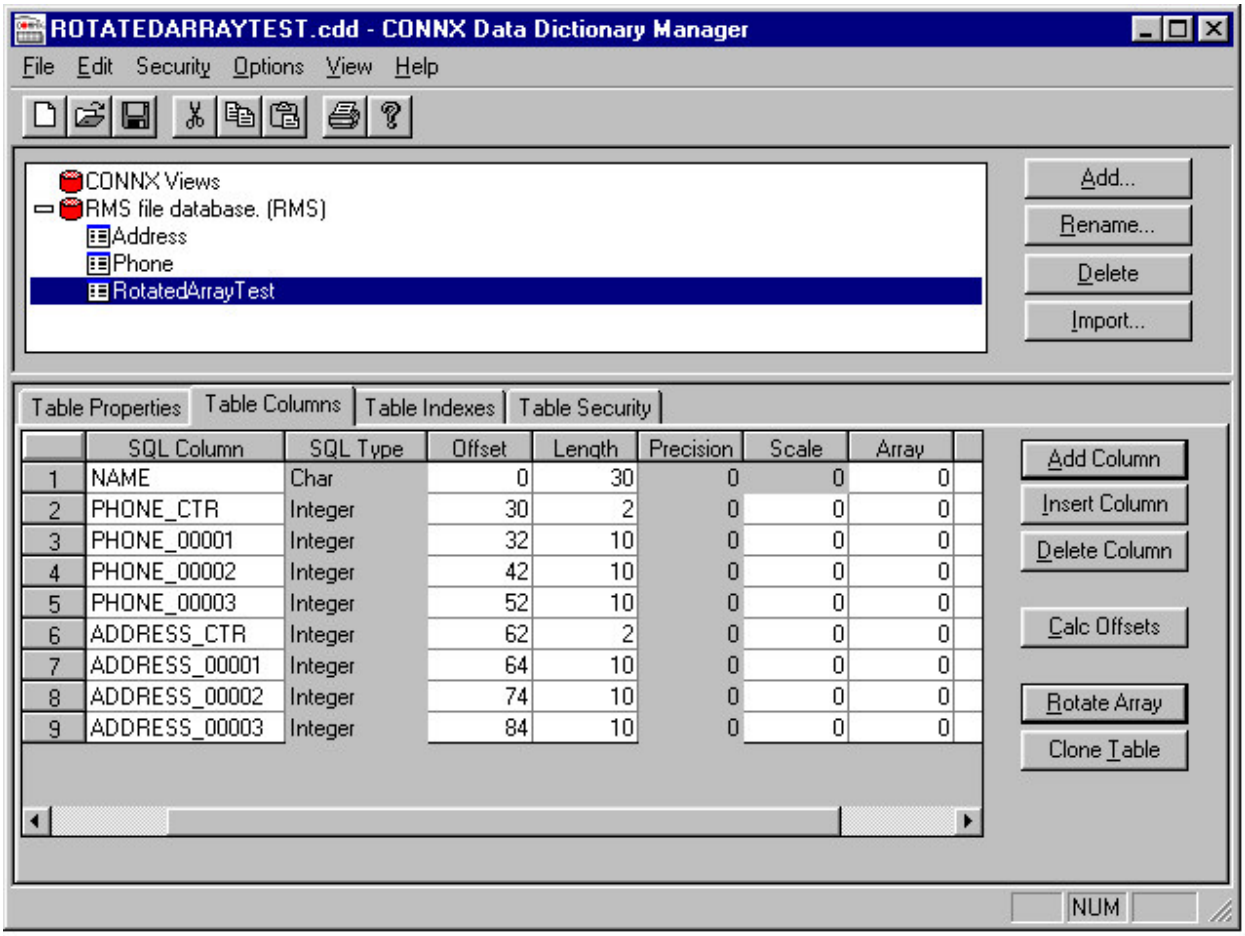
Example 2 (An array with two elements):

CustomerName	Address	CNXARRAYCOLUMN	City
Nick Delmonico	Suite 1A	0	North Bend
Nick Delmonico	555 West Elm	1	North Bend
Nick Delmonico	Apt C	0	Lincoln
Nick Delmonico	14 H Street	1	Lincoln

Length Array
20 Address_Line1_0001
20 Address_Line2_0001
10 City_0001
20 Address_Line1_0002
20 Address_Line2_0002
10 City_0002

1 element = 50 bytes

7. Click the **Table Columns** tab.



8. Specify the offset, length, and data type of the **first** element of the array in the **Offset, Length,** and **DataType** columns.

9. Specify the size of one complete element of the array in the **Array** field. Size is determined by record layout.

Example:

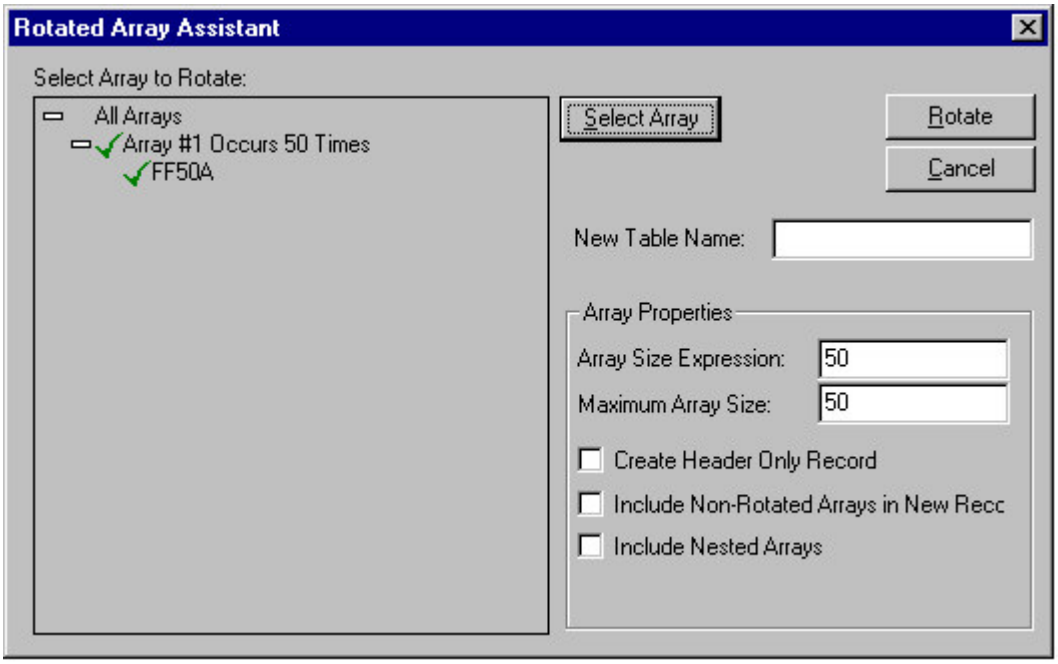
Column name Length
Address_CTR 2
Address_0001 10
Address_0002 10
Address_0003 10

The value of the array equals 10.

Using the Rotated Array Assistant

The Rotated Array Assistant is a wizard that simplifies the process of creating rotated arrays. The Rotated Array Assistant automatically creates a copy of the selected table and rotates the specified array.

1. Select a table in the upper pane of the CONNX Data Dictionary Manager window.
2. Click the **Table Columns** tab and then click the **Rotate Array** button. The **Rotated Array Assistant** dialog box appears.



3. Double-click on an array, or single-click to highlight the array, and then click the **Select Array** button.
4. Type a name for the new table in the **New table name** text box.Table-naming conventions are as follows:

1. Maximum size is 50 characters.
2. There must be no spaces in the name.
3. Table name cannot begin with a number.
4. Table name must be unique.
5. Select the check boxes beside the Array Properties described below, as required.

6. **Array Size Expression**
   This field defaults to the number of times the array is repeated. For variable length arrays there will be a counter field (possibly ending in _CTR) in the CDD, listing the number of occurrences of the array. Type this field name in the **Array Size Expression** field. The text box automatically displays the number of occurrences for arrays that have a constant number of occurrences.
7. **Maximum Array Size**
   Default value.
8. **Create Header Only Record**
   Creates a record that includes everything but the rotated arrays. All non-arrayed columns remain.
9. **Include non-rotated arrays in new record**
   Use this option for small arrays, which are not worth rotating. If left unchecked, only the rotated arrays appear.

10. **Include nested arrays**
    Use this option to include nested arrays (arrays which contain arrays within themselves).

6. Click the **Rotate** button. The new arrays appear in the CONNX Data Dictionary Manager window.

SCT Plus2000-specific Non-standard Rotated Arrays

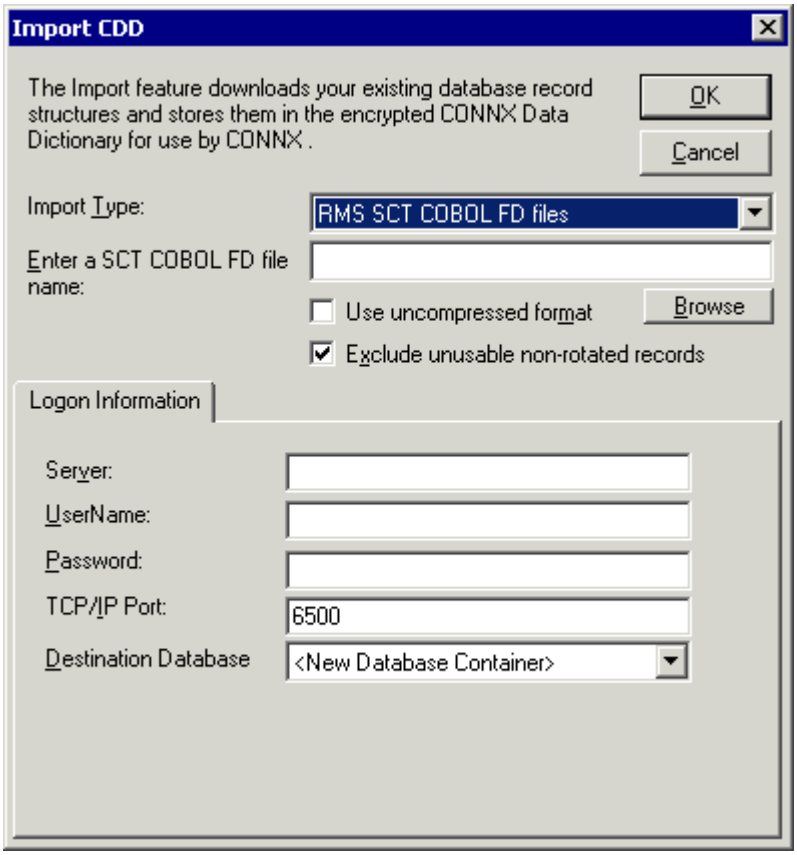
Non-standard, non-dynamic rotated array import capability has been added to CONNX for use by SCT sites that choose not to compress rotated arrays. The Import CDD dialog box has a new checkbox that can be selected to import SCT files under the assumption that the arrays are not compressed.

Related Topics

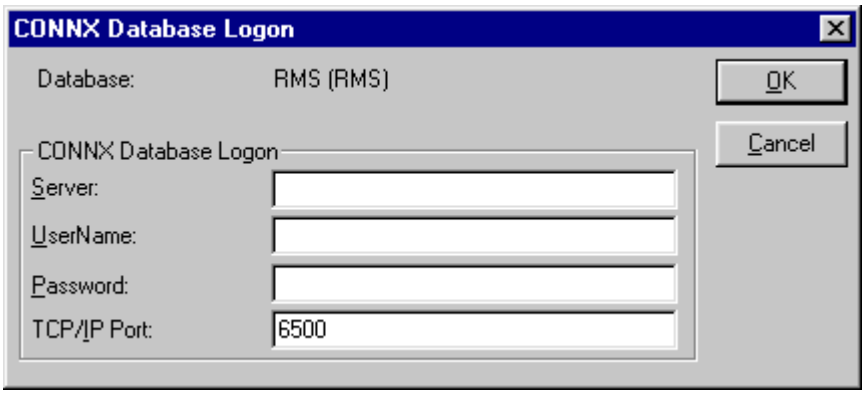
- Rotated Arrays (RMS and VSAM only)
- Configuring a rotated array
- Using the Rotated Array Assistant
- Using non-standard uncompressed format within SCT Plus2000-specific rotated arrays

Using non-standard uncompressed format within SCT Plus2000-specific rotated arrays

1. Select **Import from RMS SCT COBOL FD Files** or **Import from VSAM SCT COBOL FD Files** in the Import CDD dialog box.
2. Type an **RMS** or **VSAM SCT COBOL FD file name**.
3. Select the **Use uncompressed format** check box, and then click the **OK** button.



4. You can also use the Browse button below the text box to locate files to import. If you do not need to use the Browse button, proceed to Step 7.
5. Click the **Browse** button. If you are not connected to a VMS server, the CONNX Database Login dialog box appears.

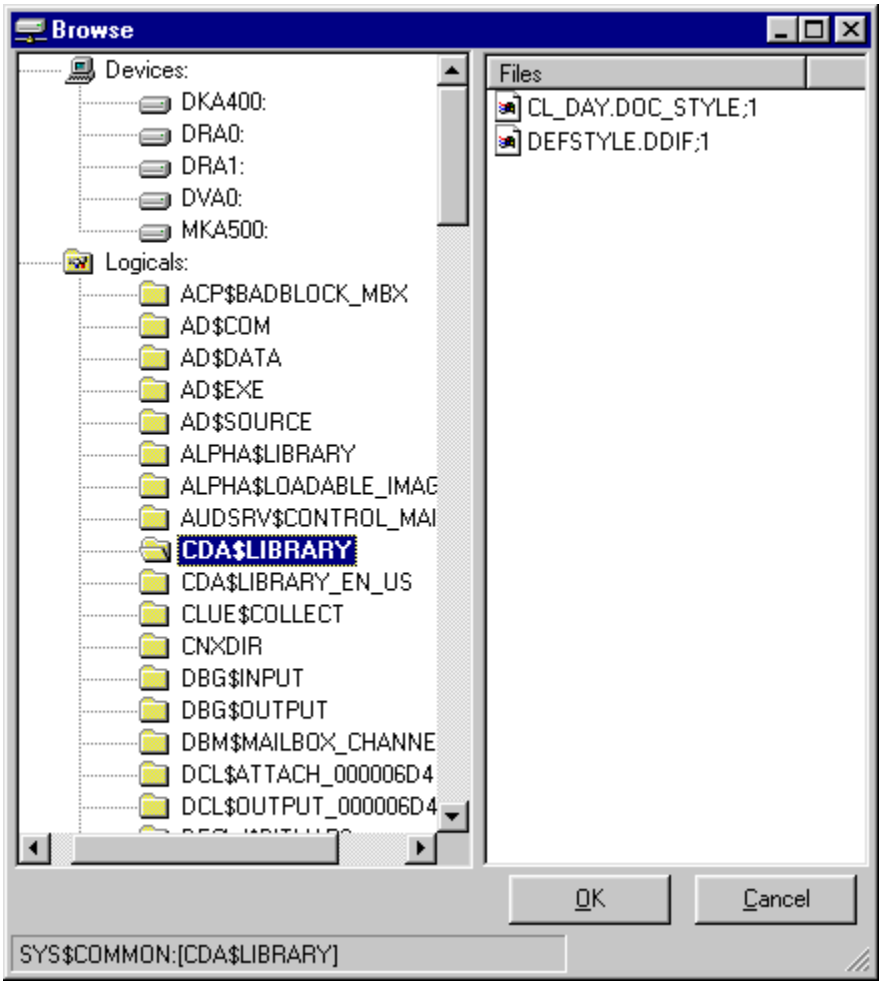


Note: *Use the following wildcard syntax to import multiple SCT COBOL FD files from the SIS module: SI$SOURCE:*RC.LIB*

Note: *If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

Note: *Multiple files can be specified in the Enter an SCT COBOL FD file name text box by separating each file name with a comma. The allowable limit is 255 characters.*

6. Type the server name or IP address, a user name, and password in the corresponding text boxes. Port 6500 is listed in the **TCPIP Port** text box by default.
7. Click the **OK** button. The **Browse** dialog box appears.
8. Select the file(s) in the **Browse** dialog box, and then click the **OK** button to return to the **Import CDD** dialog box.



9. Enter the following information in the **Import CDD** dialog box:

1. Type the server name or IP address, a user name, and password in the corresponding text boxes on the **Logon Information** tab.
2. Port 6500 is listed in the **TCP/IP Port** text box by default. Any change made to the port setting in this text box becomes a permanent change to the port setting of the imported database. See "To edit the OpenVMS Site-Specific Startup Command Procedure" in the CONNX Installation Guide for information on changing the port setting on the server.

3. Select the destination database from the **Destination Database** list box. See Adding a database connection for more information.
4. Click the **OK** button.

10. The RMS data files name is automatically entered in the table properties using the standard SCT data logicals, for example, the standard SCT data logicals, for example, SI$DATA:ADFILE.DAT. See CONNX and SCT Import Rules for more information on the import logic.

11. Save the CDD by clicking **Save** on the **File** menu. The selected RMS or VSAM SCT COBOL FD files appear in the CONNX Data Dictionary Manager window.

Related Topic

Rotated Arrays (RMS and VSAM only)

SCT-specific Non-standard Rotated Arrays

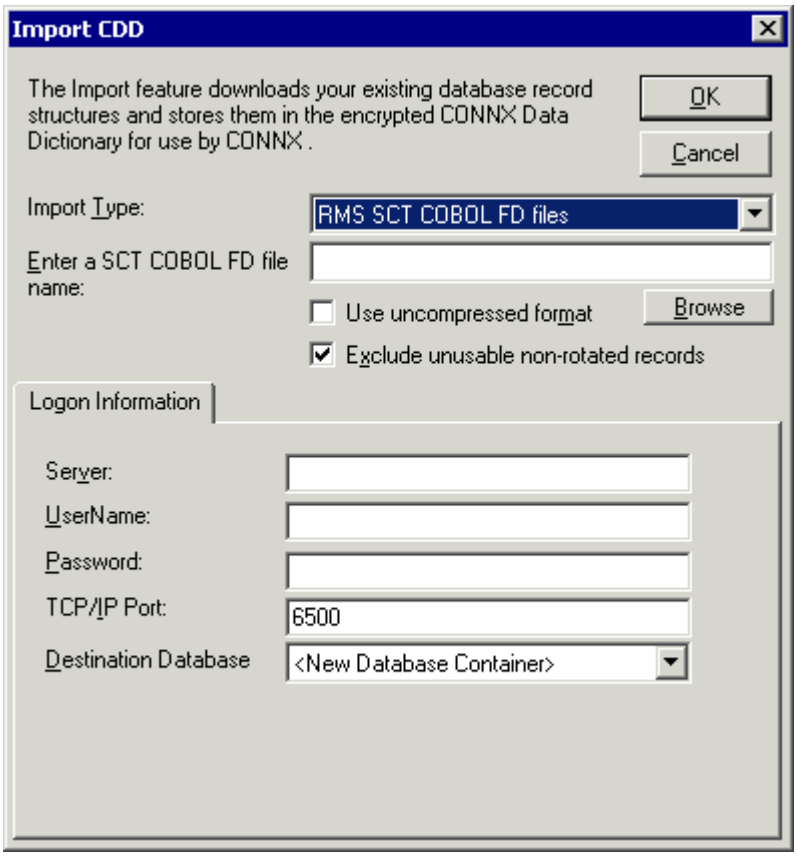
SCT Plus2000-specific Unusable Non-rotated Records

By default, the Exclude Non-rotated Records feature is invoked to prevent the population of cumbersome table records in the CONNX Data Dictionary during the SCT import.

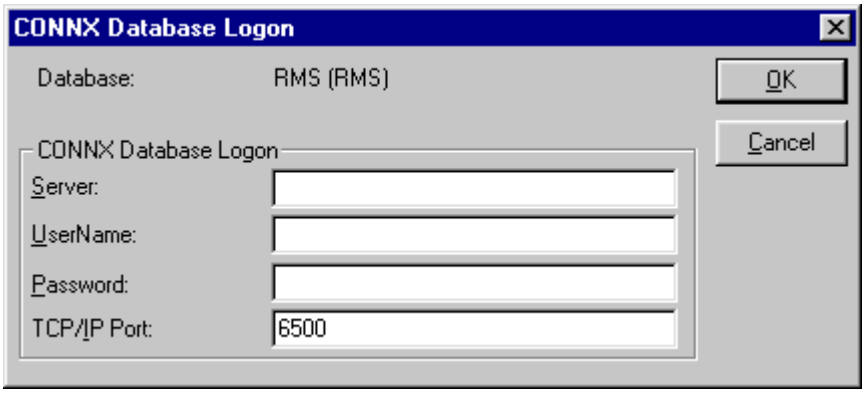
Many SCT IA Plus tables contain multiple types of data or multiple iterations of like data within a single file. CONNX represents these multi-data structures individually. One CONNX table is built for each record type or repeating segment encountered within an SCT file. In such cases, importing the entire table structure, without breaking it down into separate tables, renders the use of this table awkward for most purposes.

Excluding unusable non-rotated records within SCT Plus2000-specific rotated arrays

1. Select **Import from RMS SCT COBOL FD Files** or **Import from VSAM SCT COBOL FD Files** in the Import CDD dialog box.
2. Type an RMS or VSAM SCT COBOL FD file name.
3. Select the **Exclude unusable non-rotated records** check box, and then click the **OK** button.



4. You can also use the Browse button below the text box to locate files to import. If you do not need to use the Browse button, proceed to Step 7.
5. Click the **Browse** button. If you are not connected to a VMS server, the CONNX Database Login dialog box appears.



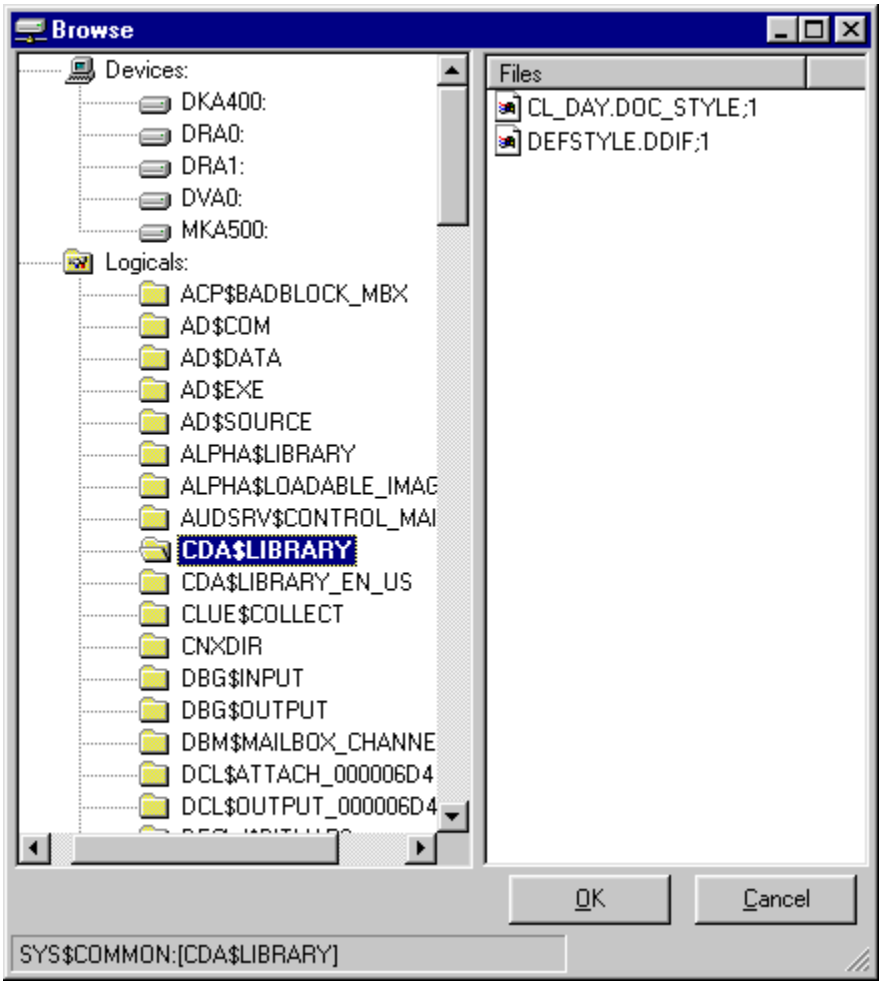
Note: *Use the following wildcard syntax to import multiple SCT COBOL FD files from the SIS*

*module: SI$SOURCE:*RC.LIB*

***Note:** If CONNX detects more than one record layout when importing, multiple data dictionary entries are created.*

***Note:** Multiple files can be specified in the Enter an SCT COBOL FD file name text box by separating each file name with a comma. The allowable limit is 255 characters.*

6. Type the server name or IP address, a user name, and password in the corresponding text boxes. Port 6500 is listed in the **TCPIP Port** text box by default.
7. Click the **OK** button. The **Browse** dialog box appears.
8. Select the file(s) in the **Browse** dialog box, and then click the **OK** button to return to the **Import CDD** dialog box.



9. Enter the following information in the **Import CDD** dialog box:

1. Type the server name or IP address, a user name, and password in the corresponding text boxes on the **Logon Information** tab.
2. Port 6500 is listed in the **TCP/IP Port** text box by default. Any change made to the port setting in this text box becomes a permanent change to the port setting of the imported database. See "To edit the OpenVMS Site-Specific Startup Command Procedure" in the CONNX Installation Guide for information on changing the port setting on the server.
3. Select the destination database from the **Destination Database** list box. See Adding a database connection for more information.

4. Click the **OK** button.

10. The RMS data files name is automatically entered in the table properties using the standard SCT data logicals, for example, the standard SCT data logicals, for example, SI$DATA:ADFILE.DAT. See CONNX and SCT Import Rules for more information on the import logic.

11. Save the CDD by clicking **Save** on the **File** menu. The selected RMS or VSAM SCT COBOL FD files appear in the CONNX Data Dictionary Manager window.

Related Topic

Rotated Arrays (RMS and VSAM only)

SCT-specific Non-standard Rotated Arrays

Clone Table Assistant (RMS, C-ISAM, VSAM, and Adabas SQL Gateway only)

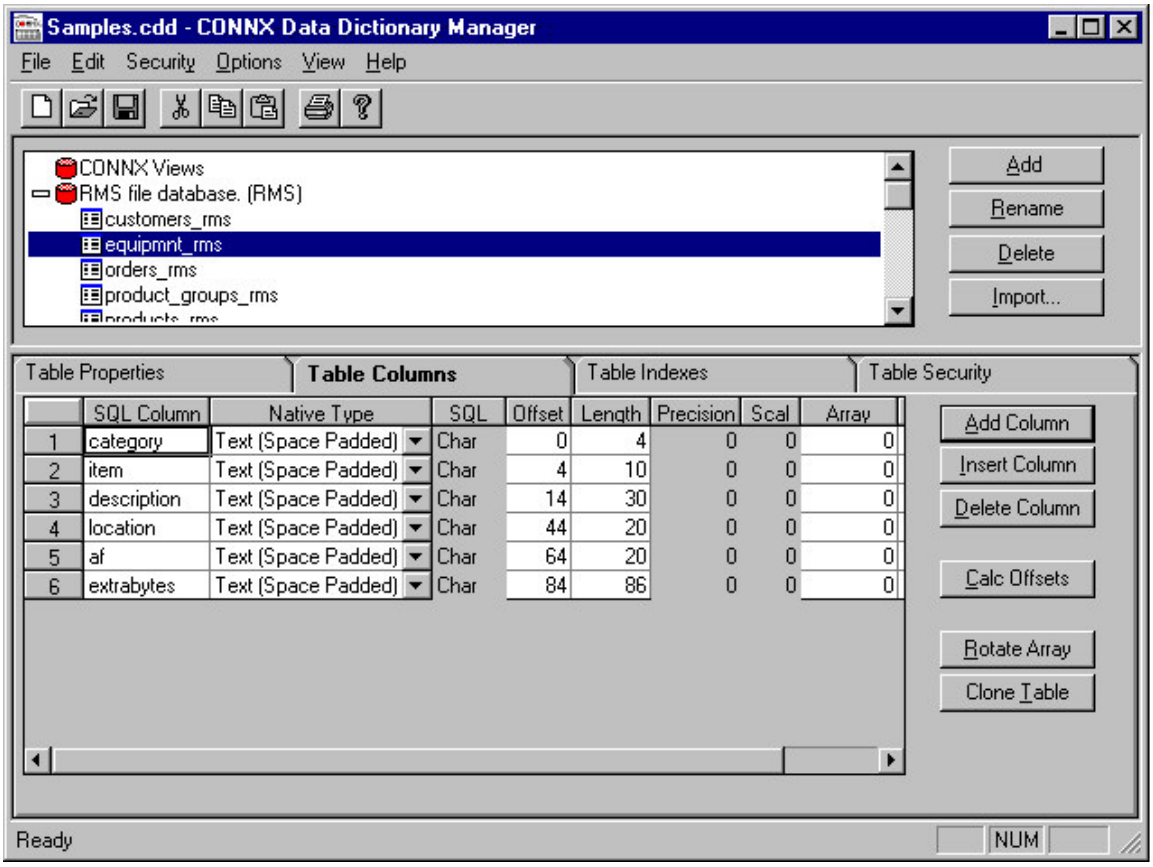
The Clone Table Assistant is a CONNX tool that simplifies the process of creating a copy of a table definition containing fewer fields than the original. This is useful for tables that contain more than 255 columns, because many popular tools, including Microsoft Access and Visual Basic, cannot use tables that contain more than 255 fields. With the Clone Table Assistant, virtual tables that contain limited information and increase data security and usability can be created.

Related Topic

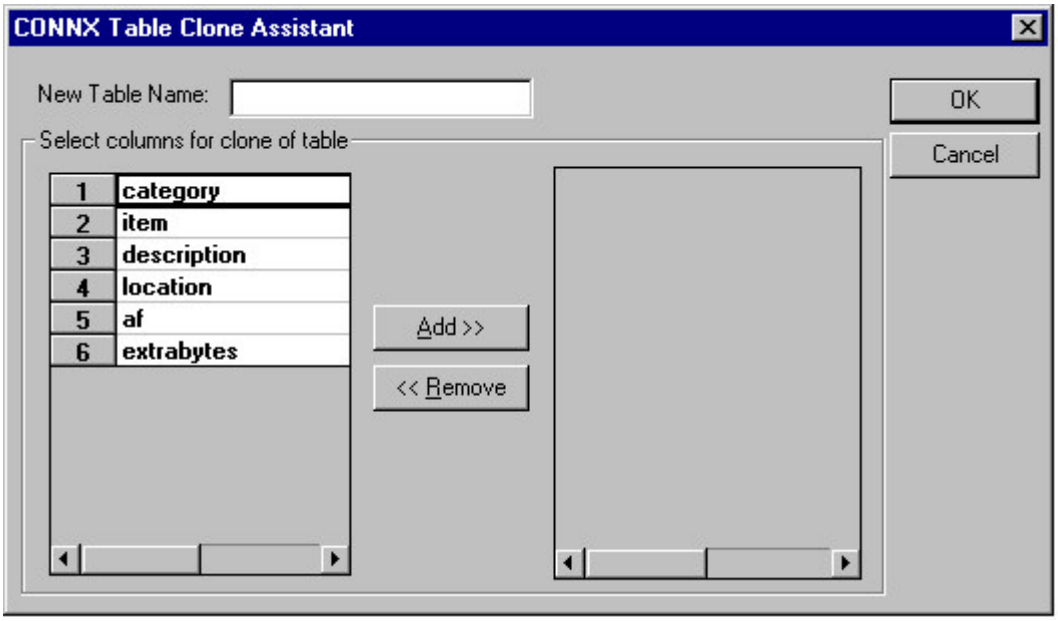
To use the Clone Table Assistant

To use the Clone Table Assistant

1. Select a table to clone in the CONNX Data Dictionary Manager window, and then click the **Clone Table** button.



2. The CONNX Table Clone Assistant window appears.



3. Type a name for the new table in the **New Table Name** text box. Table-naming conventions are as follows:

1. Maximum size is 50 characters.
2. There must be no spaces in the name.
3. The table name cannot begin with a number.
4. The table name must be unique.

4. Highlight the fields in the original table to include in the new table, and then click the **Add** button. If not including one of the fields selected for the new table, select the field and then click the **Remove** button.

Note: If you are cloning an IMS table, include all the virtual columns that relate to the parent key. These columns have a CNX_ prefix and are used to preserve the table hierarchy. If you do not include these columns, CONNX can not properly navigate the table hierarchy and may cause unpredictable results.

5. Click the **OK** button. The newly cloned table contains a subset of available fields, selected from the original table.

Hot Connections

To open a hot connection to a server

With CONNX, you can create a hot connection to an OpenVMS server that enables a connection to remain open. Leaving a connection open helps shorten the length of time it takes to connect to a server, which can be beneficial to users who are opening and closing connections repeatedly or who are operating in an environment where a high volume of short connections are made, as with Web servers. The number of open connections should be kept to a minimum in order to maintain the highest levels of VMS system performance.

Important: *An open server connection decreases the security of data stored on that server.*

1. Connect to the VMS server with which you are working.
2. Open a terminal emulator, such as Telnet or Pathworks.
3. Type the following DCL command, and then press **<Enter>:**

   ```
   $ SET DEF CNXDIR
   ```

4. Type the following DCL command, and then press **<Enter>:**

   ```
   $ RUN CNXHOT
   ```

5. Type a VMS user name, using uppercase characters, and then press **<Enter>.**
6. Type a VMS password, using uppercase characters, and then press **<Enter>.**
7. Define a system-level logical that determines the number of prestarted servers for each type of database.
    8. For **RMS** databases:
       DEFINE /SYSTEM CNXRMSSERVERCOUNT
       <number of prestarted servers>
    9. For **Rdb** databases:
       DEFINE /SYSTEM CNXRDBSERVERCOUNT
       <number of prestarted servers>
    10. For **DBMS** databases:
        DEFINE /SYSTEM CNXDBMSSERVERCOUNT <number of prestarted servers>
8. Restart the CONNX Listener process.

CONNX and Oracle Advanced Features

Using Oracle stored procedures with CONNX

Oracle stored procedures can be executed within CONNX by using the following ODBC Call syntax. If using Oracle databases within Microsoft Access, use the ODBC passthrough command in the Query window. If using Oracle databases with Microsoft Visual Basic, Oracle stored procedures can be executed directly in the Visual Basic application window.

```
{ CALL storedprocedurename( param1, param2, ...)}
```

Related Topics

»Performing Queries in Passthrough Mode in Oracle Rdb Databases

»Importing Oracle Database Tables

»Importing Oracle Rdb Database Tables

SQL Functionality and Oracle Native Functions

The following is a list of CONNX native SQL functions that expand Oracle SQL functions.

AutoCounter
BitLength
Convert
COT
Degrees
Difference
Extract
Hex
If
IfNull
Insert
Left
Location
OctetLength
Position
Rand
Radians
Repeat
Reverse
Right
Space
Strcmp
Stuff
TimeStampAdd
TimeStampDiff
Trim

SQL functions not natively supported by Oracle cannot be used with SQL passthrough.

Related Topic

Using Oracle stored procedures with CONNX

Performing Queries in Passthrough Mode in Oracle Rdb Databases

Queries can be performed in passthrough mode through the Oracle Rdb driver. Joins made between two Rdb tables stored in the same Rdb database are performed on the OpenVMS server automatically. This enhancement increases the speed of join queries of this type, especially those that use a Where clause in the SQL statement.

If a function is used within the query that does not exist in the Rdb database, CONNX processes the query internally.

Related Topics

Importing Oracle Rdb Database Tables

How CONNX Transparently Maps Dynamic SQL to Static SQL for DB2

For all ODBC applications which connect to CONNX DB2 module data sources, a dynamic SQL statement executes as static SQL provided that the CDD opened by the data source defines one or more static SQL packages, each with one or more static SQL statements, and that there is an equivalent static SQL statement for the dynamic SQL statement in the data source CDD. This process is completely transparent to the ODBC application, end-user, or developer. Consider, for instance,

```
Select empno from employee where empno > ?
```

which looks like a dynamic SQL statement, but is executed as static SQL if the CONNX CDD contains the appropriate static package and statement. If there is no match, the SQL statement executes as dynamic SQL.

With CONNX, dynamic SQL can be transformed and executed as static SQL, which provides for enhanced security and performance. A potential scenario for this feature is the development of an ODBC-based application using dynamic SQL statements. Once the dynamic SQL statements are finalized, they can be built into one or more static SQL packages and the results stored in a CDD. The completed ODBC application can then point to the CDD, and all dynamic SQL statements will execute as static SQL, provided that there is an appropriate match in the CONNX CDD.

The net result is that, although there is no support for static SQL in the ODBC specification, CONNX automatically transforms dynamic SQL to static SQL, without requiring changes to the ODBC application code.

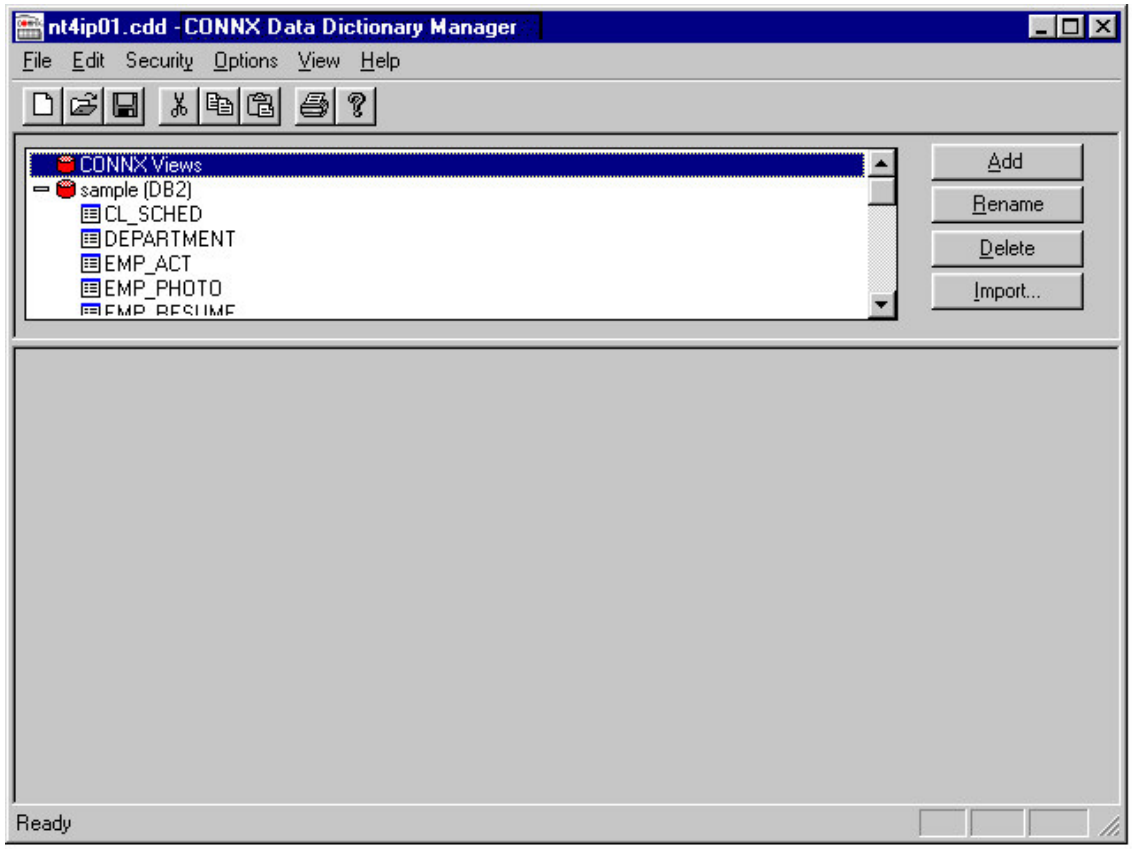
Related Topics

- To build a static SQL package using the CONNX CDD import utility
- Host Configuration for DB2 Administrators
- CONNX DB2 Dynamic SQL Packages
- To establish CONNX and DB2 CDD configuration options
- Importing Stored Procedures

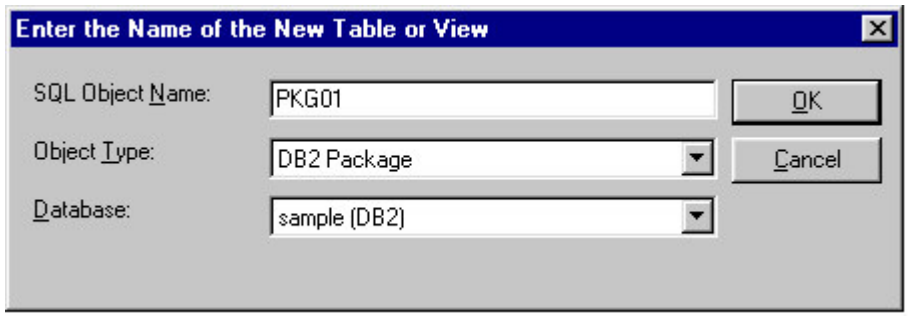
To build a static SQL package using the CONNX CDD import utility

The following steps show how to use CONNX to build a static SQL package with one SELECT, INSERT, UPDATE, AND DELETE against tables defined in the DB2 UDB sample database.

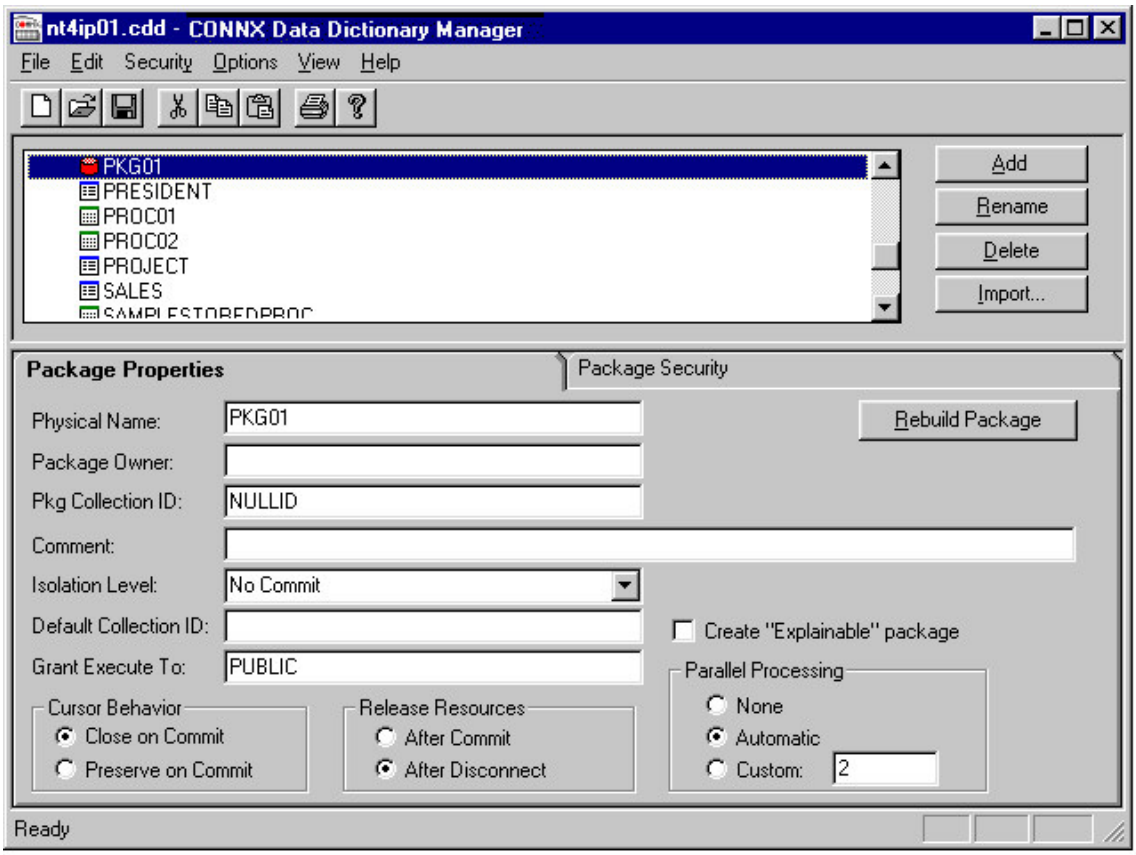
1. Click the **Add** button in the CONNX Data Dictionary Manager window.



2. The **Enter the Name of the New Table or View** dialog box appears. Type a 1- to 18-character package name in the **SQL Object Name** text box.

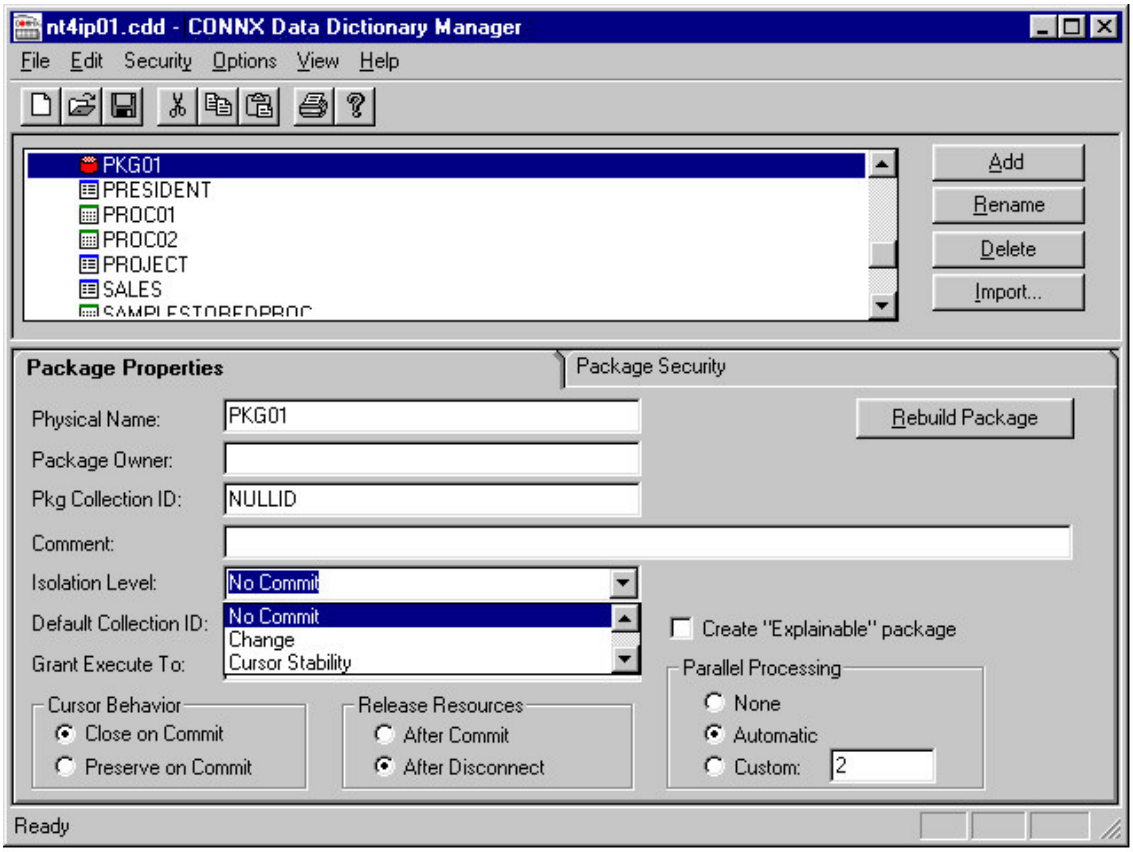


3. Select **DB2 Package** in the **Object Type** list box.
4. Select the target database name in the **Database** list box.
5. Click the **OK** button.
6. The **Package Properties** tab is selected in the lower pane of the CONNX Data Dictionary Manager window.



The **Physical Name** text box contains the 1- to 18-character name of the package.

7. Type the name of the package owner in the **Package Owner** text box, for example, QUSER, SYSIBM, or PUBLIC. This is an optional field.
8. The **Pkg Collection ID** text box contains the 1- to 18-character collection, library, owner, or schema name in which the package is created. The default is NULLID.
9. The **Comment** text box is an optional field.



10. Select an isolation level in the **Isolation Level** list box.
11. The **Default Collection ID** text box is an optional 1- to 18-character entry that resolves flat table names to two-part table names.
12. The **Grant Execute To** text box can contain a user or group ID. The default is PUBLIC.
13. Under **Cursor Behavior** select **Close on Commit** to build cursors which close after a commit or select **Preserve on Commit** to build cursors with hold, which preserve their current row pointers across commits.

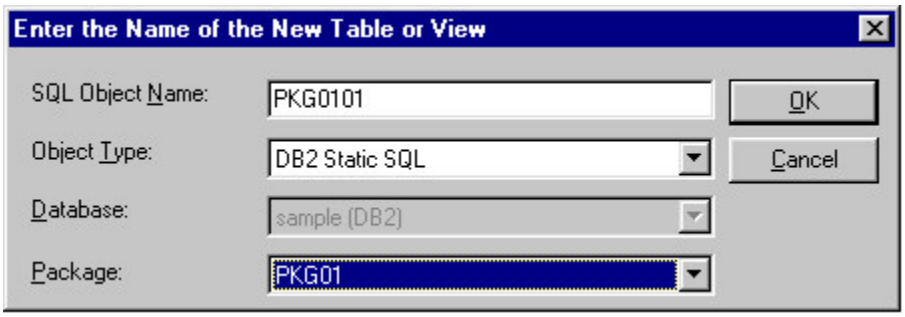
    Note: *Cursor behavior Option Preserve on Commit is not implemented for DB2 for OS/390 and DB2/MVS.*
14. Under **Release Resources** select **After Commit** or **After Disconnect** to define when resources are released.

    This static SQL package bind time parameter instructs the target DB2 server to release execution resources and serialization or sharing locks at commit or at disconnect time. After Commit instructs the DB2 target system to release resources and locks after a successful commit or rollback, whereas After Disconnect instructs the DB2 system to hold the resources and locks until the connection is disconnected.
15. Select the **Create Explainable Package** check box to create an SQL package which can be explained by the DB2 Explain utility.
16. Under **Parallel Processing,** an optional field, select **None, Automatic,** or **Custom** to specify the degree of I/O parallelism.

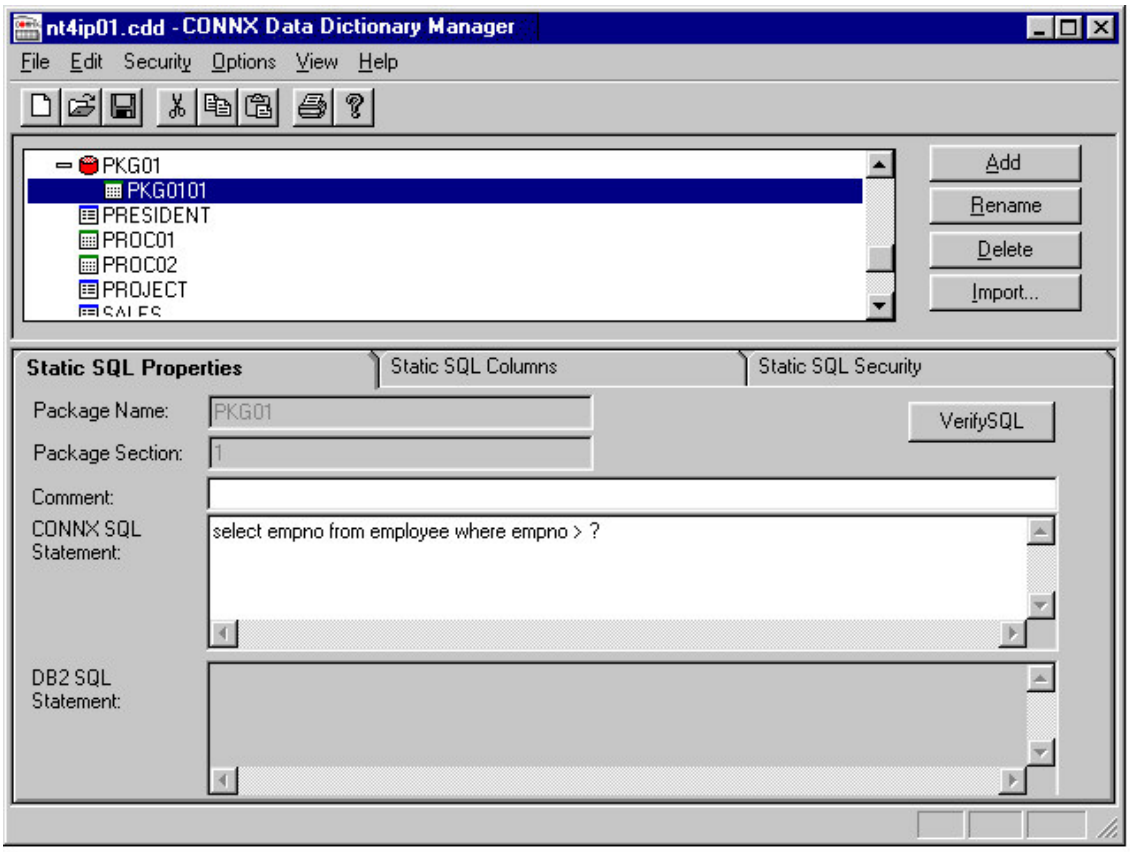
This static SQL package bind time parameter instructs the target DB2 server to use n (1 <= n <= 32767) parallel subqueries for all SELECT statements bound into the package. The default setting (Automatic) represents the special degree of ANY, which instructs the target DB2 system to determine the number of parallel subqueries at run time. Specifying a custom value of between 3 and 32767 is meaningful only when the DB2 target database is running on a symmetric multi-processing (SMP) architecture, such as an OS/390 Parallel Sysplex, an RS/6000, or a Windows Cluster Server. Specifying a custom value of 2 can improve performance for I/O-bound queries, even on single processor machines.

To define static SQL statements for the static DB2 package

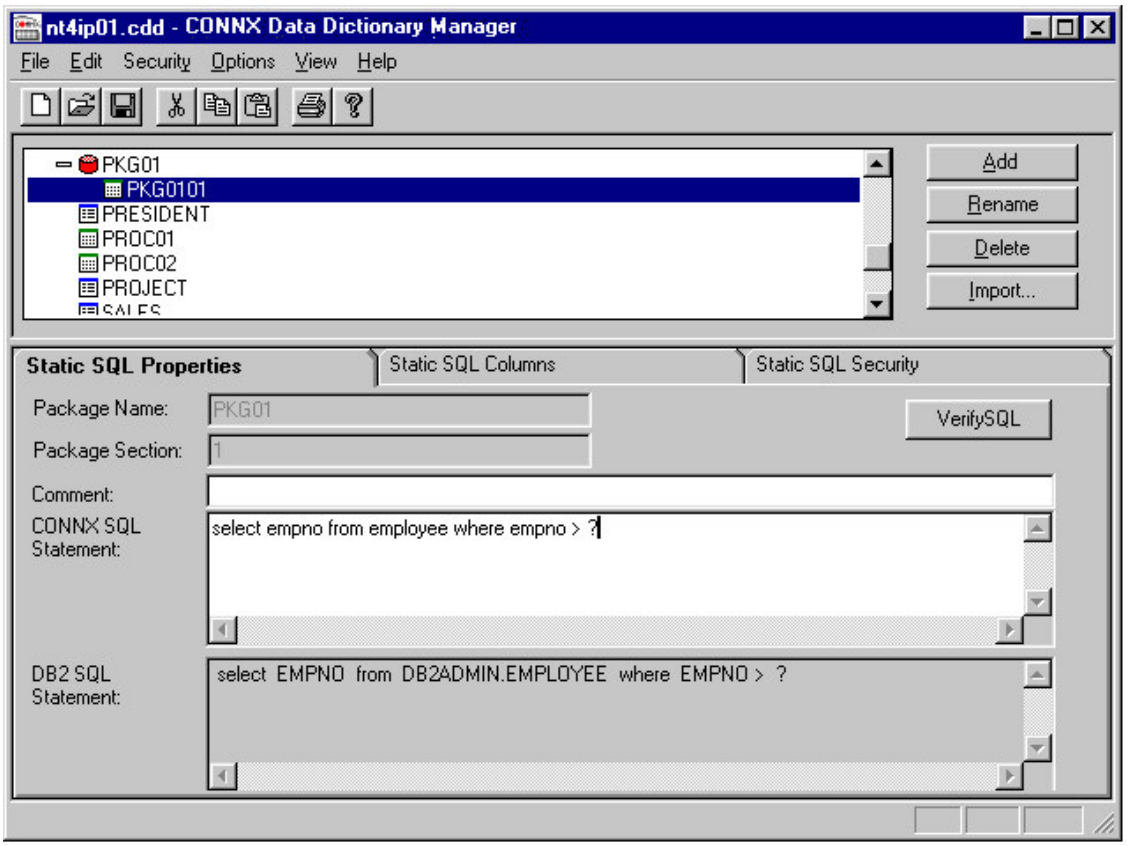
1. Click the **Add** button in the CONNX Data Dictionary Manager window.
2. The **Enter the Name of the New Table or View** dialog box appears. Type a user-defined character string in the **SQL Object Name** text box.



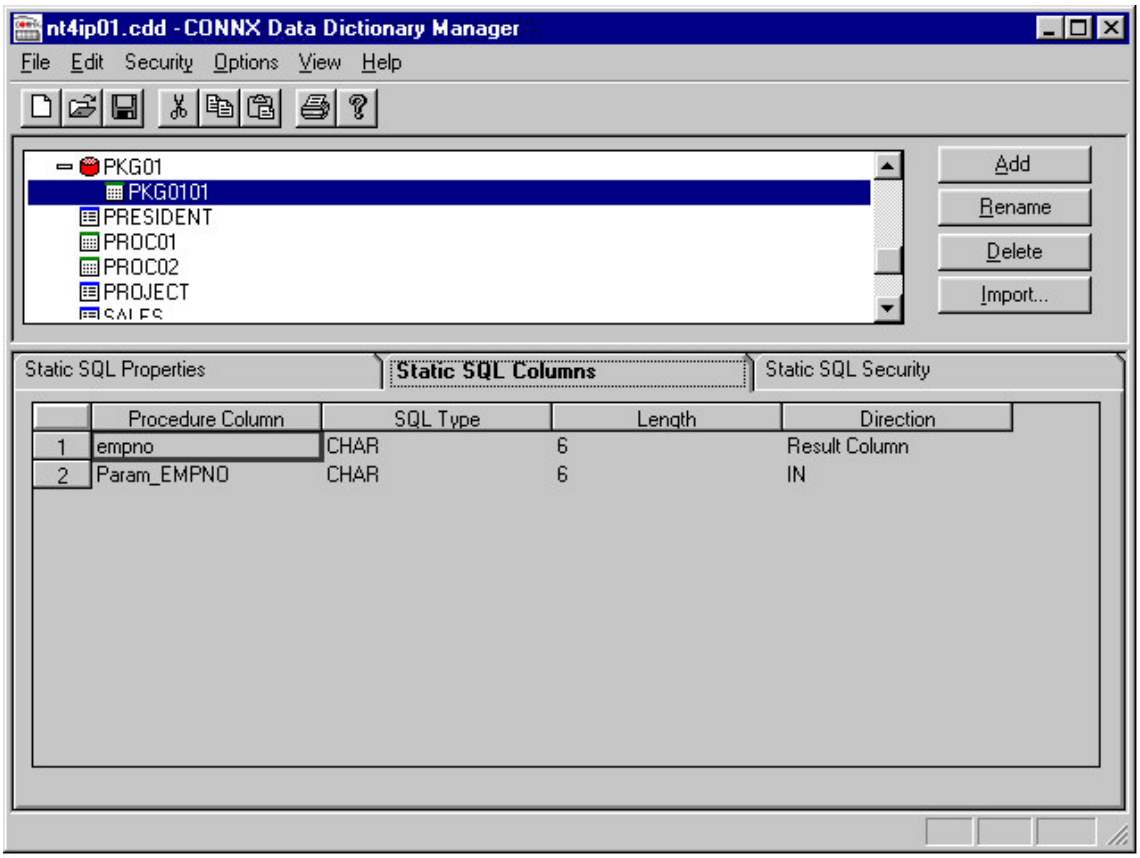
3. Select **DB2 Static SQL** in the **Object Type** list box.
4. Select the package name in the **Database** list box.
5. Click the **OK** button.
6. The **Static SQL Properties** tab is selected in the lower pane of the CONNX Data Dictionary Manager window. Type a dynamic SQL statement in the **CONNX SQL Statement** text box.



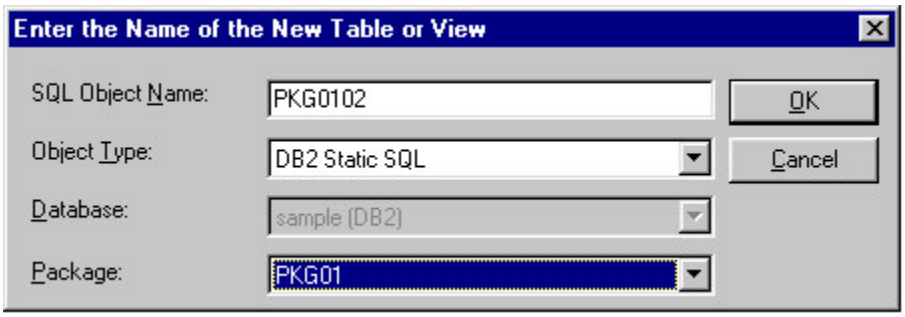
7. Click the **Verify SQL** button to view the results of the DB2 SQL statement in the lower **DB2 SQL Statement** pane.



8. Select the **Static SQL Columns** tab, which displays the attributes of the result column(s) and the input parameter(s) of the static SQL statement.

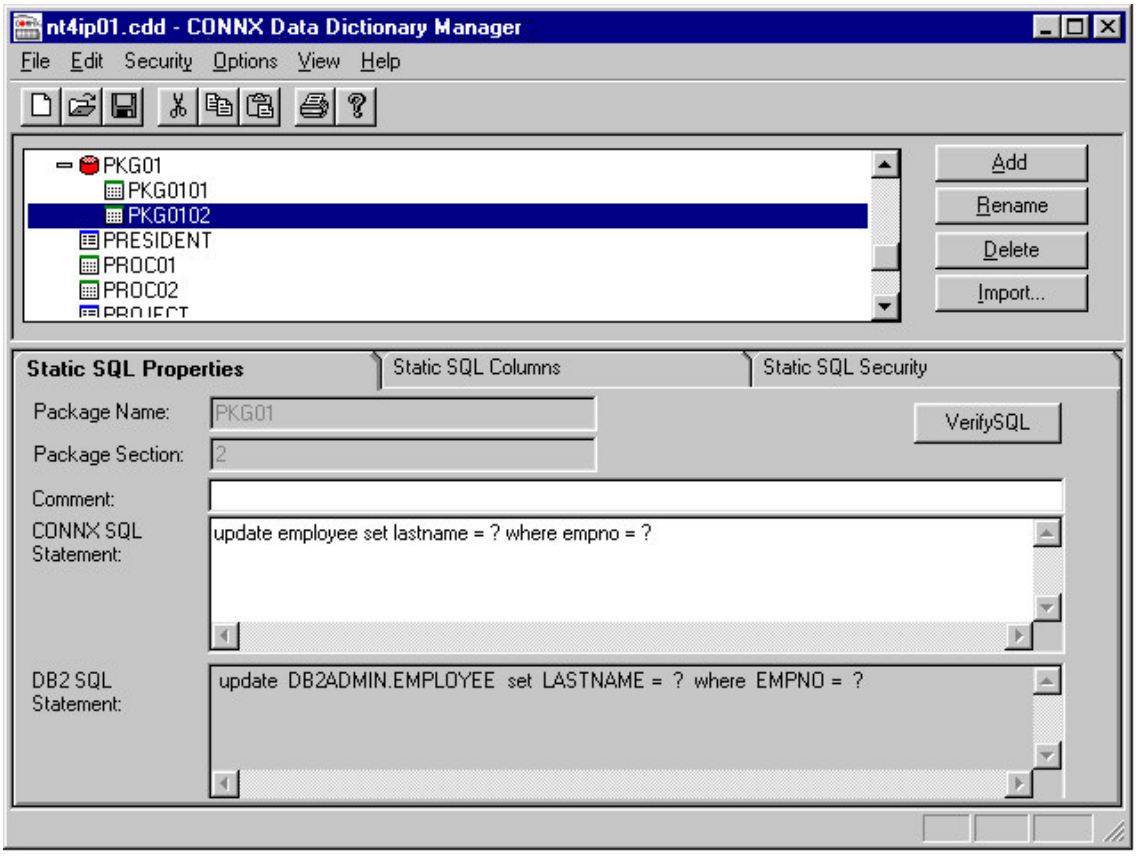


9. The first static SQL statement in package PKG01 is now defined. To bind another static SQL statement, return to the main CDD window and click the **Add** button. In keeping with the naming convention of step 2, define a user-defined SQL Object Name = PKG0102, an object type = DB2 Static SQL, and select package PKG01 from the list box.



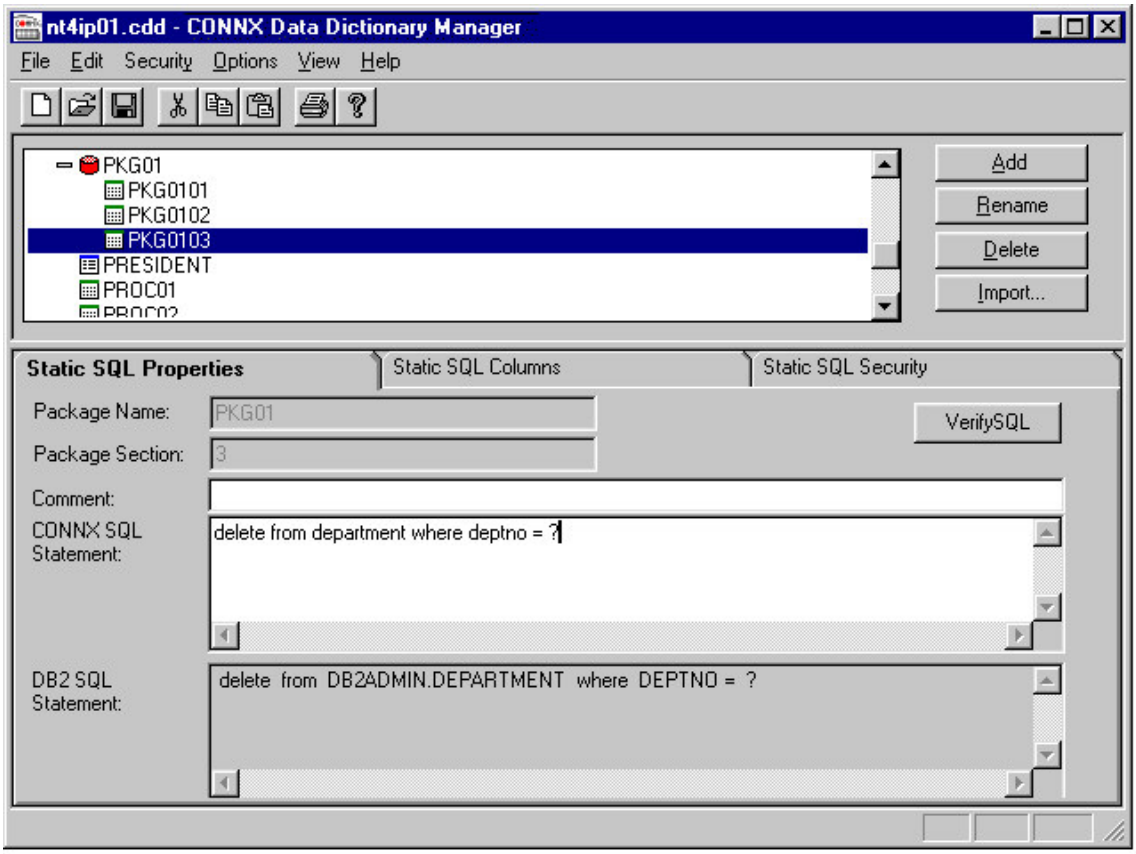
10. Enter the dynamic SQL **UPDATE** statement as it appears in the **CONNX SQL Statement** list box.

11. Click the **Verify SQL** button. The DB2 SQL statement appears in the lower window.

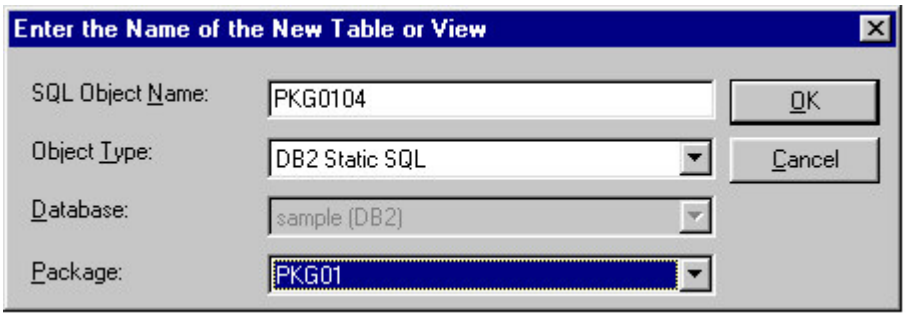


12. Return to the main CDD window and click the **Add** button. In keeping with the naming convention of step 2, define a user-defined SQL Object Name = PKG0103, an object type = DB2 Static SQL, and select package PKG01 from the list box.
13. Enter the dynamic SQL **DELETE** statement as it appears in the **CONNX SQL Statement** list box.

14. Click the **Verify SQL** button. The DB2 SQL statement appears in the lower window.

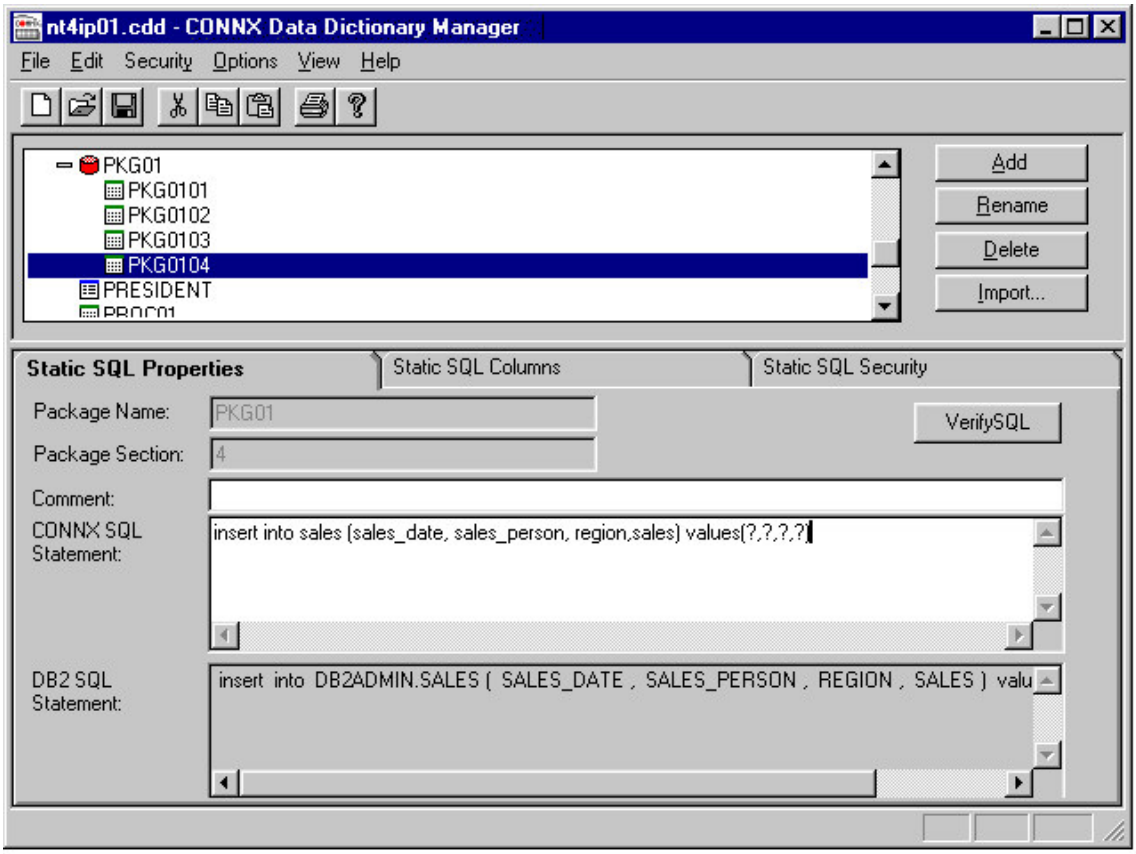


15. Select the **Static SQL Columns** tab, which displays the attributes of the result column(s) and the input parameter(s) of the static SQL statement (not shown).
16. Return to the main CDD window and click the **Add** button. In keeping with the naming convention of step 2, define a user-defined SQL Object Name = PKG0104, an object type = DB2 Static SQL, and select package PKG01 from the list box.



17. Enter the dynamic SQL **INSERT** statement as it appears in the **CONNX SQL Statement** list box.

18. Click the **Verify SQL** button. The DB2 SQL statement appears in the lower window.

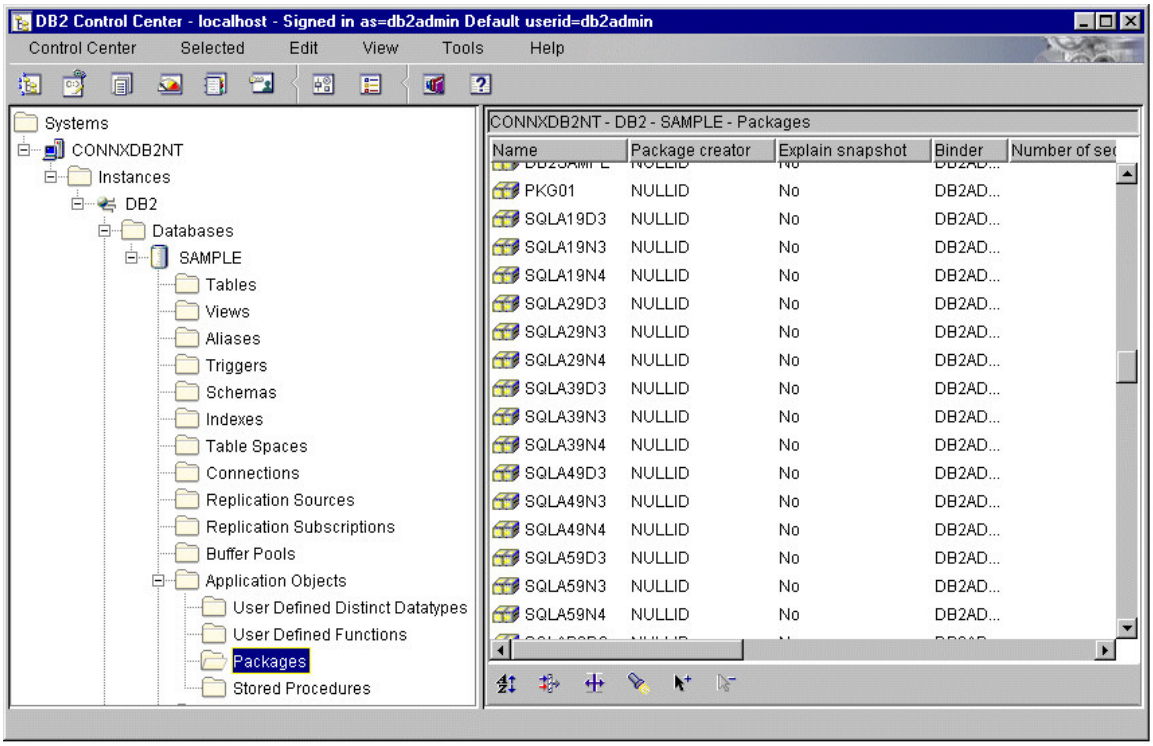


19. Select the **Static SQL Columns** tab, which displays the attributes of the result column(s) and the input parameter(s) of the static SQL statement.
20. Return to the main window and click the **PKG01** icon. Then click the **Rebuild Package** button.
21. If you are not currently logged on to the target host, the **CONNX DB2 Database Logon** dialog box appears.
22. CONNX logs on to the target host and issues Static SQL Binds into package PKG01 for the four statements described in the preceding steps.

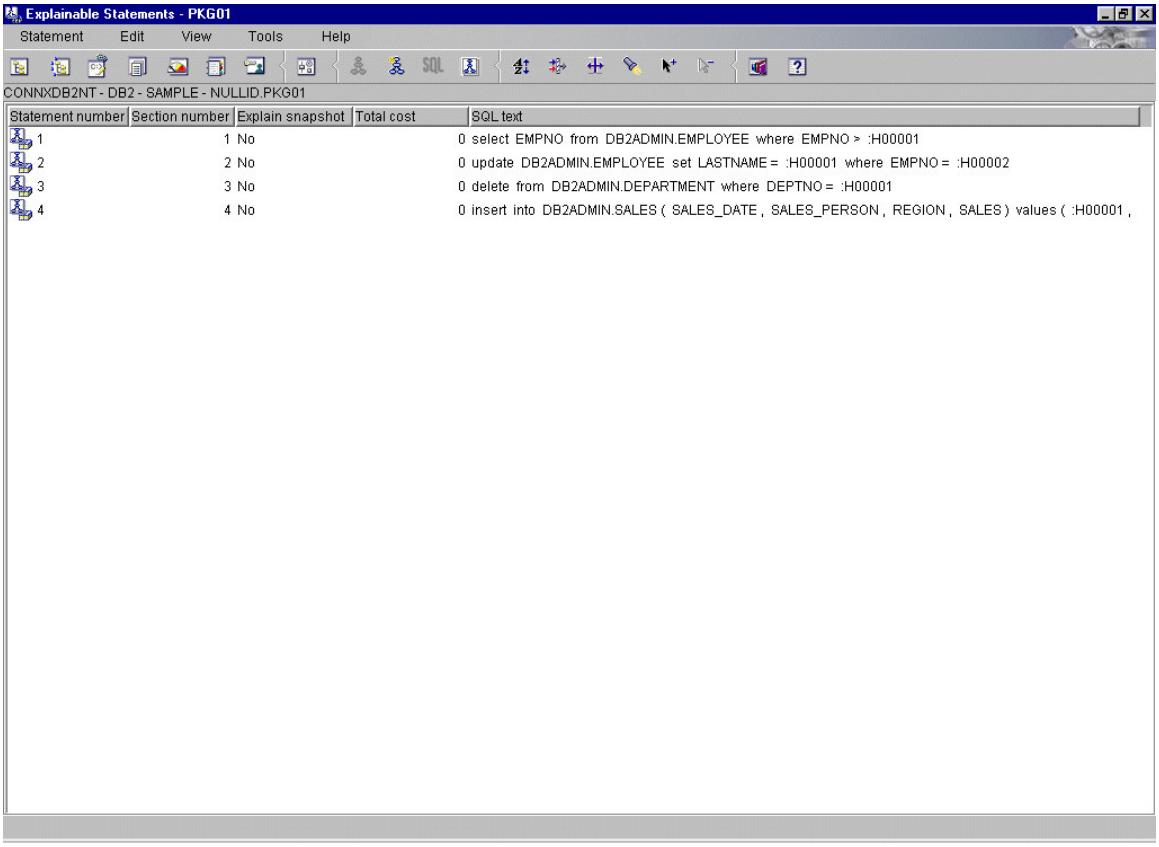
Verifying a successful package build at the target host

For DB2 UDB (Windows and Linux) hosts:

1. Start the **Control Center.**
2. Select the **Target system/instance/database.**
3. Select **Application Object**s in the left pane.
4. Double-click the **Packages** icon.
5. Locate the package name in the right pane.



6. Click the **Package name (PKG01).**
7. Right-click on the **Show Explainable Statement**s menu item.
8. In the **Explainable Statements** window, click on a statement number and right-click to select **Show SQL Text.**



Note that the original dynamic SQL example statement above has been bound as

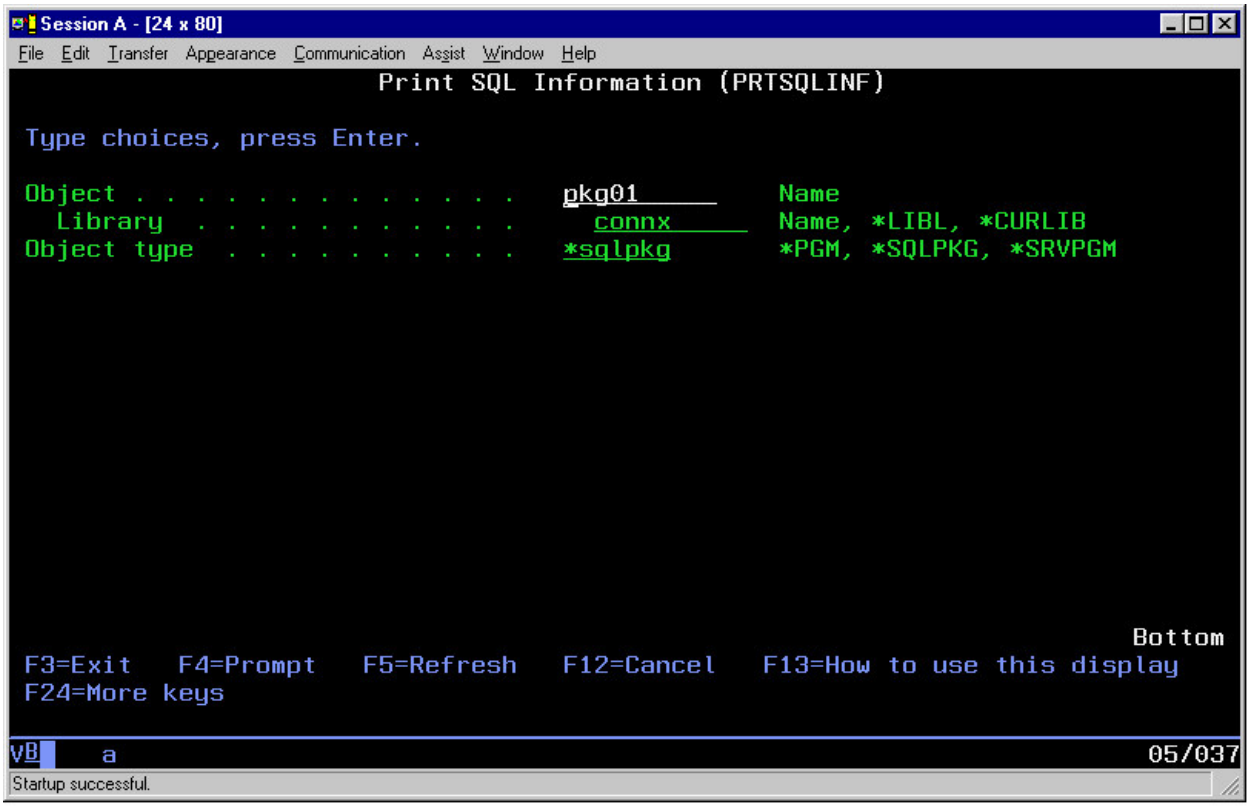
```
Select EMPNO from db2admin.employee where empno > :H00001
```

For OS/400 hosts:

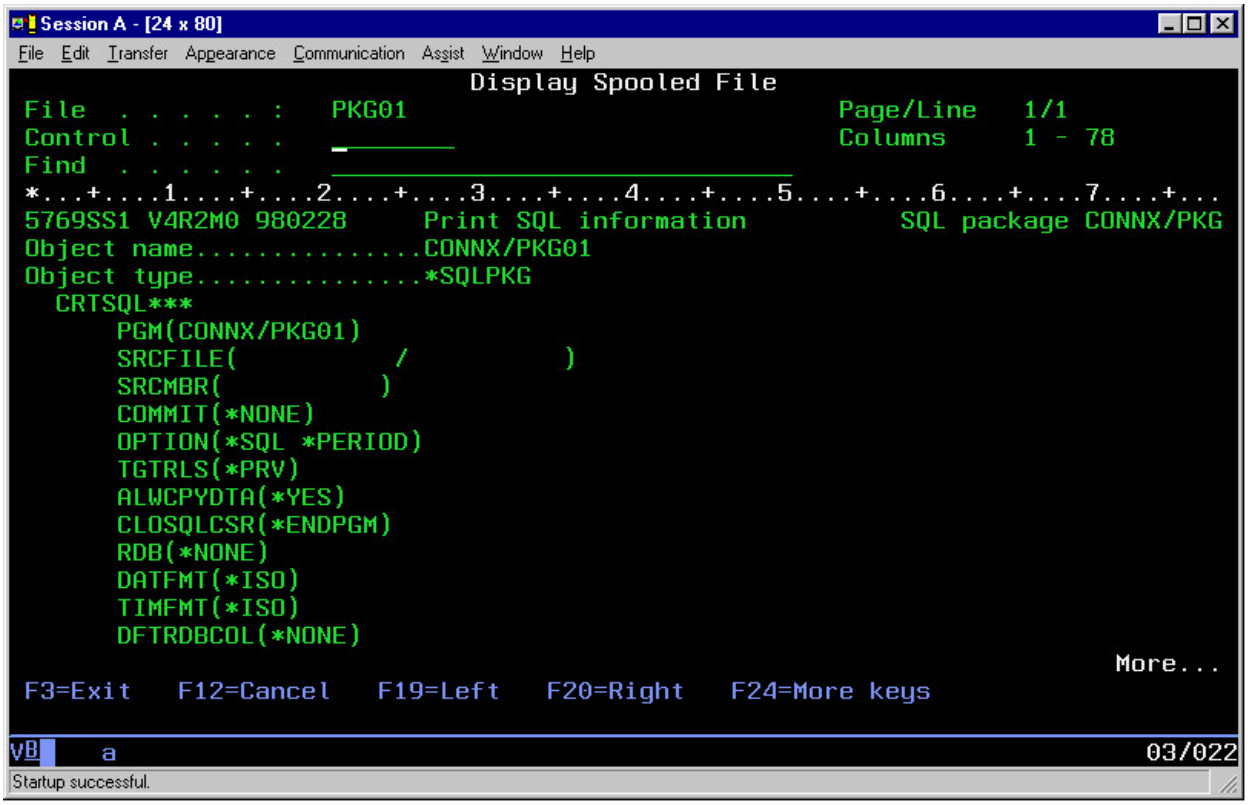
1. Log on to the target OS/400 machine using a 5250 terminal emulator product. At the command line, type the following:

   PRTSQLINF

2. Press **<F4>**.
3. Type the package name next to **Object = PKG01.**
4. Type the target **Library** name.
5. Type the Object Type = *SQLPKG.
6. Press **<Enter>.**



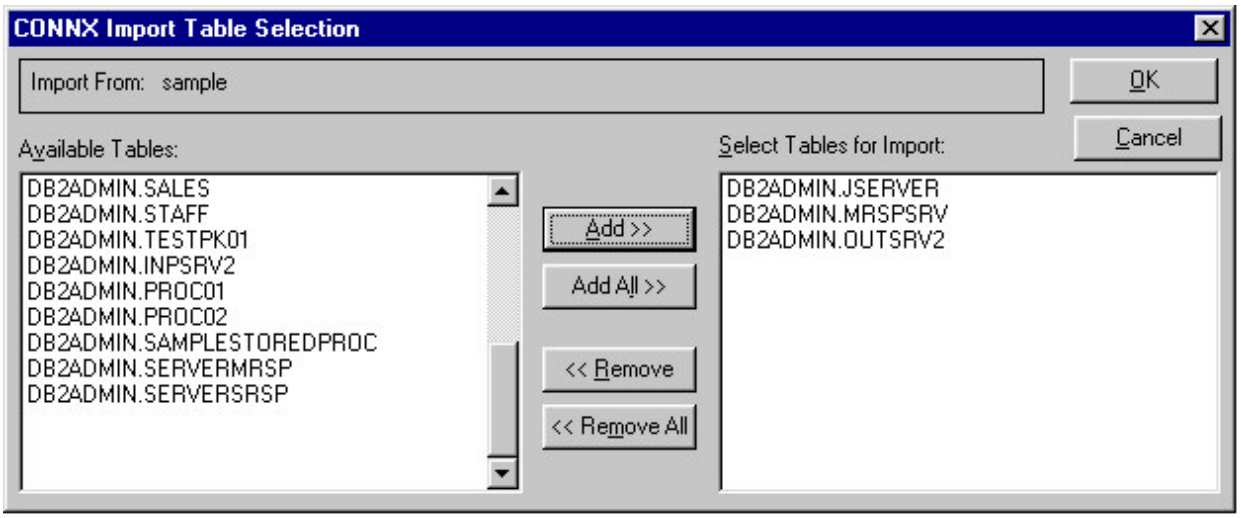
The output is spooled. Use the **WRKSPLF** command to locate the output created by the PRTSQLINF command, and then display the output via the **WRKSPLF** command as shown in the following example:



Importing Stored Procedures

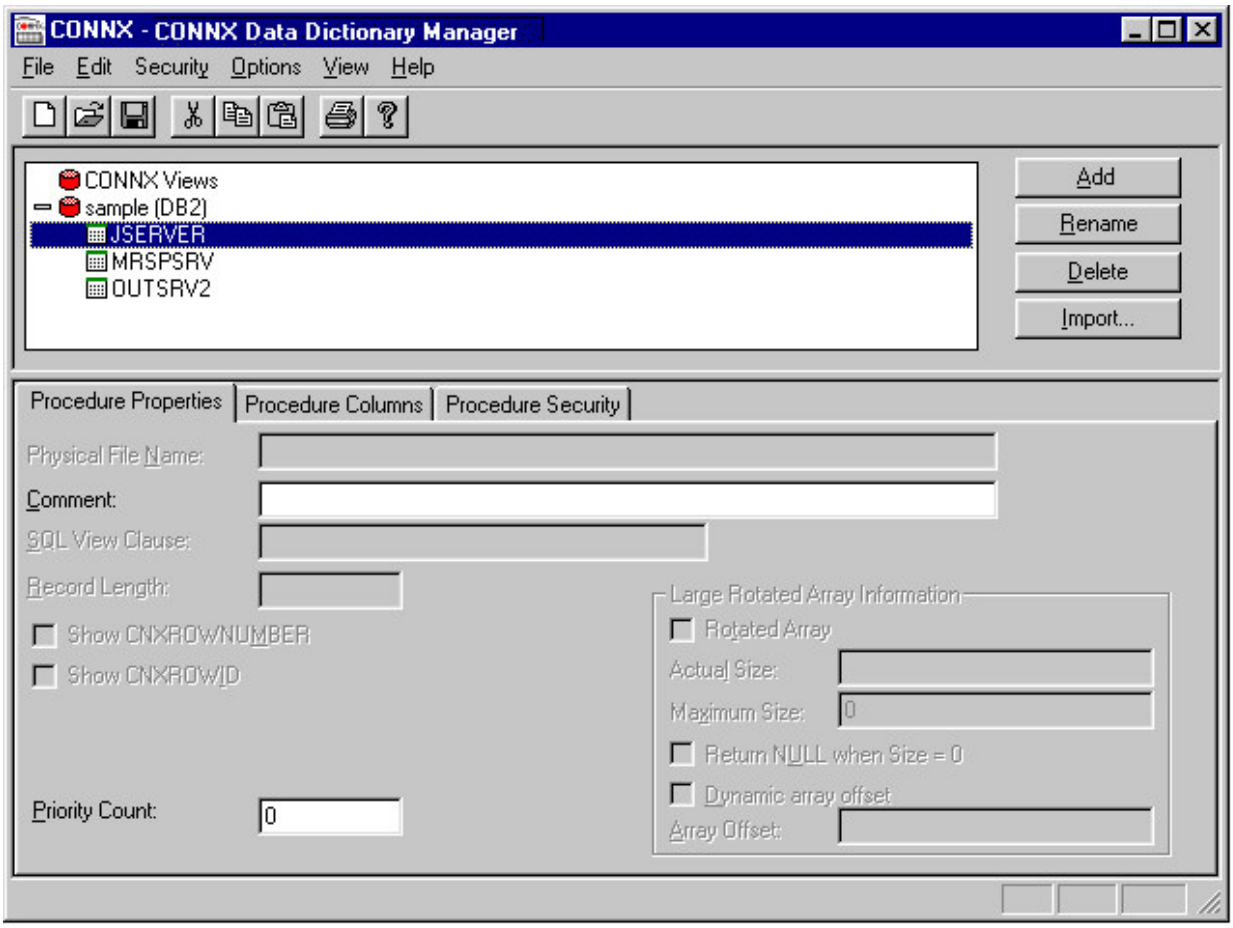
CONNX automatically imports stored procedures during the CONNX table/view import process. In the example below, three stored procedures defined on a DB2 UDB 6.1 for Windows target machine appear as tables in the CONNX Import Table Selection dialog box.

DB2ADMIN.JSERVER and DB2ADMIN.MRSPSRV are Java stored procedures, whereas DB2ADMIN.OUTSRV2 is a C++ stored procedure defined in a DLL (dynamic linked library). These example procedures are included in the DB2 UDB 5.x and 6.x SDK (Software Development Kit).



CONNX can import stored procedures from any DRDA-compliant target, including DB2 UDB 5.x and 6.x for Windows/Linux, DB2/400, and DB2 for MVS/OS/390. On supported platforms, CONNX can import stored procedures with input, output, and input/output parameters, as well as stored procedures which return multiple result sets. Stored procedures can be implemented in a variety of high-level languages, including Java, C/C++, Cobol, RPG, and DB2/400 Command Language. The CONNX user does not need to know the implementation details. As long as a stored procedure is defined to the target DB2 system metadata catalog via the Create Procedure SQL statement, CONNX can find and import it.

Once imported to the CDD, the DB2 stored procedure properties are displayed in a tabbed dialog format similar to imported tables or views.



An ODBC application can invoke a DB2 stored procedure via CONNX with the following syntax:

```
?={call rdbname.schema.procname(?,?,...?)}

or

?={call schema.procname(?,?,...?)}
or
?={call procname(?,?,...?)}
```

where the left question mark (?) is an optional return parameter, and the parameters within parentheses are input/output parameters defined by the ODBC 2.x/3.x SQLBindParameter API calls. DB2 stored procedure result set attributes are not imported; these attributes are determined at run time.

For more information on procedure calls, see Chapter 8, SQL Statements, in Volume 1 of the *Microsoft ODBC 3.0 Programmer's Reference and SDK Guide,* published by Microsoft Press, Redmond, Washington, 1997, ISBN 1-57231-516-4.

Related Topics

Import and Connect-Time Security Requirements

How the CONNX DB2 Module Maps ODBC to DRDA Isolation Levels

Performing Queries in Passthrough Mode in SQL Server Databases

Queries can be performed in passthrough mode through the SQL Server driver. Joins made between two SQL Server tables stored in the same SQL Server database are performed on the Windows SQL Server automatically. This enhancement increases the speed of join queries of this type, especially those that use a Where clause in the SQL statement. If a function is used within the query that does not exist in the SQL Server database, CONNX processes the query internally.

Important: *SQL functions not natively supported by SQL Server cannot be used with SQL passthrough.*

How CONNX handles Adabas Periodic Groups and Multi-Value Fields

CONNX creates two styles of SQL tables to represent ADABAS Files that contain MUs or PEs. 1) a flattenedable, and 2) a root table, and one or more rotated tables

Flattened Table

CONNX creates a single flattened table, where each occurrence of an MU or PE is a separate column. The sample ADABAS Employees table contains two period groups, and three multi value fields. One of the period groups contains a multi value field.

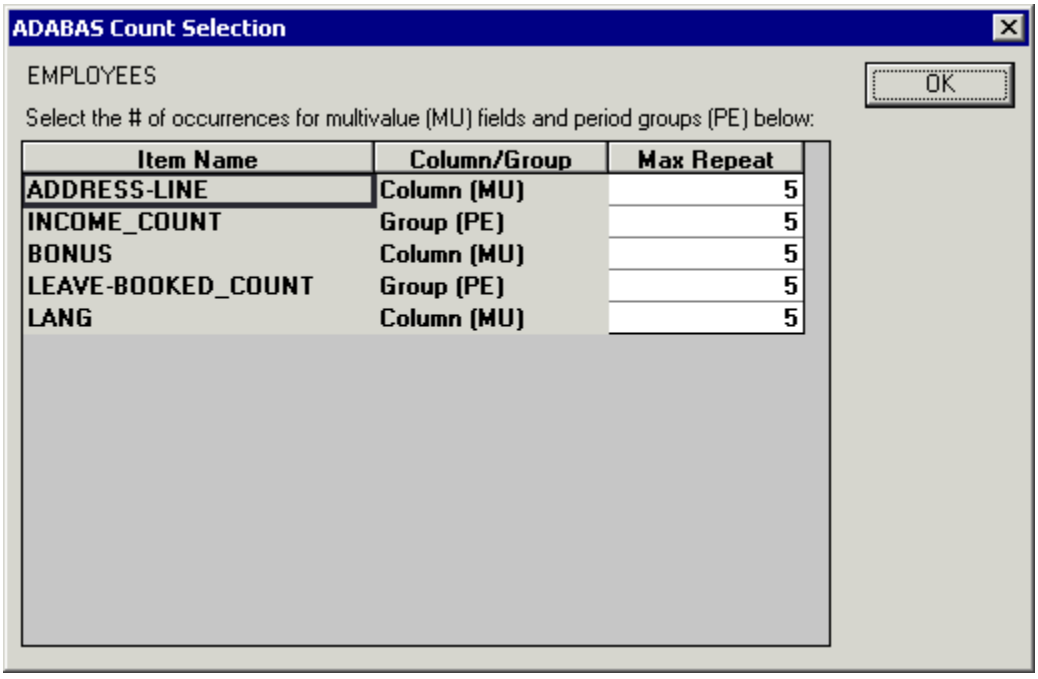
Figure 1: List of MUs and PEs in the Employees table

ADDRESS_LINE	Multi-value field
INCOME	Period group (This group contains a multi-value field called BONUS.)
BONUS	Multi-value field (within the INCOME period group)
LANG	Multi-value field
LEAVE BOOKD	Periodic Group

ADABAS currently has a limit of 191 occurrences for any MU or PE group. If we were to use that limit to fully flatten out the employees table, it would have 37,842 columns. This is primarily because there is a multi value field within a period group, which effectively squares the number of columns in the table. Obviously, a table with 37,842 columns is not very meaningful. Additionally, CONNX has limit of 30,000 columns for any SQL table.

In most cases, even though the maximum # of occurrences possible is 191, the actual number is far less. During the import, CONNX displays a dialog that allows you to specify the number of occurrences to be flattened for every MU and PE within the table being imported. For example, you will get the following dialog for the employees table:

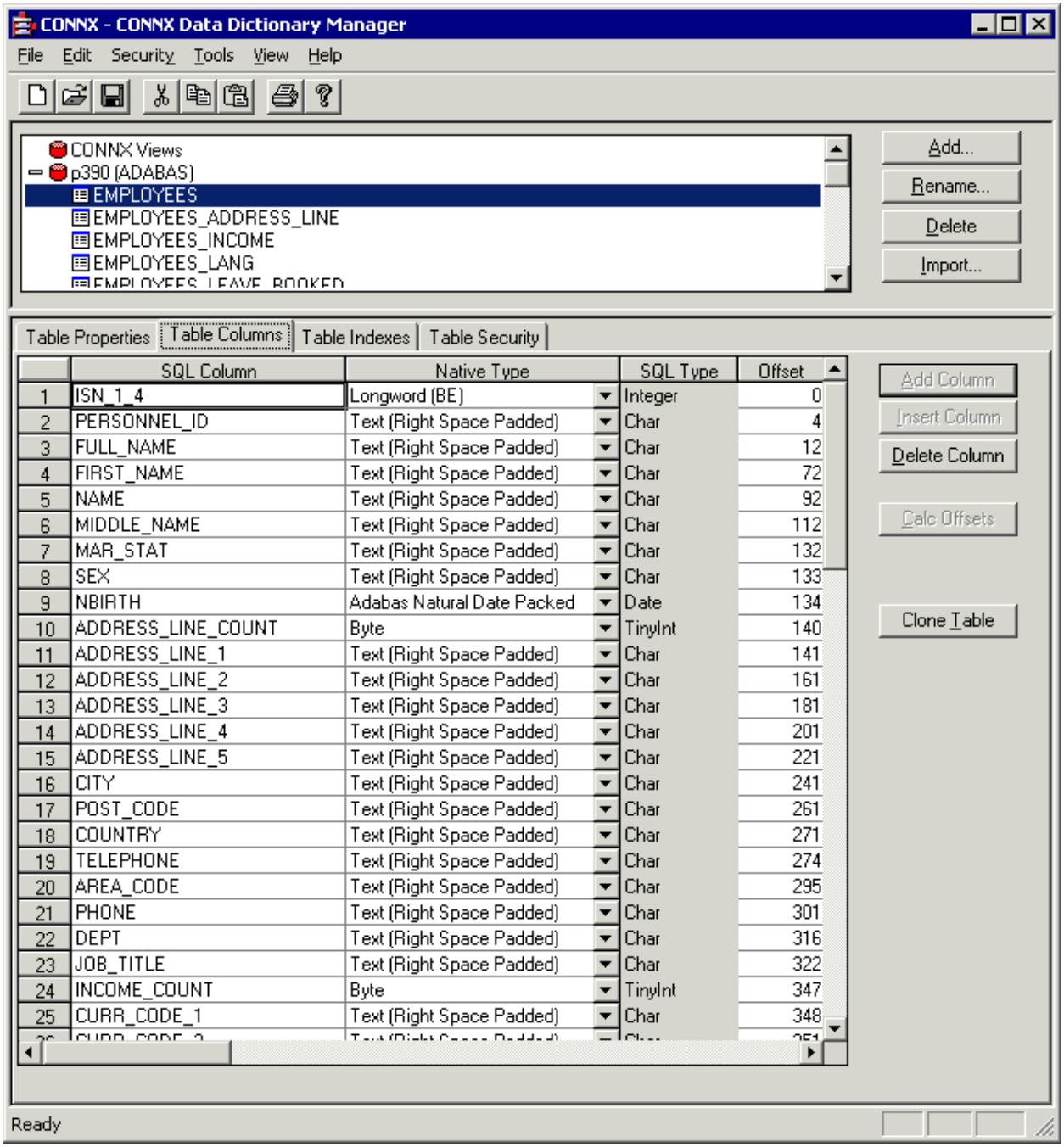
Figure 2: Max repeat count dialog displayed during import of the employees table



By default, CONNX flattens the first 5 occurrences of each periodic group and multi value field. If you need more occurrences flattened, adjust the max repeat count during the import process on the dialog above.

After the import complete, you will see a flattened table, where the occurrences of each MU and PE are represented as separate columns.

Figure 3: Flattened Employee Table



Rotated Tables

CONNX creates rotated (non-root) tables, one for each top level MU and PE within the ADABAS file.

After CONNX performs the import, in addition to the root table, there will be new tables for every MU, PE, and MUPE (MU within a PE). CONNX has converted the ADABAS file hierarchical structure into relational tables. SQL calls can be made to these relational tables.

CONNX uses the ISN number as the primary unique key for all ADABAS tables. The ISN is always the first field of any CONNX ADABAS table. This ISN field has a naming convention of ISN_<*database id*>_<*file id*>.

Example: The ISN field for file #4 contained within database #12 would be ISN_12_4.

Each rotated table will have a unique primary key that combines the ISN number with at least one array occurrence (psuedo) column.

CNXARRAYCOLUMN and CNXARRAYCOLUMN_2

All rotated tables have at least one extra pseudo column called CNXARRAYCOLUMN.

For MU or PE tables, CNXARRAYCOLUMN is a zero-based numeric field that indicates which occurrence of the MU/PE is represented by the record. CNXARRAYCOLUMN will be inserted into the table right after the ISN number.

If the rotated table happens to be a MUPE table, there will be two extra psuedo columns for every row - CNXARRAYCOLUMN and CNXARRAYCOLUMN_2:

- CNXARRAYCOLUMN is the first array occurrence column. It refers to the specific PE row.
- CNXARRAYCOLUMN_2 is the second array occurrence column. It refers to the specific MU occurrence within the Periodic Group of which it is a child of. It will be inserted into the table right after CNXARRAYCOLUMN.

The type of rotated table determines how many fields the primary unique key will contain:

- The primary unique key for MU and PE rotated tables always contains two fields: the ISN and CNXARRAYCOLUMN.
- The primary unique key for MUPE rotated tables always contains three fields: the ISN, the CNXARRAYCOLUMN, and the CNXARRAYCOLUMN_2.

Example:

Since the sample employees table contains 4 top level MU/PEs, CONNX creates 4 rotated tables and 1 root table.

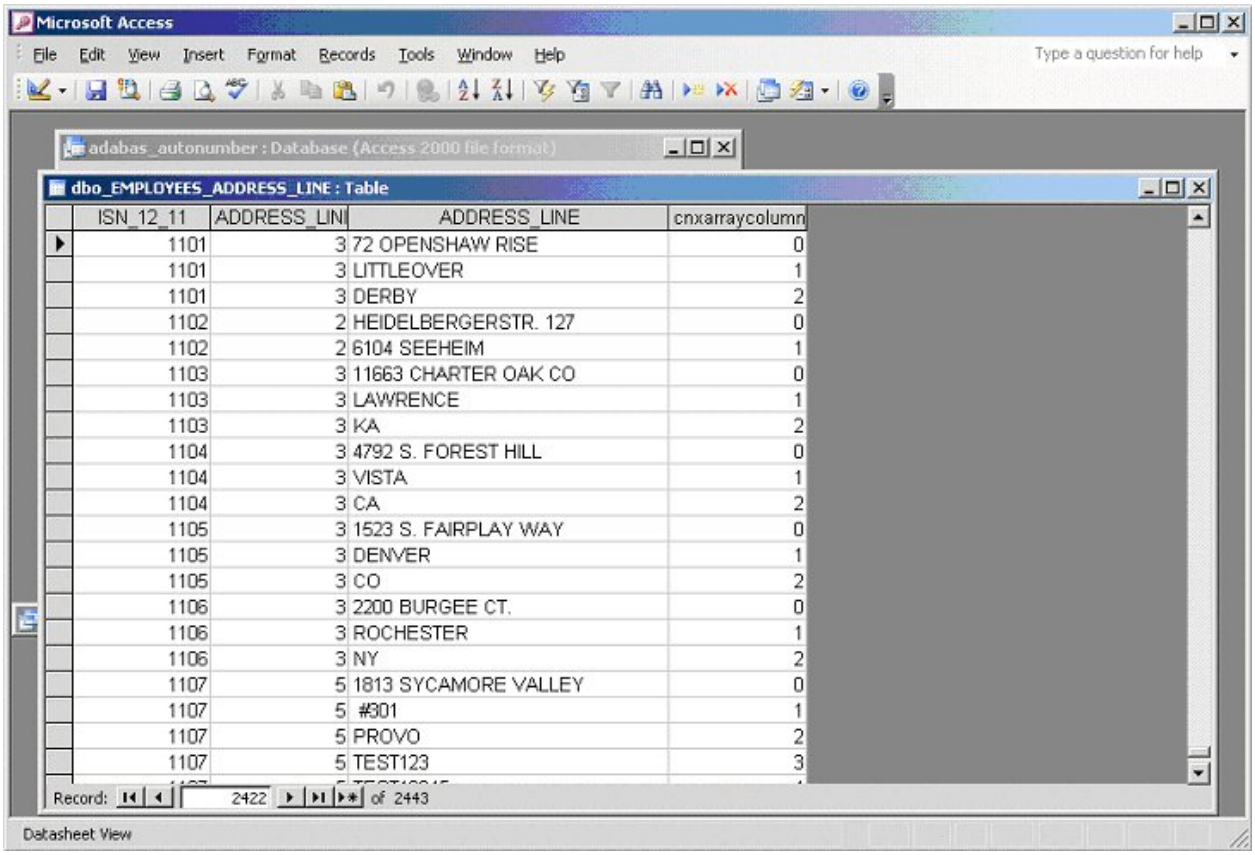
Figure 1: List of SQL tables created by CONNX for the sample employees table

EMPLOYEES_ROOT	The root table
EMPLOYEES_ADDRESS_LINE	The rotated table that represents the ADDRESS_LINE multi-value field.
EMPLOYEES_INCOME	The rotated table that represents the INCOME period group, and also the BONUS multi-value field contained within the periodic group.
EMPLOYEES_LANG	The rotated table that represents the LANG multi-value field.
EMPLOYEES_LEAVE_BOOKED	The rotated table that represents the LEAVE_BOOKED period group.

EMPLOYEES	The flattened version of the employees table, discussed in the prior section.

In Figure 2, you can see that for the physical ADABAS record with an ISN of 1101, there are 3 address lines. Accordingly, three records are returned for this ISN, each with a cnxarraycolumn of 0, 1, and 2 respectively.

Figure 2: Example of rotated array in Microsoft Access

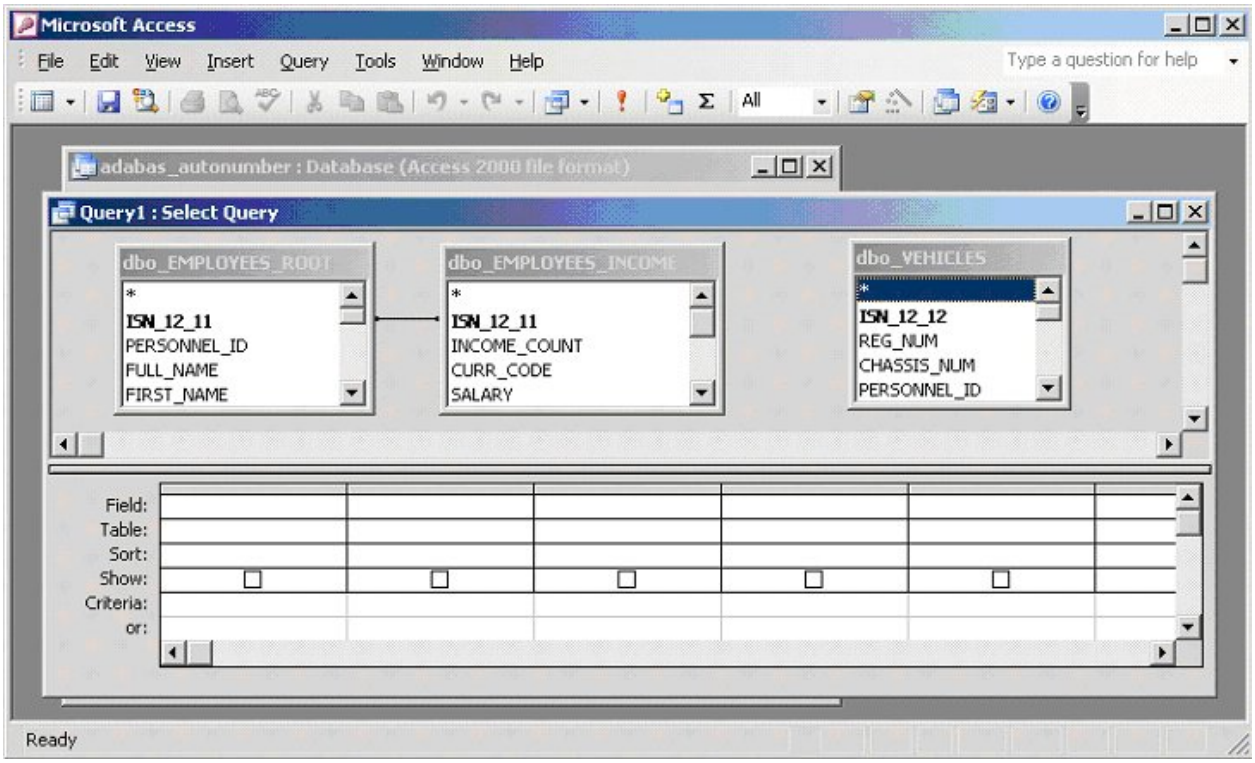


ISN_12_11	ADDRESS_LIN	ADDRESS_LINE	cnxarraycolumn
1101	3	72 OPENSHAW RISE	0
1101	3	LITTLEOVER	1
1101	3	DERBY	2
1102	2	HEIDELBERGERSTR. 127	0
1102	2	6104 SEEHEIM	1
1103	3	11663 CHARTER OAK CO	0
1103	3	LAWRENCE	1
1103	3	KA	2
1104	3	4792 S. FOREST HILL	0
1104	3	VISTA	1
1104	3	CA	2
1105	3	1523 S. FAIRPLAY WAY	0
1105	3	DENVER	1
1105	3	CO	2
1106	3	2200 BURGEE CT.	0
1106	3	ROCHESTER	1
1106	3	NY	2
1107	5	1813 SYCAMORE VALLEY	0
1107	5	#301	1
1107	5	PROVO	2
1107	5	TEST123	3

In order to retrieve data from both MU/PE fields, and non-MU/PE fields, you must join the "ROOT" version of the table with the appropriate rotated representations of the desired MU and PE fields using the ISN field. Many GUI query tools such as Microsoft Access and Crystal Reports attempt to guess which fields link tables together based on field names, and whether they are a part of a unique index. When you drag and drop the root table and a rotated table into the query designer of any of these GUI query tools, they correctly link the tables together based on name of the ISN field. Most importantly, the tools do not erroneously link unrelated tables together, because the name of the ISN field varies between ADABAS files.

Figure 3 illustrates how Access properly links the Root and the rotated table together based on the ISN. The ISN of the vehicles tables cannot be linked to the employees table, because the field names are different.

Figure 3: Auto linking of fields in Microsoft Access



In order to link the Employees to the Vehicles table, drag and drop the personnel_id field from one table to the other. Linking across different tables does not occur automatically.

Inserting data into MU and PE fields

When the record does not exist:

When using a flattened version of the table, you must specify all of the data to be inserted for the entire record using a single SQL statement. This process is straightforward, as each occurrence of a MU or PE field is a separate column.

When using the rotated and root version of the table, several separate insert statements may be required to insert a new record into ADABAS.

First, you must insert a record into the Root version of the table.

Then, using the ISN returned from the Root insert, you insert data into the appropriate rotated tables.

Example:

```
Insert into employees_root (personnel_id, first_name, last_name) values
   ('12345678', 'John', 'Smith')
```

Use the following SQL statement to determine the ISN of the last record inserted into ADABAS by CONNX:

```
Select @@IDENTITY
```

Let's say for this example, the above query returns 1004, indicating that the ISN number for the record just inserted is 1004.

Using this ISN number, you can now insert data into the MU and PE fields for this record.

```
Insert into employees_lang (ISN_12_11, LANG) values (1004, 'ENG')
Insert into employees_lang (ISN_12_11, LANG) values (1004, 'FRE')
Insert into employees_lang (ISN_12_11, LANG) values (1004, 'SPA')
```

The above 3 SQL statements add 3 occurrences to the LANG multi-value field for the ADABAS record with ISN # 1004.

When the record already exists:

When using flattened version of the table, the only way to add more occurrences to a given record is to use a SQL update statement. Using the update statement, simply provide a value for the column occurrences you want to add. For example, if a record in the employees table currently has a single occurrence in the LANG MU field, and you want to add a second occurrence, use the following SQL statement (you must know the ISN number of the existing record you want to modify):

```
update employees set LANG_2 = 'FRE' where ISN_12_11 = 1004
```

When using the rotated and root version of the table, a SQL insert statement is used to add an occurrence to an existing ADABAS record. Using the same example as above with the rotated version of the tables, the SQL statement would look like this:

```
insert into employees_lang (ISN_12_11, LANG) values (1004, 'FRE')
```

NOTE: Inserting a value into the SQL count fields that return the # of occurrences in a MU/PE will not cause an error. However, the value will be ignored. The # of occurrences will be strictly be determined by the amount of data inserted into the MU/PE group.

Updating data in existing MU and PE fields

When using flattened version of the table, updates to MU and PE fields are accomplished by using the update SQL statement. Simply provide a new value for the column occurrence you want to modify. For example, if a record in the employees table currently has two occurrences in the LANG MU field, and you want to modify the second occurrence, use the following SQL statement (you must know the ISN number of the existing record you want to modify):

```
update employees set LANG_2 = 'ENG' where ISN_12_11 = 1004
```

When using the rotated and root version of the table, a SQL update is also used to modify the data. The desired occurrence for modification must be specified using the cnxarraycolumn pseudo field. This pseudo field is zero based. So if you want to modify the first occurrence of a MU field, you would specify cnxarraycolumn=0 in the where clause of the update statement. If you want to modify the second occurrence of a MU field, you would specify cnxarraycolumn=1 in the where clause of the update statement.

Using the same example above, updating the second occurrence of the LANG MU field would require the following SQL statement.

```
update employees_lang
set LANG = 'ENG'
where ISN_12_11 = 1004 and cnxarraycolumn = 1
```

NOTE: Updating the SQL count fields that return the # of occurrences in a MU/PE will not cause an error, but any attempts to change the count field directly using an update statement will be ignored.

For example: the following SQL statements will have no effect on the ADABAS record:

```
update employees set LANG_COUNT = 4 where ISN_12_11 = 1004
update employees_lang set LANG_COUNT = 6 where ISN_12_11 = 1004
```

Deleting data in existing MU and PE fields

Note: It is not possible using ADABAS direct calls to delete occurrences of PE fields, and therefore we cannot offer this capability through SQL.

The rest of this section will describe how to delete occurrences of MU fields.

When using flattened version of the table, deletes of MU fields are accomplished by using the update SQL statement. Simply blank (if the data type is text) or zero (if the data type is numeric) out the field occurrence you want to delete.

For example, if a record in the employees table currently has two occurrences in the LANG MU field, and you want to delete the 2nd occurrence, use the following SQL statement (you must know the ISN number of the existing record you want to modify):

```
update employees set LANG_2 = '' where ISN_12_11 = 1004
```

When using the rotated and root version of the table, a SQL delete is used to remove the data for MU fields. The desired occurrence for deletion must be specified using the cnxarraycolumn pseudo field. This pseudo field is zero-based. So if you want to delete the first occurrence of a MU field, you would specify cnxarraycolumn=0 in the Where clause of the delete statement. If you want to delete the second occurrence of a MU field, you would specify cnxarraycolumn=1 in the where clause of the delete statement.

Using the same example above, deleting the second occurrence of the LANG MU field would require the following SQL statement.

```
delete from employees_lang
where ISN_12_11 = 1004 and cnxarraycolumn = 1
```

User Exit (Open Systems Only)

End users can construct CONNX user exits that will be called during Adabas processing. User exits can be call before each Adabas call, after each Adabas call, or both.

CONNX User exists are only available on Open Systems.

The User exist must be constructed as DLL on windows, or a shared library on unix.

To enable a user exit, the following configuration setting must be enabled on the Adabas server.

On unix, these should be added to the connxserver script as additional environment variables.

On windows, they should be added using the CONNX Configuration Manager under the CONNX\ADABAS key.

The following setting will determine when the user exit is called.

USEREXITFLAGS

1 = PRECALL - Call before every Adabas call

2 = POSTCALL - Call after every Adabas call

3 = PRECALL and POSTCALL - Call before and after every Adabas call

The following setting is the name of the 64bit user exit. Use this setting if the CONNX server is configured as 64bit.

USEREXITNAME64

The following setting is the name of the 32bit user exit. Use this setting if the CONNX server is configured as 32bit.

USEREXITNAME32

The only requirement of the user exit DLL is that it must have a C entry point with the name of UserExitFunc.

The UserExitFunc function has the following signature:

```
extern "C" int UserExitFunc(unsigned int threadid, // unique session number
                                        const char *userid,        //
                                        user name passed to the CONNX
                                        Adabas server
                                        const char * clientid,     //
                                        clientid (if any) passed to the
                                        CONNX Adabas server
                                        void * AdabasControlBlock, //
                                        the Adabas control block
                                        void * szFormatBuffer,     //
                                        the Adabas format buffer
                                        void * szRecordBuffer,     //
                                        the Adabas record buffer
                                        void *szSearchBuffer,      //
                                        the Adabas search buffer
                                        void *szValueBuffer,       //
                                        the Adabas value buffer
```

```
                                        void *szISNBuffer,          //
                                        the Adabas ISN buffer
                                        int nPrePost )              //
                                        integer flag to indicate whether
                                        this is a pre (0) or post (1)
                                        call.
```

Here is a C++ example of a user exit DLL function:

```
// adauserexit.cpp : Defines the exported functions for the DLL application
//
#include "stdafx.h"
#include "adauserexit.h"
#include <stdio.h>
#include <stdlib.h>
#pragma pack(1)
typedef       struct        _ADABASCONTROLBLOCK
{
       char cb_cmd_res[2];  // Reserved
       char cb_cmd_code[2]; // Command Code
       char cb_cmd_id[4];   // COMMAND ID
       short cb_file_nr;
       short cb_return_code;
       int cb_isn;
       int cb_isn_l1;
       int cb_isn_quantity;
       unsigned short cb_fmt_buf_lng;
       unsigned short cb_rec_buf_lng;
       unsigned short cb_sea_buf_lng;
       unsigned short cb_val_buf_lng;
       unsigned short cb_isn_buf_lng;
       char cb_cop1;
       char cb_cop2;
       char cb_add1[8];
       char cb_add2[4];
       char cb_add3[8];
       char cb_add4[8];
       char cb_add5[8];
       int cb_cmd_time;
       char cb_user_area[4];
} ADABASCONTROLBLOCK;
#pragma pack()
```

```
// This is an example of an exported function.
extern "C" ADAUSEREXIT_API int UserExitFunc(unsigned int threadid,
                                            const char *userid,
                                            const char * clientid,
                                            void * AdabasControlBlock,
                                            void * szFormatBuffer,
                                            void * szRecordBuffer,
                                            void *szSearchBuffer,
                                            void *szValueBuffer,
                                            void *szISNBuffer,
                                            int nPrePost )
{
       FILE * fp;
       fp = fopen("c:\\adaexit.log", "a+");
       ADABASCONTROLBLOCK * pBlock = (ADABASCONTROLBLOCK
*)AdabasControlBlock;
       fprintf(fp, "thread (%d), user(%s), clientid(%s), adabas
command(%2.2s)\n",
        threadid, userid, clientid, pBlock->cb_cmd_code);
       fclose(fp);
       return 0;
}
```

Summary of the Data Dictionary Comparison Tool

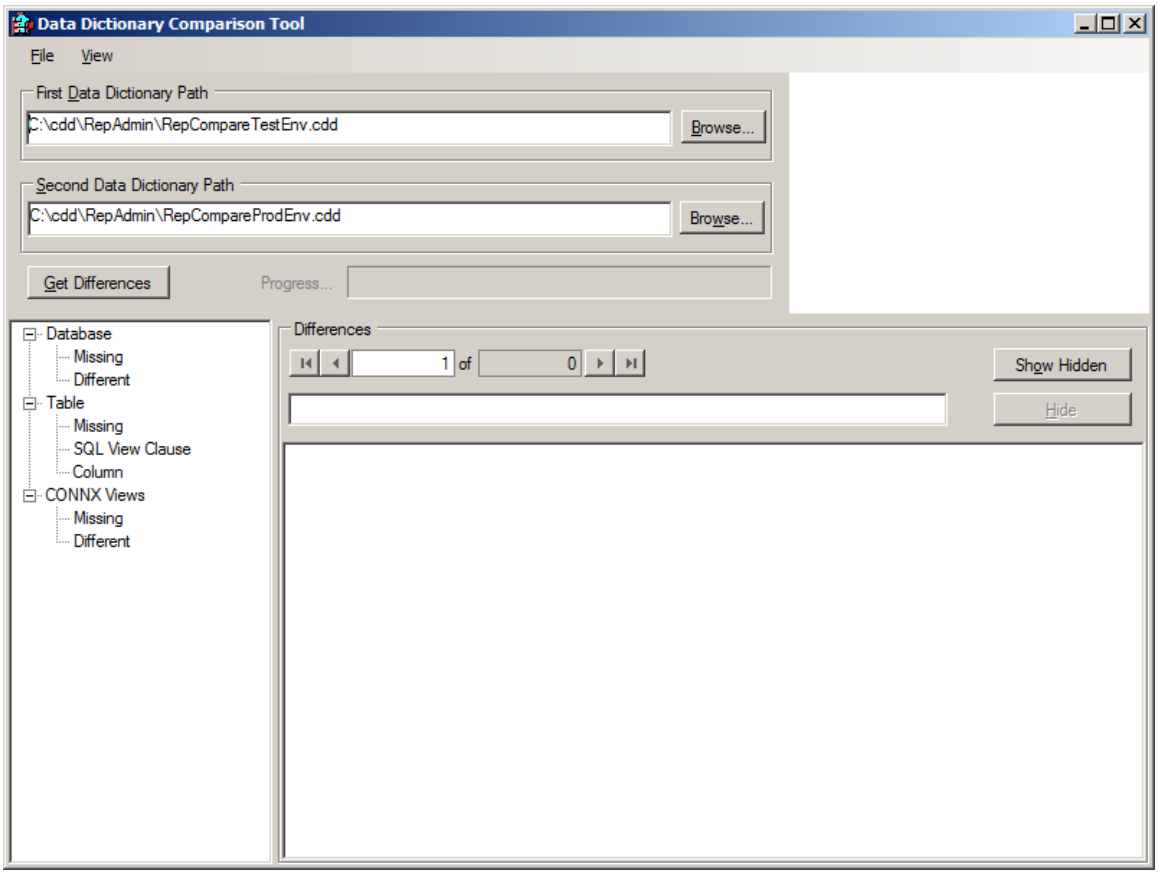
When maintaining multiple data dictionaries for different environments such as development, testing and production it can be useful to compare the contents of these data dictionaries to make sure the only differences are planned. The utility for this is the **Data Dictionary Comparison Tool** found in the **CONNX Driver** folder in the **Start Menu**. This tool displays database, table and CONNX View differences.

For data dictionaries used in Adabas Event Replication it also displays differences in replications and controllers. Once the tool is opened from the **Start Menu**, the user selects two different data dictionaries, then presses the **Get Differences** button and the differences will be put into categories with the details displayed one at a time.

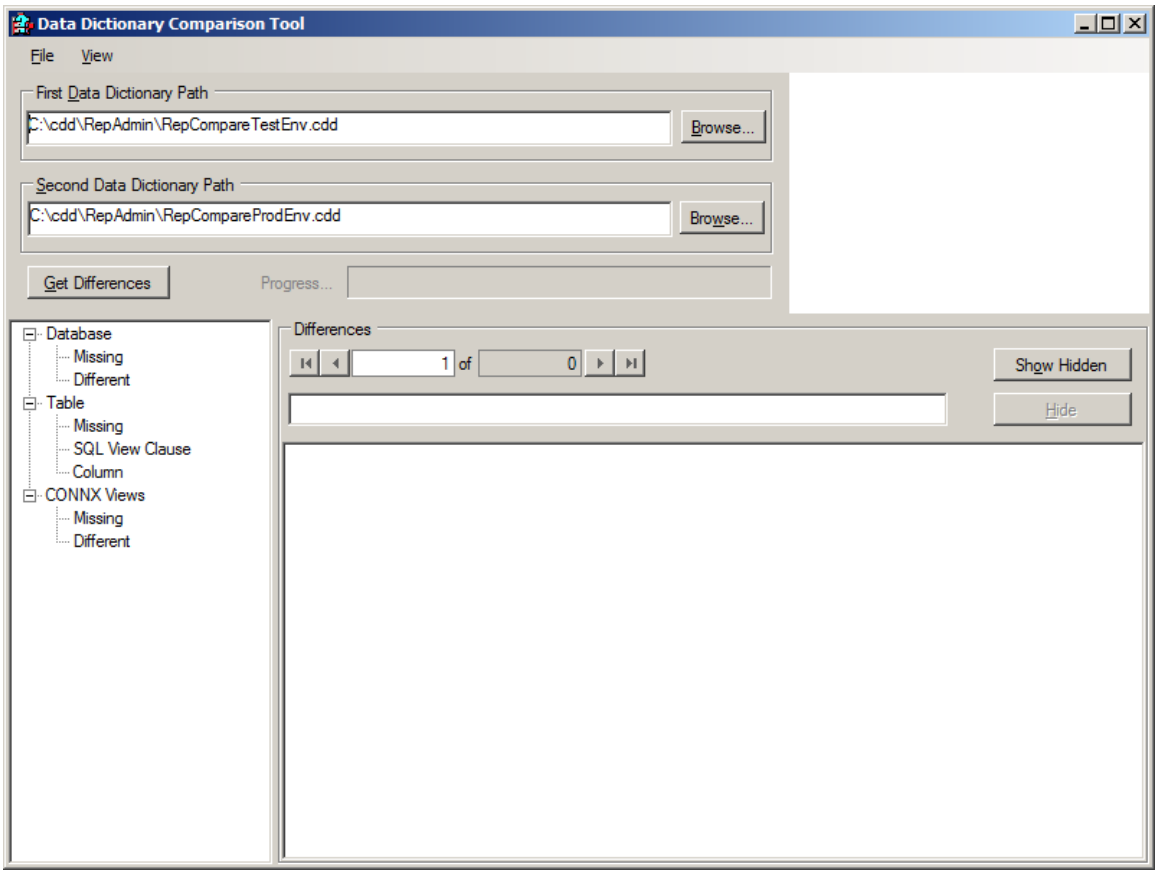
If unplanned differences are discovered, the data dictionary can be modified in the **CONNX Data Dictionary Manager.** For Replication or Controller differences, they need to be changed in the **Replication Administrator**.

Using the Data Dictionary Comparison Tool

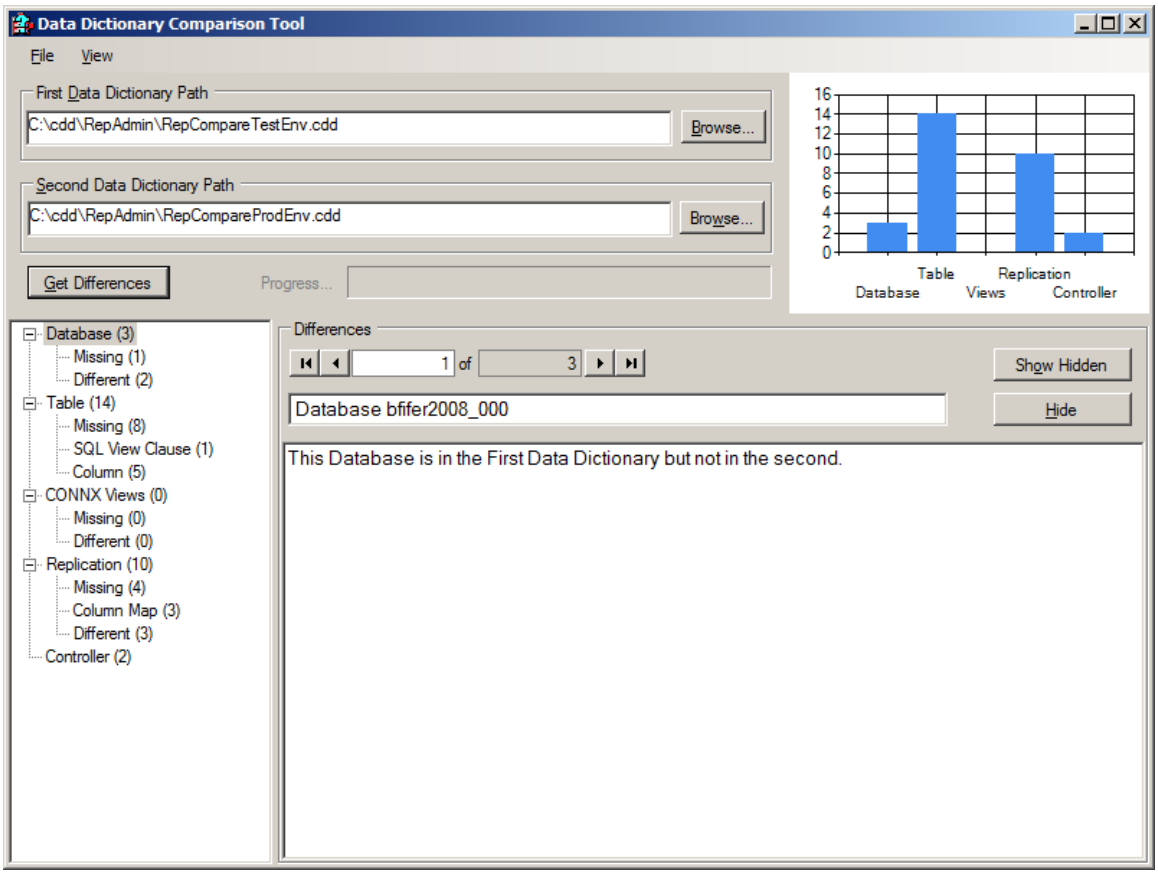
1. Open tool from the start menu, **CONNX Driver** --> **Data Dictionary Comparison Tool**. The first time it runs the First and Second Data dictionary paths will be blank, after it has been run once it will remember the last data dictionaries used and populate the path automatically. Enter information for the two data dictionaries you want to compare in the First and Second Data Dictionary Path entry fields. The path and name can either be typed in or use the browse button to find the correct data dictionary.



2. Once the data dictionaries are selected press the **Get Differences** button and the two data dictionaries will be compared with the results showing in the Difference Type tree and Differences panel below. There may be just a few or many differences, the differences are organized into categories and subcategories in the tree few. The first difference will be automatically selected with details about it displayed in the **Differences** panel.

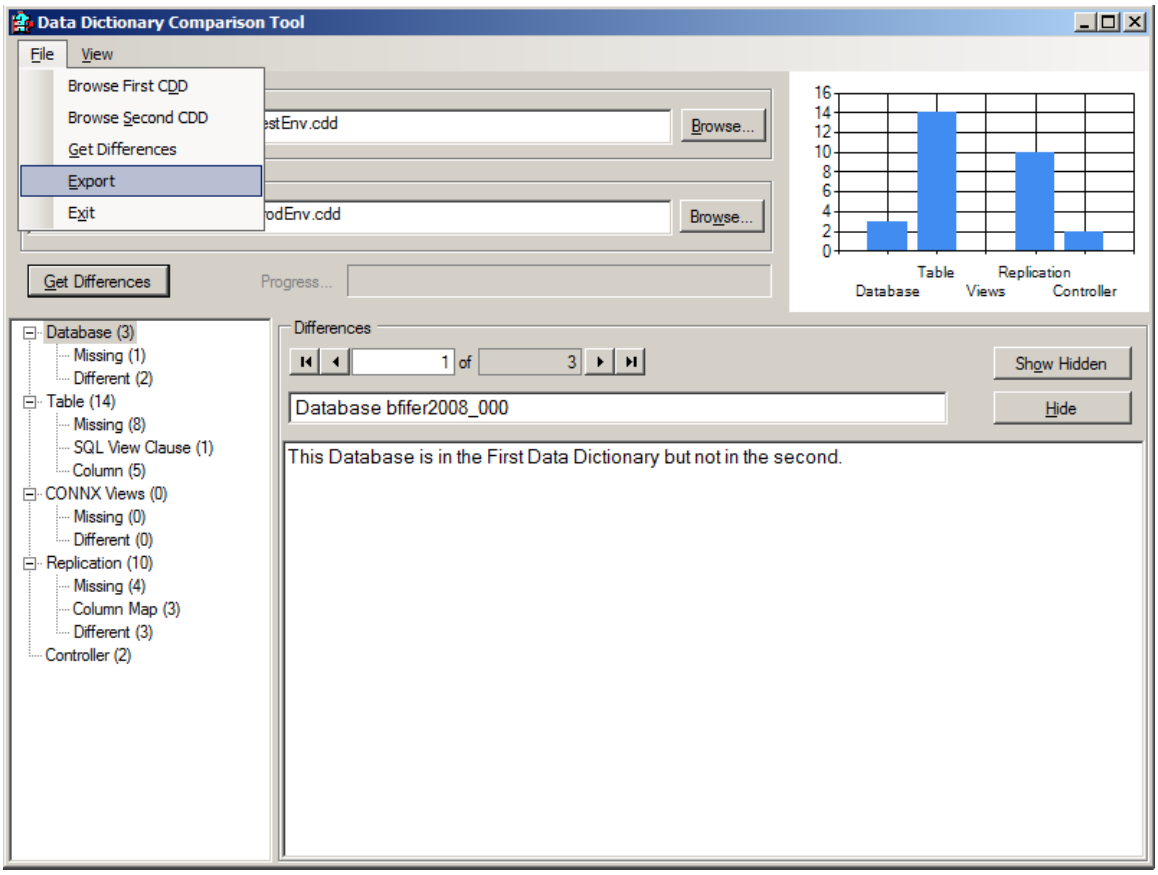


3. If the data dictionaries are used in Adabas Event Replication, they will have two more Difference Categories listed - Replication and Controller. When there are multiple differences found, they can be searched through and looked at one at a time. When a Difference Category is selected in the tree view the details of the difference are displayed to the right. If there are more than one difference in the Category the others can be seen by selecting the arrow buttons in the Differences panel to move back and forth through the list.



4. The **Hide** and **Show Hidden** buttons are to ease sorting the differences when there are many. If the user looks at the detail of a difference and knows that the difference was planned, press the **Hide** button and it will no longer be shown. The user can continue looking at the rest of the differences. To show all differences that have been previously hidden, press the **Show Hidden** button. Also, when the **Get Differences** button is pressed it resets any hidden differences and shows them.

5. To see all the differences at once in a text file use the **File** --> **Export** menu item. This will put all the differences into a Comma Delimited File (.csv) that can be opened by Excel or a text editor. All the differences will be listed with each difference on a separate row.


Data Dictionary Comparison Tool
File
View
Browse First CDD
Browse Second CDD
Get Differences
Export
Exit
estEnv.cdd
odEnv.cdd
Browse...
Browse...
Get Differences
Progress...
16
14
12
10
8
6
4
2
0
Database
Table
Views
Replication
Controller
Database (3)
Missing (1)
Different (2)
Table (14)
Missing (8)
SQL View Clause (1)
Column (5)
CONNX Views (0)
Missing (0)
Different (0)
Replication (10)
Missing (4)
Column Map (3)
Different (3)
Controller (2)
Differences
1 of 3
Show Hidden
Hide
Database bfifer2008_000
This Database is in the First Data Dictionary but not in the second.

Category Definitions for the Data Dictionary Comparison Tool

In the Data Dictionary Comparison Tool there are Categories of differences, this is a list of descriptions for those differences.

Database - Missing: Databases that are in one data dictionary but not the other.

Database - Different: Database is present in both data dictionaries, but some attributes are not the same, like a different **Physical Database Name**.

Table - Missing: Tables in one data dictionary but not in the other. One difference is listed for each table that is different.

Table - SQL View Clause: This compares the SQL View Clause that can be found on the **Table Properties** tab in the **CONNX Data Dictionary Manager**. If they do not match it is reported here.

Table - Column: If a table is in both data dictionaries but there are differences in some of the columns of the table, it will be listed here. Each entry is for one table, in the details section it will show all the columns in the table that have differences. Differences include columns that are in one table but not the other, and columns with different attributes - datatype length, precision and scale.

CONNX Views - Missing: Views that are created in the data dictionary manager that are in one data dictionary but not the other.

CONNX Views - Different: The CONNX View is in both data dictionaries but the SQL syntax that defines it is not the same.

If the data dictionaries being compared are used in Adabas Event Replication:

Replication - Missing: Replications that are in one data dictionary but not the other.

Replication - Column Map: Replications that are in both data dictionaries, but the column mapping is not the same. Each difference entry in this category is for one replication, so if there are multiple column map differences for one replication they will all be listed in the details for one difference. Differences include columns that are mapped in one replication but not the other, or source or target column attributes are different.

Replication - Different: Replications that are in both data dictionaries, but may have a different source or target table.

Controller: Differences in the replication controller between the two data dictionaries, like the controller name or engine count.

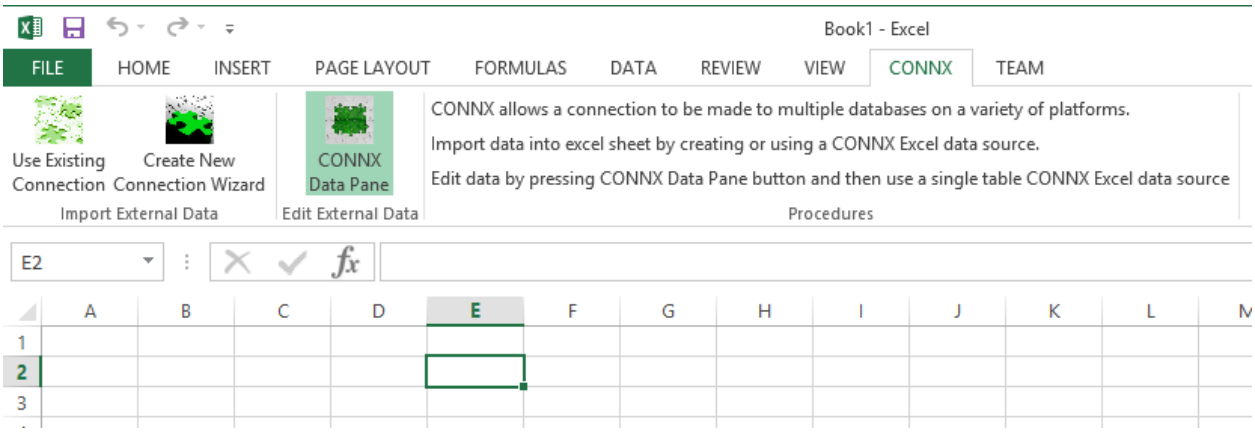
Summary of the CONNX Excel Add In

The **CONNX Excel Add In** can be used to edit or import data from a database table into an Excel spreadsheet. The Excel Add In provides easy to follow steps to create the Excel data connection to the CONNX source and get the data from a single or multiple tables. If the data is just imported into Excel and changes are not required to be sent to the database then a sql statement can be used to join multiple tables or limit the data shown. When editing data in a table only a single table can be used per worksheet so the changes can be sent to the correct database table.

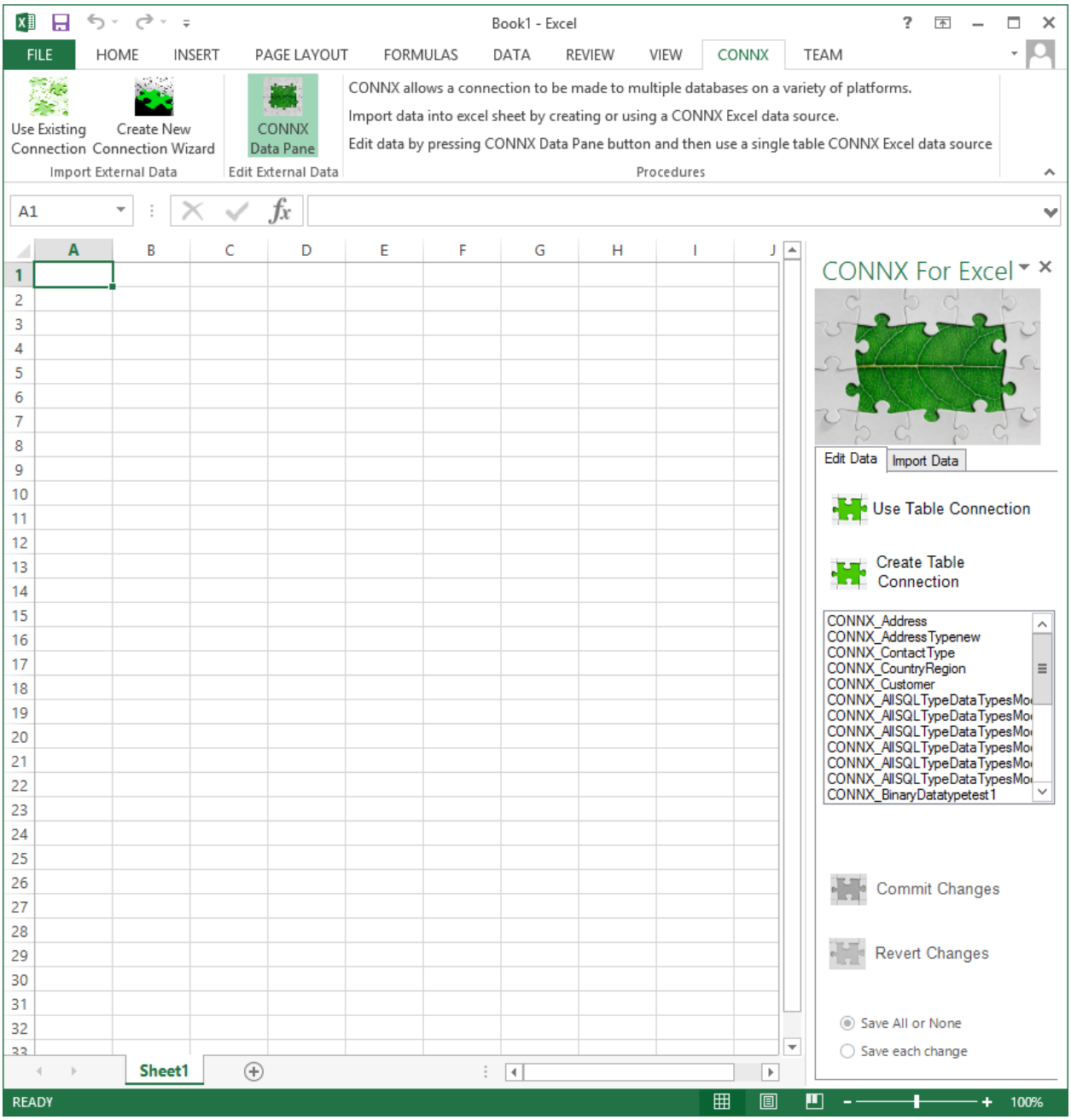
The **CONNX Excel Add In** installs with the main CONNX installation, it does not appear in the CONNX menu, it is loaded when Excel starts. When Excel starts a CONNX tab in the top ribbon appears. Selecting this CONNX tab will show the CONNX ribbon, in the ribbon an Excel connection to a CONNX data source can be made or used to import data into the spreadsheet. In order to edit data, in the CONNX ribbon press the CONNX data task pane button and the CONNX task pane will show to the right of the Excel sheet. The CONNX task pane must be used to bring data into excel that will be edited with the changes sent to the database.

Using the CONNX Excel Add In

Installing the CONNX Excel Add In it will make a CONNX tab appear in the top menu bar of Excel, press it to show the CONNX ribbon functions. Data can be imported using the "**Use Existing Connection**" and "**Create New Connection Wizard**" buttons. These will import the data into the excel spreadsheet, the data in the excel worksheet can be edited and the excel sheet saved, however those changes do not affect the database table.

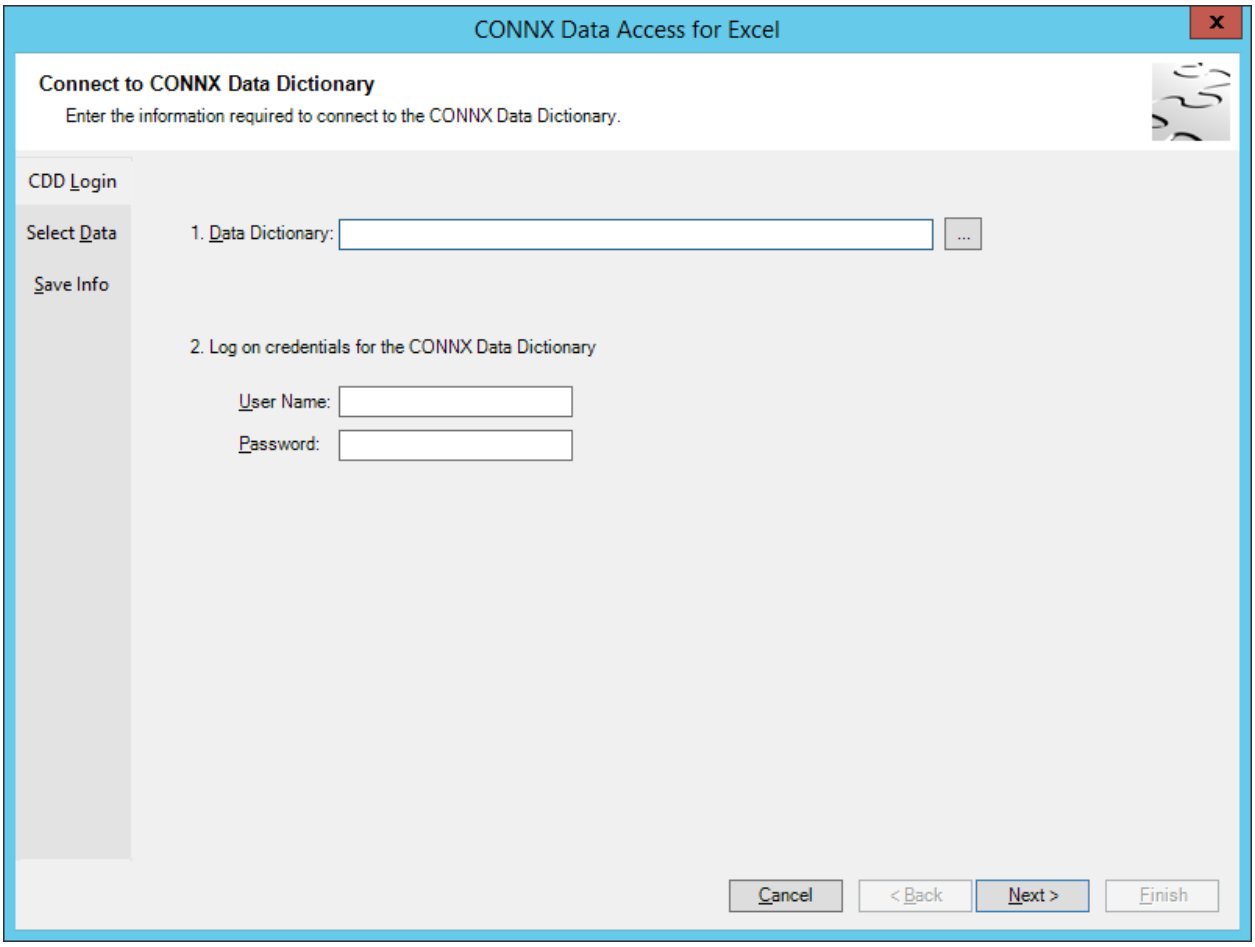


To **Edit data** in Excel and send those changes directly to the database a table connection must be made through the data task pane. In the CONNX Ribbon, press the "**CONNX Data Pane**" button and the CONNX Data Task pane will show on the right side of the worksheet.

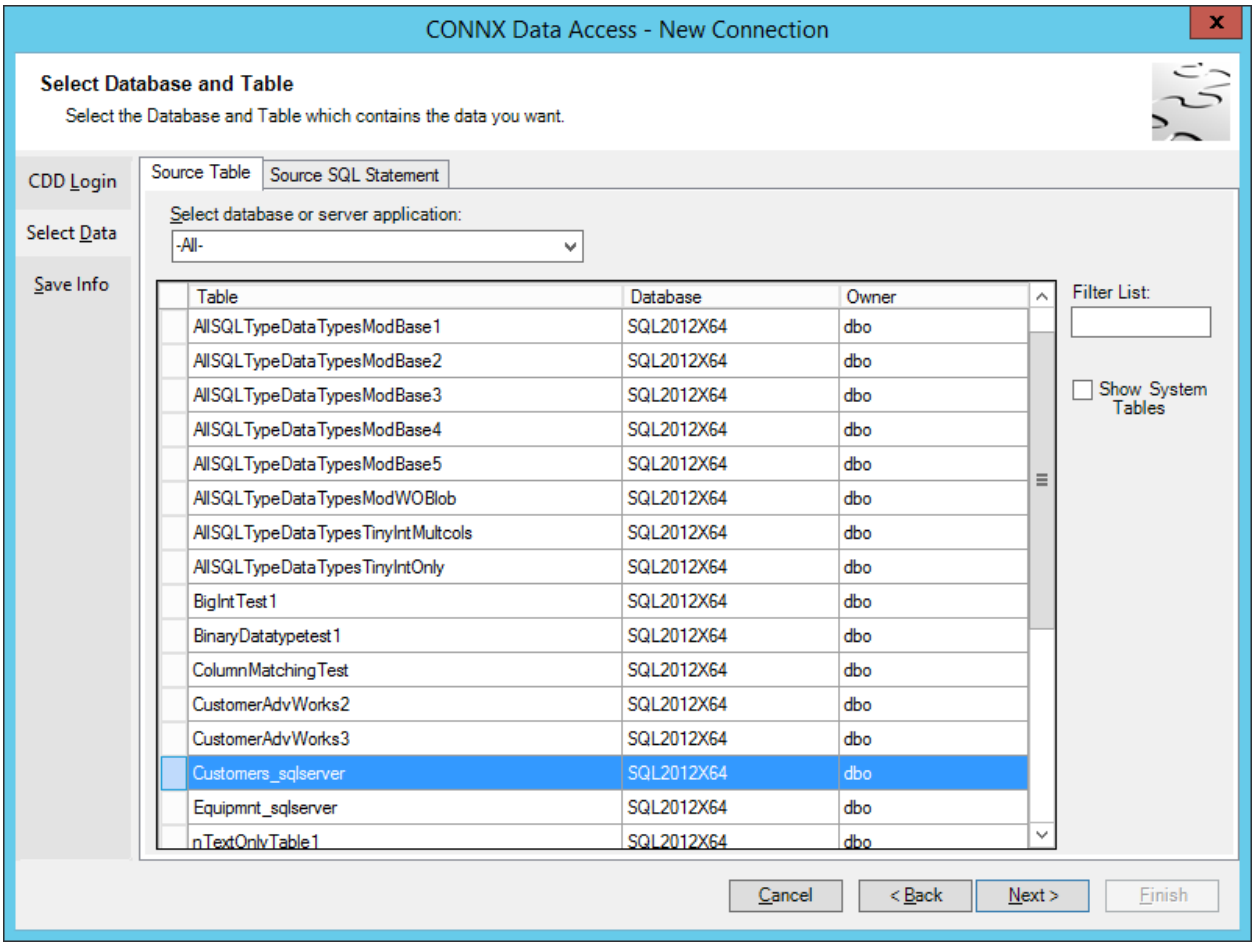


In the Data task pane there is an "**Edit Data**" tab, this is used to make a connection to a table in a database, import the data, then track and send the changes back to the database. In order to track and send the changes correctly, the connection here must be made to only one table. Joins and aggregates are not allowed in these connections. The dropdown list will show the friendly name of the existing connections that specify a single table only. If there are no existing connections a new table connection can be made by pressing the "**Create Table Connection**" button.

That brings up a form to walk through creating Excel connection to a CONNX datasource. It is a three step process, select the Data Dictionary with the data source in it and specify the user name and password to log into the data dictionary.



Then there will be a list of tables to select the one to edit the data. Note when importing data only there is an additional tab for using a SQL Statement as a source. A single table must be selected in order to enable data editing and allow changes to be sent to the database.

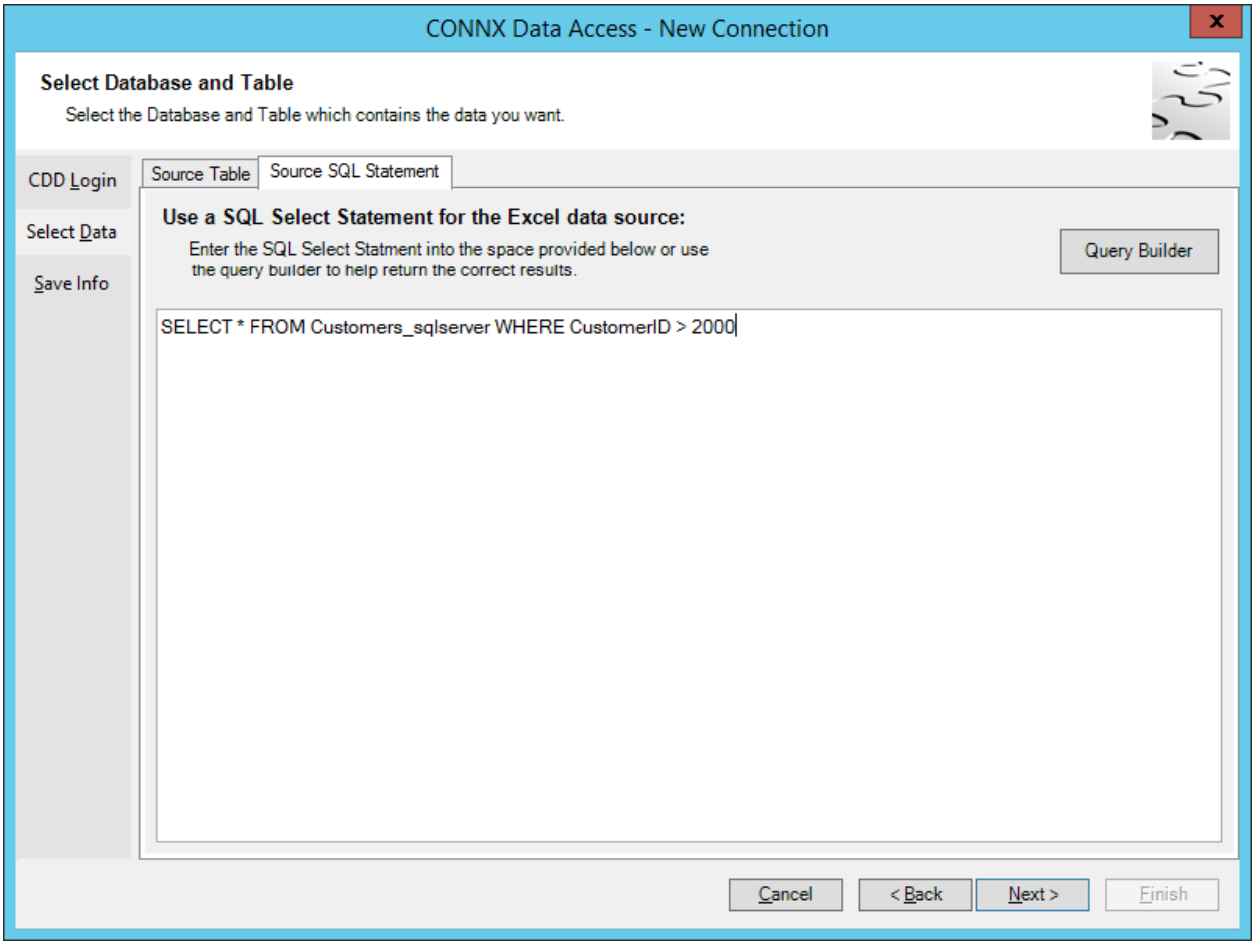


To use a select statement for the source, select the "Source SQL Statement" tab. This is useful for when the user wants to limit the amount of rows returned to Excel for edting.

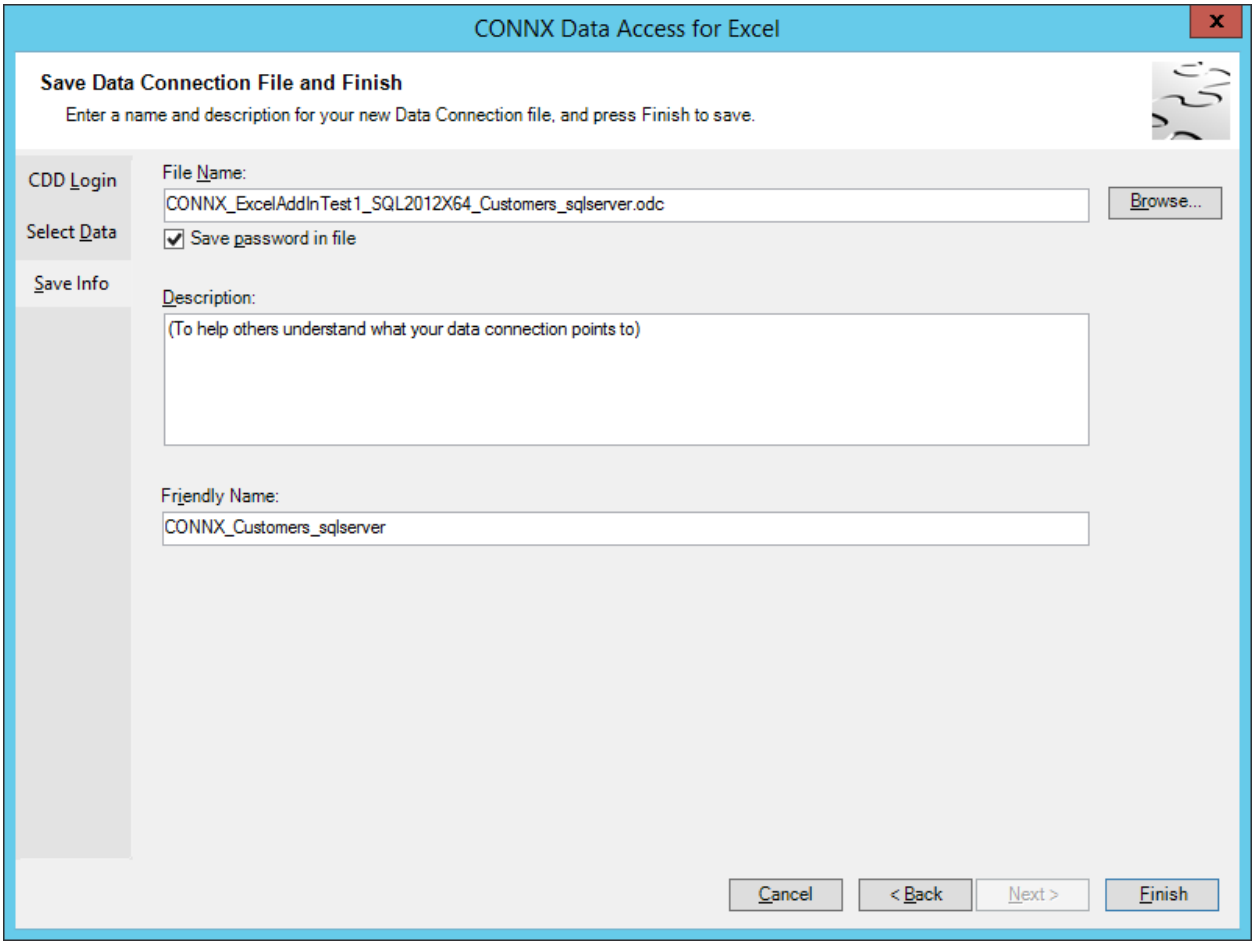
* **Caution:** When using a SQL Statement as a source for editing data the following rules apply:

Select statements cannot contain table joins, aggregates, sql functions or column alias', or any other SQL grammer that will make it impossible for the correct row in the source table to be determined while moving changes to the database.

Select statements must contain enough columns to make the data being edited unique, if not, the changes will fail because the correct row in the table will not be found.



The final screen allows customizing how the new data connection information is saved. The odc connection files are usually saved in the user\My Data Sources folder however they can be saved elsewhere. An important part of this screen is the **Friendly Name**, this is what is shown in the CONNX data task pane list. Choosing a descriptive Friendly Name will help to pick it out from the list of available CONNX ODC connections. Uncheck "**Save password in file**" to not have the Data Dictionary password written to the odc file. A registry entry can be used to disable this checkbox all the time, so no passwords are saved and the user must log in every time they load or refresh data. The registry value can be added in the Configuration manager under the CONNX key, if "**EXCELADDINNOSAVEPASSWORD**" is added and set to 1 no passwords will be able to be saved. If set to 0 or not there, then users can save passwords to odc files.



After pressing "**Finish**" on the Create Table Connection or Selecting a Table Connection for edit, the data is loaded into the Excel spreadsheet. Also this sheet is now enabled for editing and tracking changes to be sent to the database. Now that the sheet is in edit mode, the Commit and Revert changes buttons are now enabled. After changes have been made in the sheet, pressing the Commit changes button will bring up a screen with a list of all the changes in the order they happened. From there they can be sent to the database or skipped. The "**Revert Changes**" button will get rid of all changes made to the data in the sheet and reload the data from the database table, so not only does it undo changes it also gets the most recent data in the table.

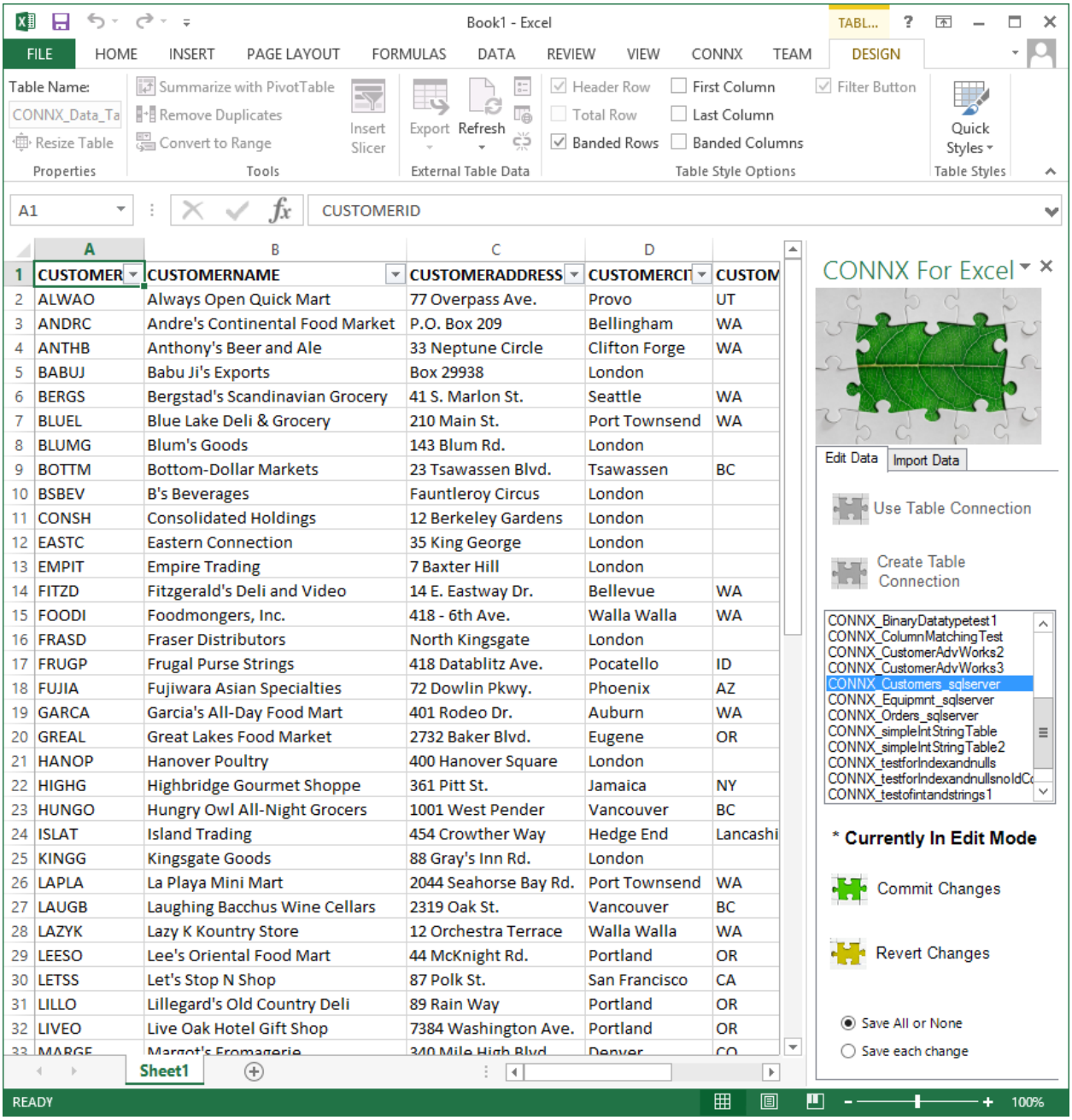
To change data in a cell simply click in the cell change to edit the cell and change it's contents. When finished click off the cell to exit the cells edit mode.

To delete a row, select the entire row or all the cells in the row and either press the delete key or right click and press delete.

To insert a new row, there is a blank row at the end of the data that has been imported and it has lines above and below it. Start at the first cell and enter data in each cell, the data will not be prepared to be put in the table until a cell in a different row is clicked on. Then the blank insert row will appear below the row that was just added.

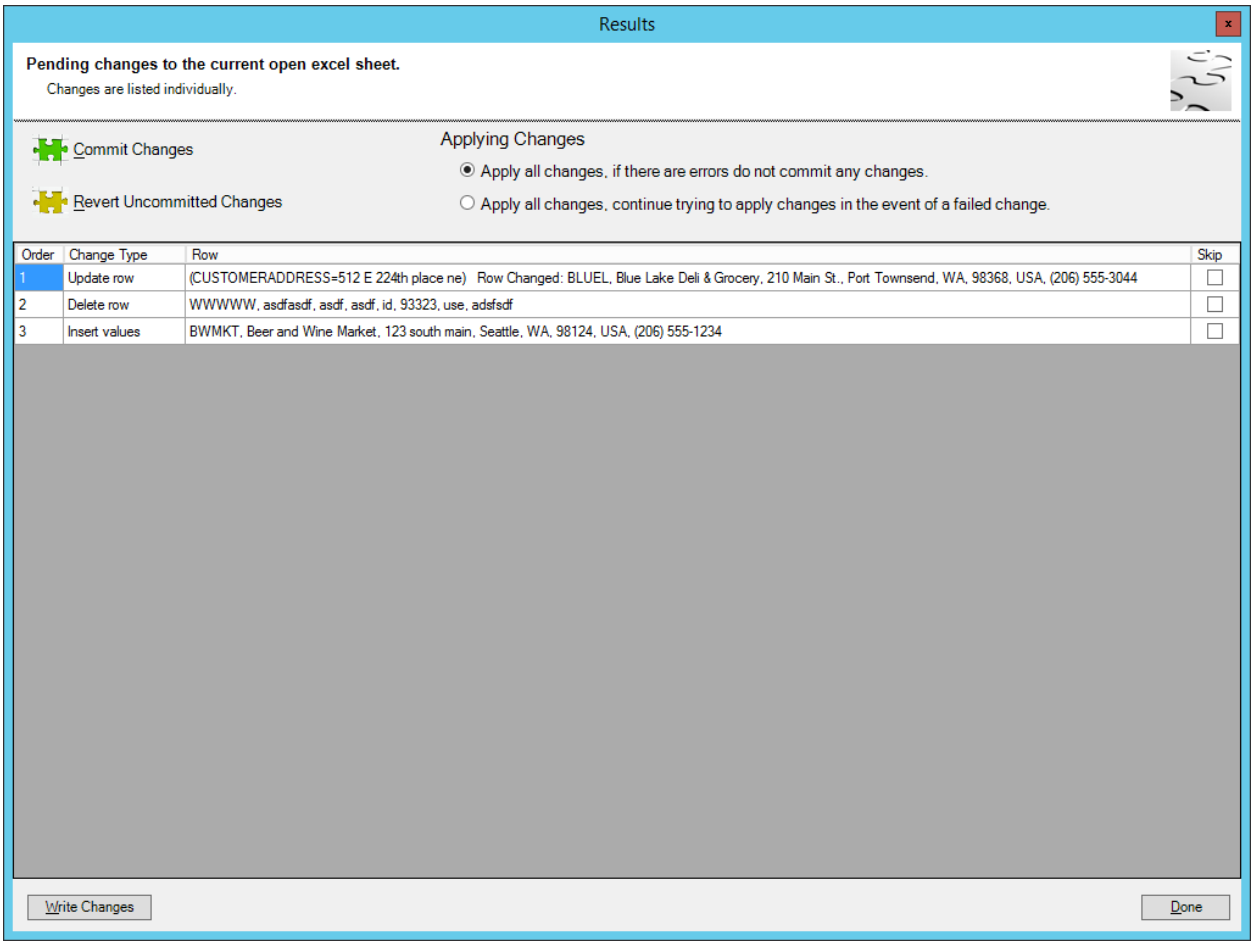
A few notes on inserting a new row, If only a few cells have had data entered in them and a different row is clicked, only the data in those cells will be inserted, even if the user goes back to the row and changes the rest of the blank cells. This is because the Data Task pane keeps track of the order of changes and

will treat it as an insert and update. If this happens, the user can skip the inserted row that was not complete when sending the changes to the database.



After changes to the data in the sheet have been made, pressing the "**Commit Changes**" button will bring up a screen that lists the changes and the order in which they happened. The changes can be reviewed and if correct, pressing the Commit Changes button will send the changes to the database. There are two options when sending the changes, all succeed or none, or each individual succeeds or fails. Like the description, if all the changes do not succeed, then none or committed to the database. This can be useful if later changes are dependant on earlier changes going into the table it stops if there is an error. If the changes are independent of each other, then it can be set to continue on sending and committing

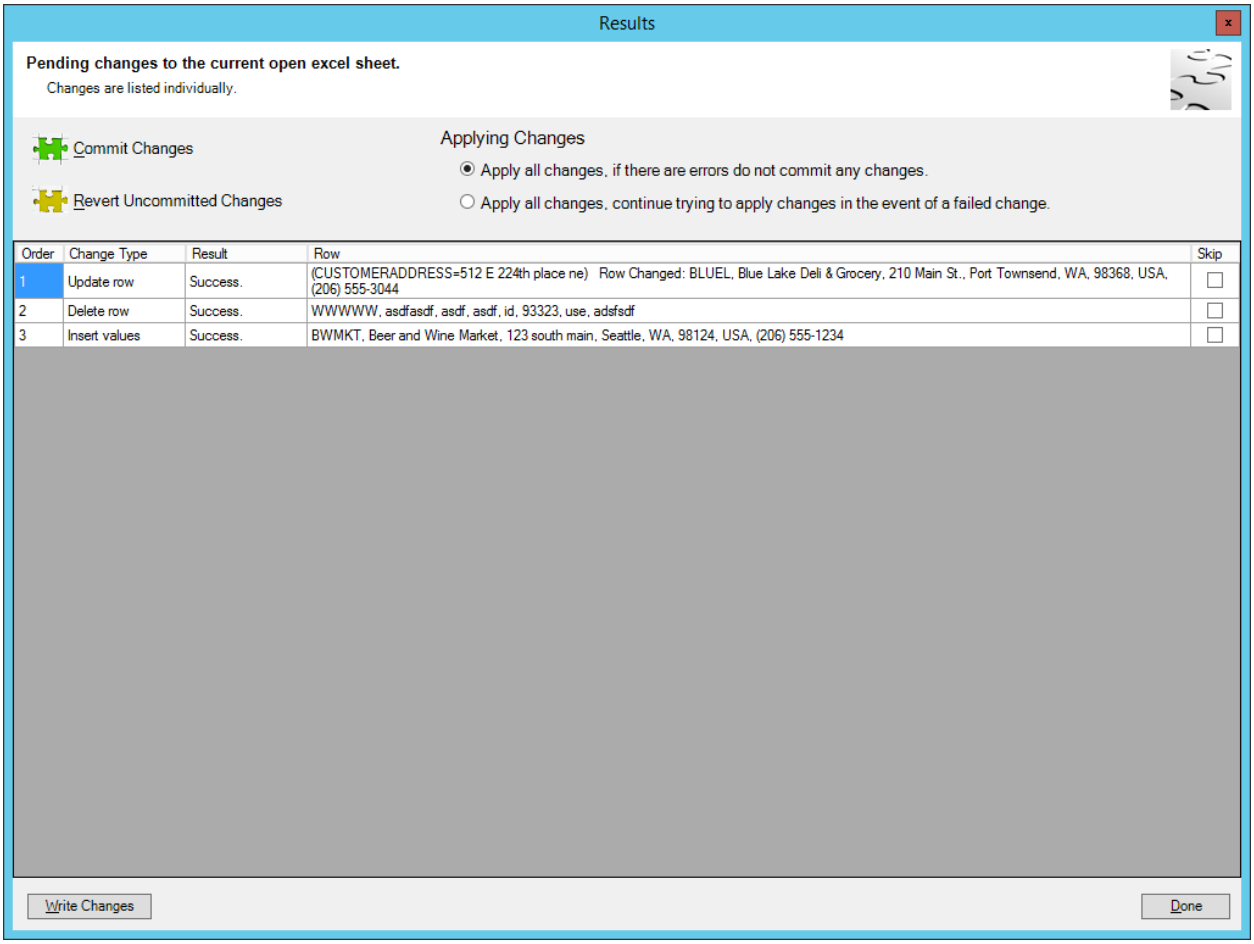
changes even if one fails. Since this is a disconnected data editing tool there is no locks or holds put on the table being edited. It is possible that data is changed in the table from another source and that could cause an error when trying to change data in that row. Also, a quick check is performed to make sure the data going into the table is the correct type of data for that column in the table.



After pressing the "**Commit Changes**" button the grid will show the results of sending the changes to the database. If they succeed it will show Success in the result column, if they fail it will show the error message of why it failed. All the changes that succeeded will be removed from the list of changes after this form is closed.

If there are errors with certain changes and it is desired that all the other changes go to the database without those, they can be skipped by checking the "**Skip**" checkbox on the right. Pressing the "**Revert Uncommitted Changes**" button will get rid of any changes that had errors or were not successfully sent to the database. This is a good step if some of the changes have errors, but all the changes that could succeed did, then it will reset everything and be ready for more changes.

To write the changes to a file at any point before or after sending them to the database, press the "**Write Changes**" button. That will write each row in the display to either a tab delimited or comma separated file. This can be done for auditing purposes or saving an error to be looked into later.


Results
Pending changes to the current open excel sheet.
Changes are listed individually.
Commit Changes
Revert Uncommitted Changes
Applying Changes
Apply all changes, if there are errors do not commit any changes.
Apply all changes, continue trying to apply changes in the event of a failed change.
Order
Change Type
Result
Row
Skip
1
Update row
Success.
(CUSTOMERADDRESS=512 E 224th place ne) Row Changed: BLUEL, Blue Lake Deli & Grocery, 210 Main St., Port Townsend, WA, 98368, USA, (206) 555-3044
2
Delete row
Success.
WWWWW, asdfasdf, asdf, asdf, id, 93323, use, adsfsdf
3
Insert values
Success.
BWMKT, Beer and Wine Market, 123 south main, Seattle, WA, 98124, USA, (206) 555-1234
Write Changes
Done

.

FAQs for CONNX Excel Add In

Question: One of my changes to the table failed when trying to commit to the database. Why can't I just undo that one change? Why do I have to revert changes to clear the bad one?

Answer: It is correct that the only way to eliminate the failing change is to revert changes and get the data from the database. The CONNX Excel Add In tracks each change as it is made, it does not save a before and after value for each cell. So there is no data held to revert the change without refreshing the data from the source table. Solutions were studied to see if this was possible, however due to limitations in the ability to track cell changes and the memory available to store copies of each cells values this was not practical.

Question: Why are large numbers are being rounded before being put into the source table?

Answer: Excel has a maximum of 15 digits of precision, after that it rounds to zero. For example if you put the number 9223372036854775807 when input into a cell, when Excel Add In tracks the change, the number Excel gives to the Add In is 9223372036854700000.

Question: After I open a connection for editing a table on an Excel sheet, in the File -> Info tab of Excel it shows that the sheet is protected. When I press the unprotect link it asks for a password, why?

Answer: When a sheet is being used to edit a database table most of the cells are protected. This is necessary to be able to track changes and stop loss of data. The protection cannot be removed, it is used for editing.

Question: I am editing a table and the last row is "cnxrowid" but when I change the values, why are they not saved?

Answer: The "cnxrowid" is a way for the CONNX to apply a row identifier to a table that has been imported into a data dictionary. This is a virtual column, it does not physically exist on the source table. By checking "Show CNXROWID" in the data dictionary, this field will show up with the table, however it cannot be edited and changes to that column are not allowed.

Question: When I try to use Replace in the Excel Edit features it says that it cannot be used when the sheet is locked, why can't I use the replace function?

Answer: The primary goal of the CONNX Data Task pane is to allow editing of data in a database table. To accomplish this some of the excel functions that are normally available have been eliminated by locking the sheet. For example filtering, sorting, adding or moving columns all would make it impossible to track changes, to disallow these functions the sheet must be locked. The sheet being locked stops the replace function from working.

Troubleshooting the CONNX Excel Add In

1) Open Excel, go to File - Options - Add Ins. In the drop down list, go to disabled items. If the CONNX Add in is there, enable it. If it is not, then in the dropdown go to COM Add Ins. The CONNX Excel Add In should be there, if not, it was not installed. If it is there, but it is unchecked, check it and hit OK, if the CONNX menu item still does not appear, continue troubleshooting with the next steps.

2) Customers using Excel 2007 or 2010 need to use CONNX 12 SP1 build 15160 or higher because it will install the required "Microsoft Visual Studio 2010 Tools for Office Runtime (x64)". If in doubt, look in the installed programs list and make sure the Runtime is there. If the CONNX Excel Add In was installed from the standalone installation with CONNX already installed, the main CONNX installation must be at least build 16063 because prior standalone installs did not have the "Microsoft Visual Studio 2010 Tools for Office Runtime (x64)" installed with it.

3) Does the a machine that the CONNX Excel Add In is installed on have the UAC on? If so, try running Excel as an administrator by setting the advanced properties on the shortcut or from c:\program files {or x86}\Microsoft Office\Office15{or version}\Excel.exe right click to run excel as administrator.

4) In Windows 7 the user must have all the windows and office updates installed for the CONNX Excel Add In to be loaded. All updates must be installed, in testing, some of the updates require the user to turn off the UAC, and if it was not, then the updates did not get installed. Even with the the user in the administrator group with the UAC on did not install all the updates in testing. It was not determined which update exactly made the Excel Add In start working during testing. Again this is for Windows 7 users.

5) If the CONNX Excel Add In is still not working, check the setting in the registry. With the UAC off, open the Registry editor directly and change the LoadBehavior registry setting to 3 (0 is disabled). This might need to be done in both the 64 and 32 bit registries. When Excel tries to load the add in, if there is a problem loading the add it, it will set this entry to 0. If the registry entries stay 3 yet the Add In is not loading, it is likely a UAC issue that it is not getting and modifying the registry. The two registry entries are:

HKLM\Software\Microsoft\Office\Excel\Addins\CONNXExcelAddin

HKLM\Software\Wow6432node\Microsoft\Office\Excel\Addins\CONNXExcelAddin

Summary of SSL Support with CONNX

CONNX Supports TLS 1.2 and below (including SSL) for TCP/IP communications in the following scenarios:

1. TCP/IP Connections from the CONNX Client on Windows or UNIX to the CONNX server on z/OS, MVS, OS/390 and zVSE.
2. TCP/IP Connections from the CONNX Client on Windows or Unix to the CONNX Server on VMS Itanium.
3. TCP/IP Connections from the CONNX Client to DB2 configured with SSL/TLS.
4. TCP/IP Connections from the CONNX Client to the UNIX based CONNX Data Servers.
5. TCP/IP Connections from the CONNX Client to the Enterprise Server Service on Windows.
6. TCP/IP Connections from the pure java JDBC Driver to the CONNX JDBC Server on Windows or Unix.

SSL on the Mainframe

CONNX Supports SSL/TLS connections from the CONNX client to any CONNX server running on the mainframe, including Adabas, IMS, and VSAM.

CONNX SSL support on the mainframe is enabled through AT-TLS (Application Transparent TLS).

AT-TLS is the IBM Recommended method of providing SSL/TLS support for IBM hosted applications. It enables companies to configure and control TLS access for all mainframe applications in a central location. It also ensures that applications keep up with the latest TLS standard.

AT-TLS provides a SSL/TLS secured listening port for the application where all encryption and decryption is performed, using high performance hardware if available. The AT-TLS service then makes a normal socket connection to the application listening on the mainframe. As far as the application is concerned, it is communicating to a client using standard TCP/IP. While all critical communication between the mainframe and the client is secured and encrypted via SSL.

In order to use CONNX with SSL on the mainframe with AT-TLS, the following requirements must be met.

1. The CNXCONNECTBACK setting in CNXPARAM in the CONNX listener on mainframe must be set to 0. This ensures that the initial encrypted socket connection is the only socket connection for a given session.
2. For CONNX servers running on VSE using the Barnard TCP/IP stack, you must upgrade to at least Build 257pre32 or higher of Barnard TCP/IP.

Enabling SSL for CONNX Servers on the Mainframe

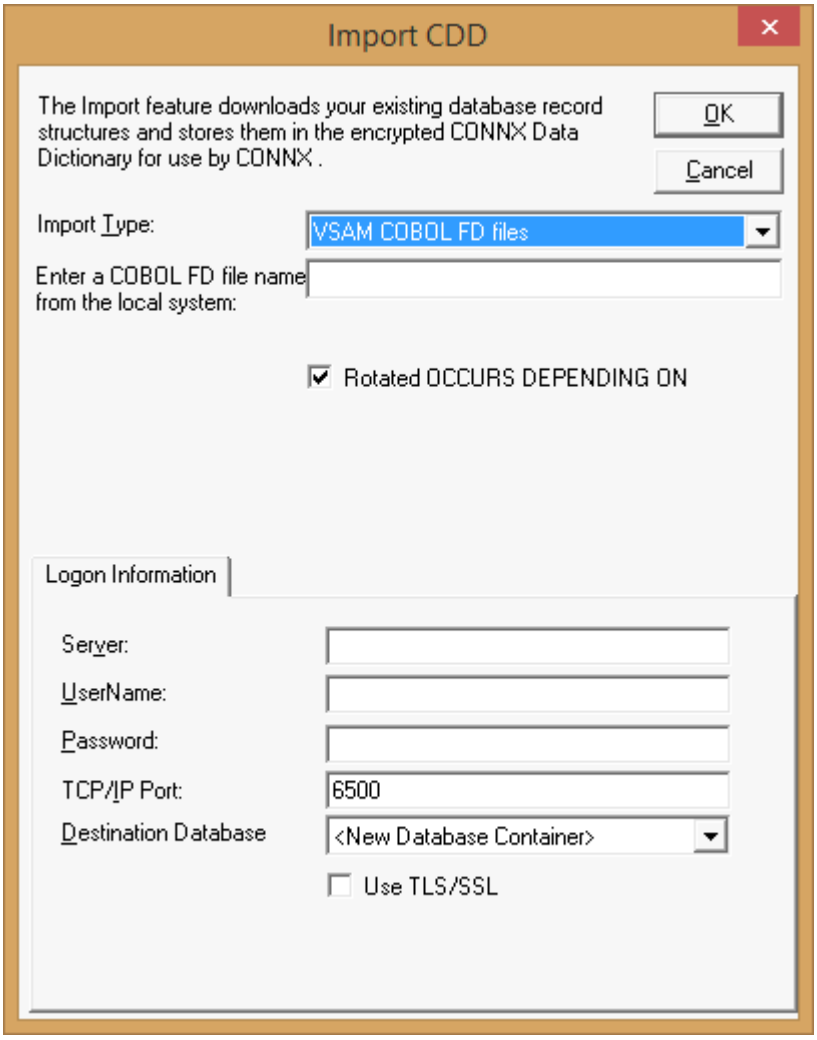
To Enable SSL on the mainframe, configure AT-TLS to provide an SSL listening port that maps to the CONNX listening port.

Please refer to the IBM and/or Barnard documentation on AT-TLS for detailed instructions on service configuration.

On the client side, the CONNX data dictionary must be configured to use SSL to connect to the server.

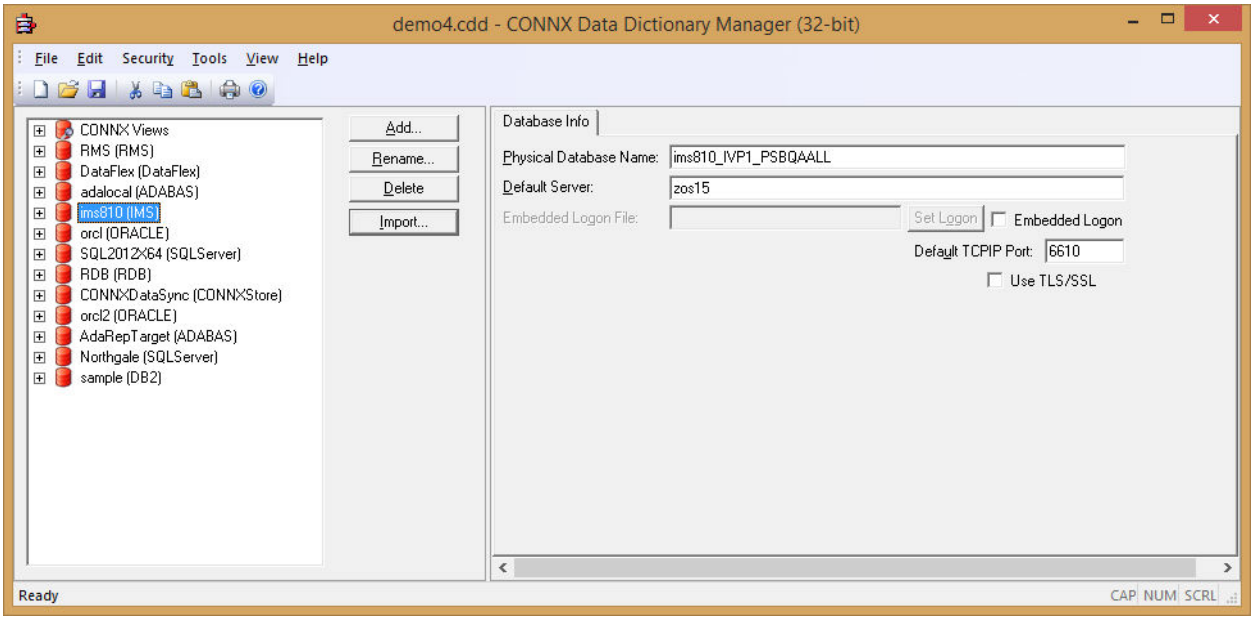
On the import dialog for VSAM, IMS, and Adabas, there is a checkbox "Use TLS/SSL"

Once AT-TLS has been configured, use the AT/TLS SSL port on the import dialog, and select the "Use TLS/SSL" button.



If you have an existing data dictionary, and you want to enable SSL for one or more databases in the CDD, there is also a "Use TLS/SSL" checkbox at the database panel for any database that supports SSL.

Use this checkbox to enable/disable SSL for the specified database connection.



It is possible to configure some databases in the CDD to use SSL, and others without SSL.

SSL on VMS

CONNX Supports SSL/TLS connections from the CONNX client to any CONNX server running on the VMS including RMS, RDB, and Codasyl DBMS.

CONNX provides SSL support on OpenVMS Itanium only.

In order to use CONNX with SSL on OpenVMS, the following requirements must be met.

1. The CNXCONNECTBACK setting on the mainframe must be set to 0. This ensures that the initial encrypted socket connection is the only socket connection.
2. The VMS Server must be Itanium based.
3. A SSL Certificate and Key file are required.

Creating your SSL Certificate and Key file

Use openssl to create your certificate and key file.

CONNX ships with a 64bit OpenSSL command line. It is located in the OpenSSL directory under the root of your 64bit CONNX installation.

Open a command prompt, and navigate to this directory.

Please refer to OpenSSL documentation for details on all of the possible settings when creating keys and certificates. Here is an example of a self-signed key an certificate creation:

```
openssl req -nodes -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -config
openssl.cnf
```

In the above example, key.pem is the private key file, and cert.pem is the certificate file. Both of these files will be required when configuring the SSL Listener.

Enabling SSL for CONNX Servers on OpenVMS

To Enable CONNX SSL support on OpenVMS, the following logical must be present before the CONNX Listener starts:

$define /system CNXLISTENSSL 1

Transfer your SSL Certificate and key to OpenVMS using FTP.

Define two logicals that will point CONNX to the certificate and key called SSLCERT and SSLKEY.

$define /system SSLCERT dka0:[myuser]CERT.PRM

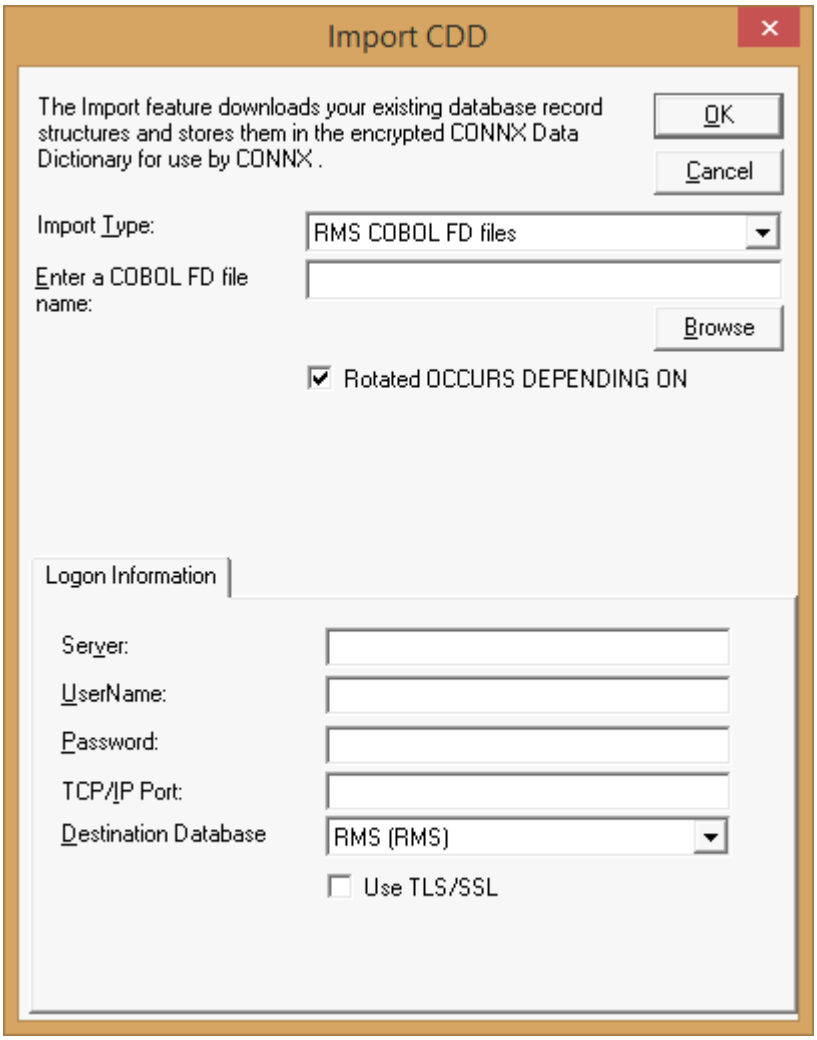
$define /system SSLKEY dka0:[myuser]KEY.PRM

These logicals must be defined at the system level. In order to start some listeners with SSL support and some without, omit defining the CNXLISTENSSL logical at the system level, and instead start the SSL listener(s) with @cnxsvr **startssl**.

Once the logicals are in place the CONNX Listener must be restarted.

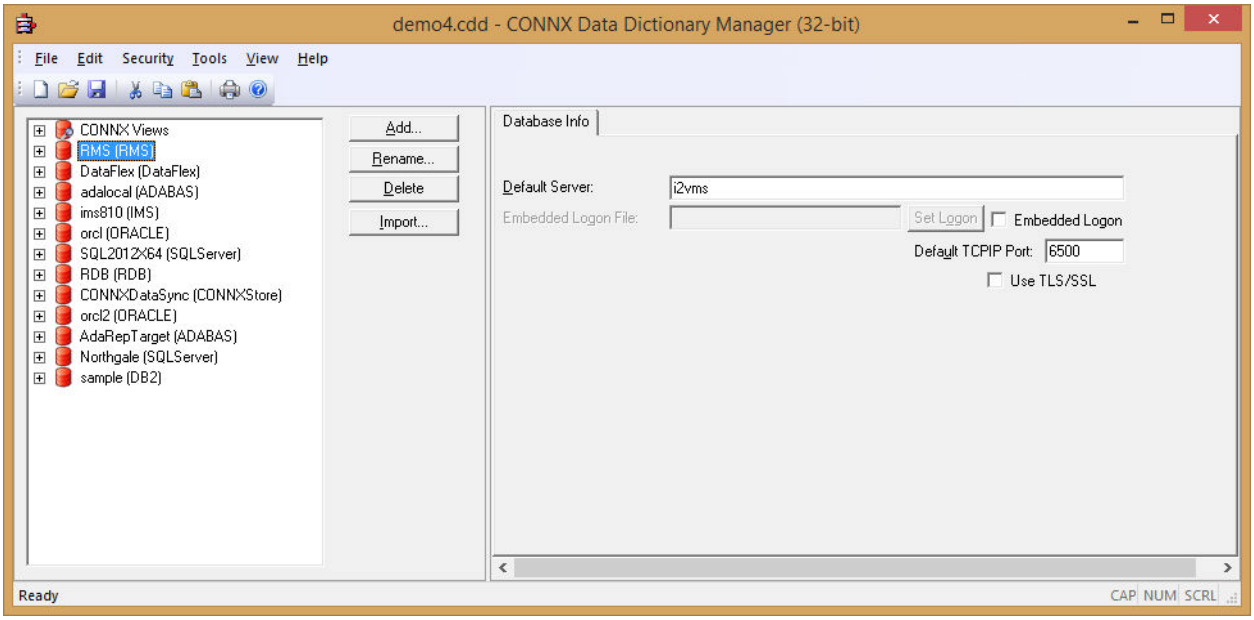
On the client side, the CONNX data dictionary must be configured to use SSL to connect to the server.

On the import dialog for RMS, RDB, and Codasyl DBMS, there is a checkbox "Use TLS/SSL"
Once SSL has been configured for the VMS listener, select the "Use TLS/SSL" button.



If you have an existing data dictionary, and you want to enable SSL for one or more databases in the CDD, there is also a "Use TLS/SSL" checkbox at the database panel for any database that supports SSL.

Use this checkbox to enable/disable SSL for the specified database connection.



It is possible to configure some databases in the CDD to use SSL, and others without SSL.

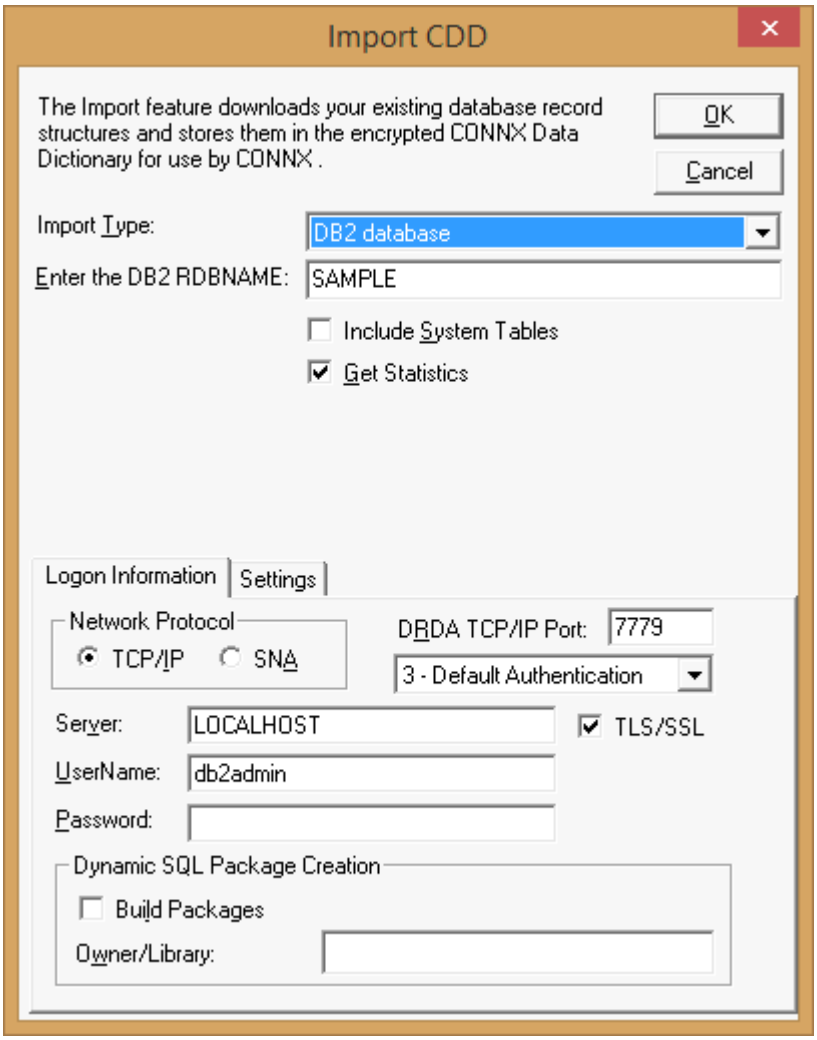
SSL to DB2

CONNX Supports SSL/TLS connections from the CONNX client to DB2 configured with SSL/TLS.
In order to use CONNX with SSL to DB2 the following requirements must be met.

1. The DB2 Database must be configured with SSL support.

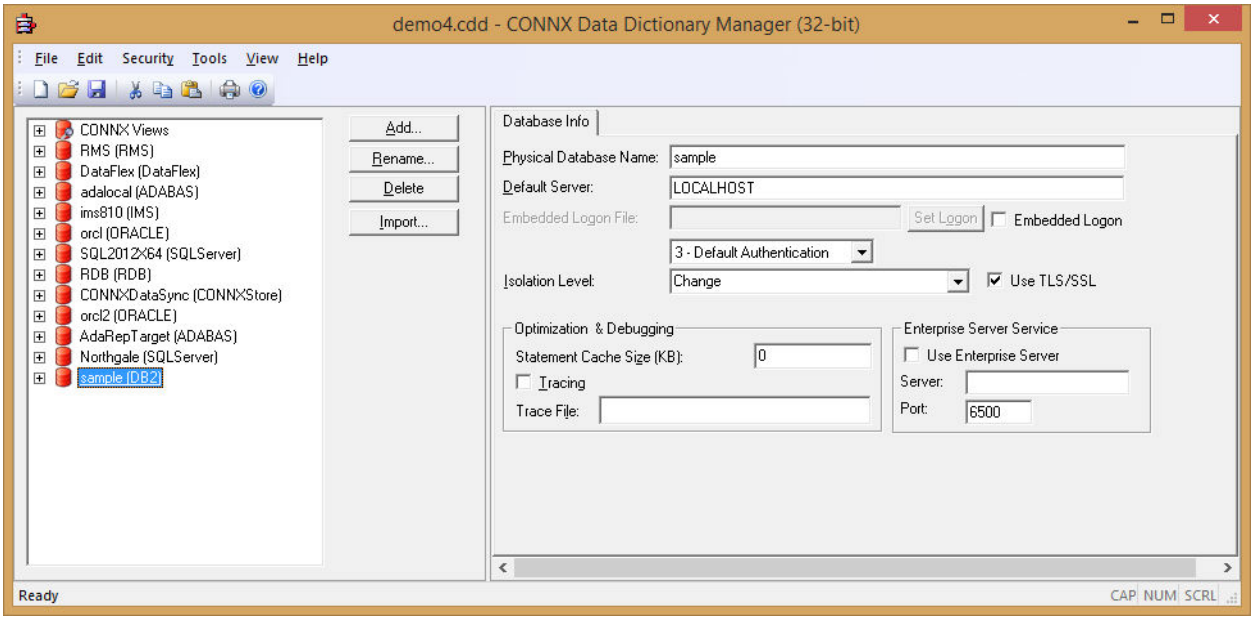
Connecting to SSL enabled DB2

On the import dialog for DB2, there is a checkbox "TLS/SSL"
Once SSL has been configured in DB2, select the "TLS/SSL" button.



If you have an existing data dictionary, and you want to enable SSL for one or more databases in the CDD, there is also a "Use TLS/SSL" checkbox at the database panel for any database that supports SSL.

Use this checkbox to enable/disable SSL for the specified database connection.



It is possible to configure some databases in the CDD to use SSL, and others without SSL.

SSL to the UNIX based CONNX Servers

CONNX Supports SSL/TLS connections from the CONNX client to the 32bit and 64bit CONNX Data Servers running on all flavors of UNIX.

In order to use CONNX with SSL when connecting to UNIX, the following requirements must be met.

1. The UNIX based CONNX Data Server must be configured to listen with SSL.
2. A SSL Certificate and Key file are required.

Creating your SSL Certificate and Key file

Use openssl to create your certificate and key file.

CONNX ships with a 64bit OpenSSL command line. It is located in the OpenSSL directory under the root of your 64bit CONNX installation.

Open a command prompt, and navigate to this directory.

Please refer to OpenSSL documentation for details on all of the possible settings when creating keys and certificates. Here is an example of a self-signed key an certificate creation:

```
openssl req -nodes -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -config
openssl.cnf
```

In the above example, key.pem is the private key file, and cert.pem is the certificate file. Both of these files will be required when configuring the SSL Listener.

Enabling SSL for the UNIX based CONNX Data Server

To Enable CONNX SSL for UNIX, the following CONNX configuration settings must be added to the connxserver script on UNIX, bolded below:

CNXLISTENSSL must be set to 1

Transfer your SSL Certificate and key to the UNIX system.

Define two configuration settings that will point CONNX to the certificate and key called SSLCERT and SSLKEY.

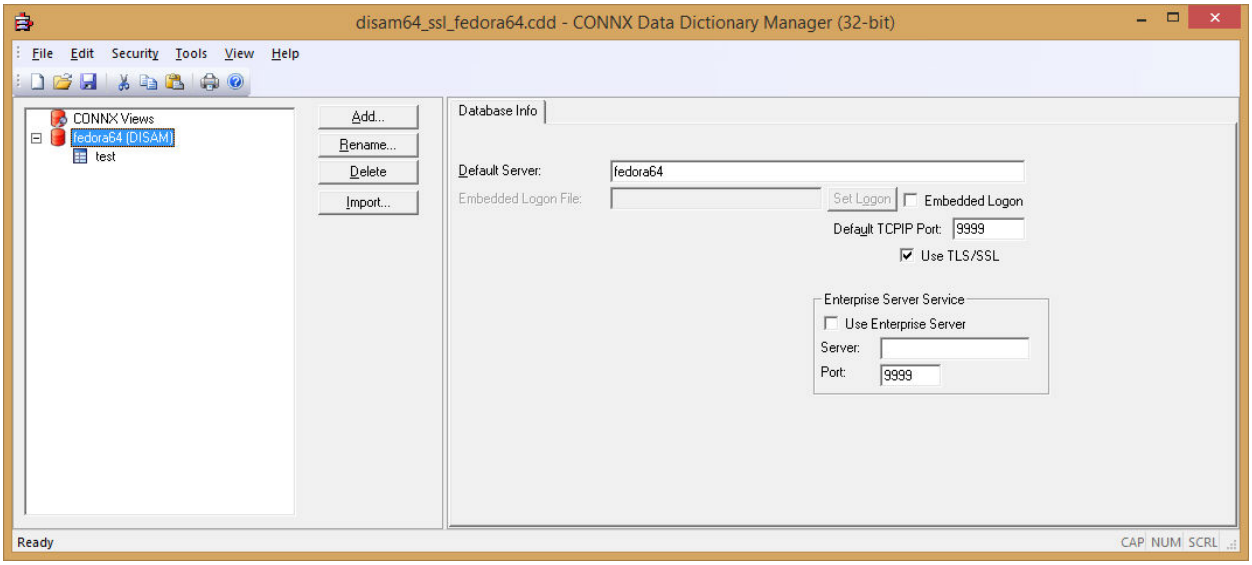
```
##############################################################################
####################
# ENVIRONMENT VARS
#
# CNXDIR               = Location of this shell script + the CONNX binaries
# CNXRUNPORT           = Default port (default=6500)
# CNXCONNECTBACK       = Enables / Disables TCP/IP socket sharing
# DEBUG                = 1 ON | 0 OFF
#
#
# See the documentation for complete information on ENVIRONMENT VARIABLES
```

```
#
##############################################################################
####################
CNXDIR=/home/cnxuser/connx
CNXRUNPORT=6500
CNXCONNECTBACK=0
DEBUG=1
CNXLISTENSSL=1
SSLCERT=/home/cnxuser/server.crt
SSLKEY=/home/cnxuser/server.key
export CNXDIR CNXRUNPORT CNXCONNECTBACK DEBUG CNXLISTENSSL SSLCERT SSLKEY
##############################################################################
####################
# DO NOT EDIT THE FOLLOWING SECTION
##############################################################################
####################
.....
```

Once these setting changes are made, the UNIX based CONNX Data Server must be restarted.

On the client side, the CONNX data dictionary must be configured to use SSL to connect to the server.

If you have an existing data dictionary, and you want to enable SSL for one or more databases in the CDD, there is also a "Use TLS/SSL" checkbox at the database panel for any database that supports SSL. Use this checkbox to enable/disable SSL for the specified database connection.



It is possible to configure some databases in the CDD to use SSL, and others without SSL.

SSL to the CONNX Enterprise Server Service

CONNX Supports SSL/TLS connections from the CONNX client to the 32bit and 64bit Enterprise Server Service (ESS) running on Windows.

In order to use CONNX with SSL when connecting to ESS, the following requirements must be met.

1. The ESS service must be configured to listen with SSL.
2. A SSL Certificate and Key file are required.

Creating your SSL Certificate and Key file

Use openssl to create your certificate and key file.

CONNX ships with a 64bit OpenSSL command line. It is located in the OpenSSL directory under the root of your 64bit CONNX installation.

Open a command prompt, and navigate to this directory.

Please refer to OpenSSL documentation for details on all of the possible settings when creating keys and certificates. Here is an example of a self-signed key an certificate creation:

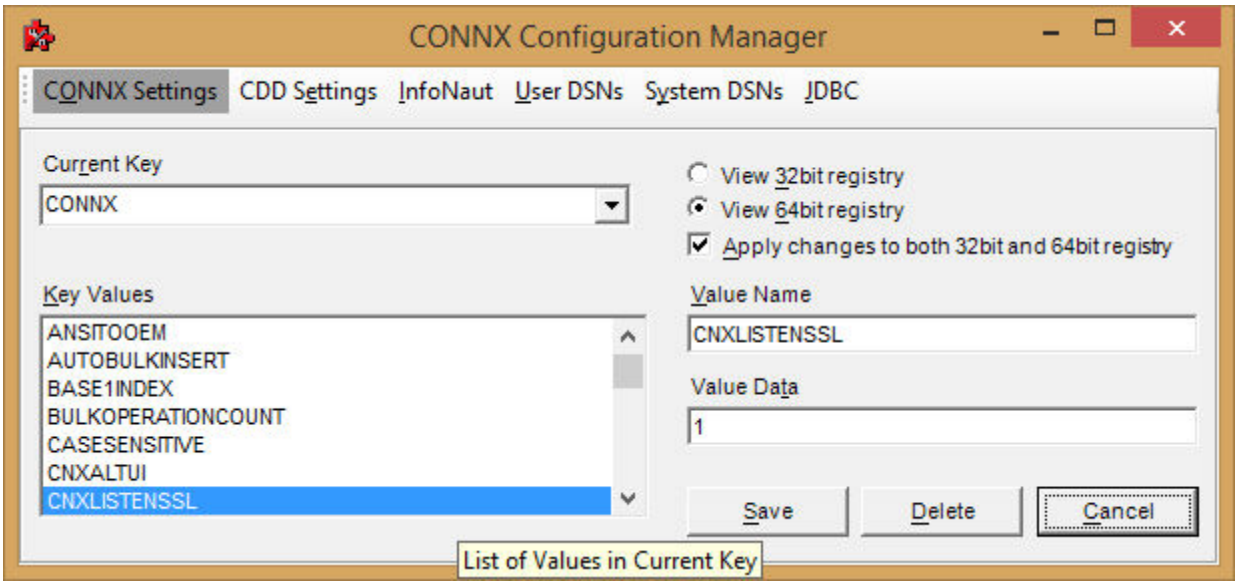
```
openssl req -nodes -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -config
openssl.cnf
```

In the above example, key.pem is the private key file, and cert.pem is the certificate file. Both of these files will be required when configuring the SSL Listener.

Enabling SSL for the CONNX Enterprise Server Service

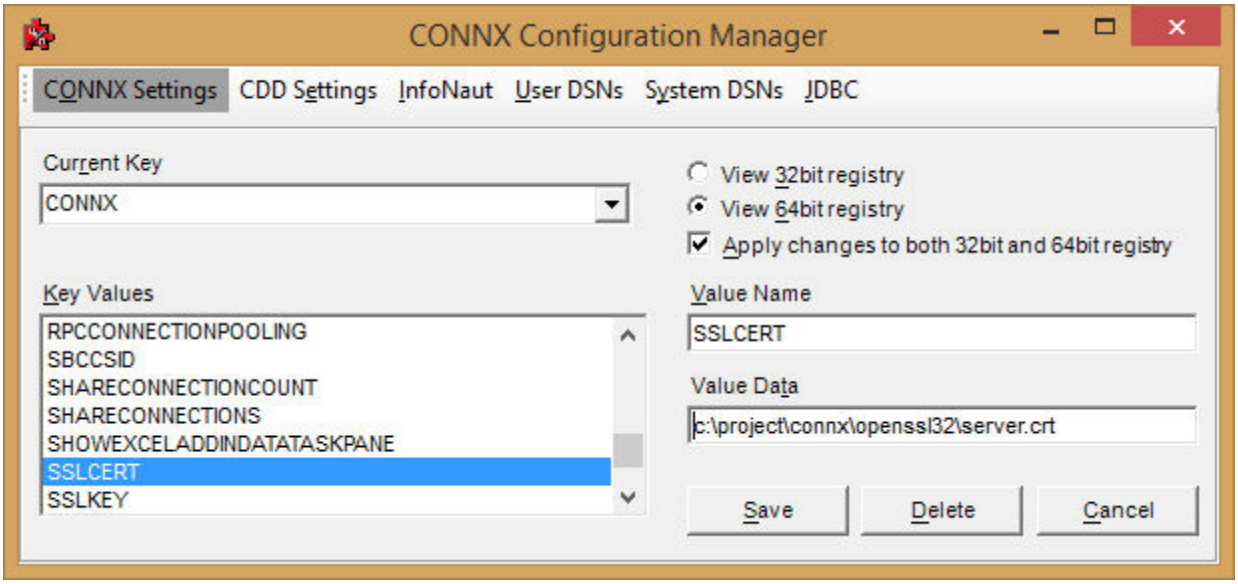
To Enable CONNX SSL for ESS, the following CONNX configuration setting must be set:

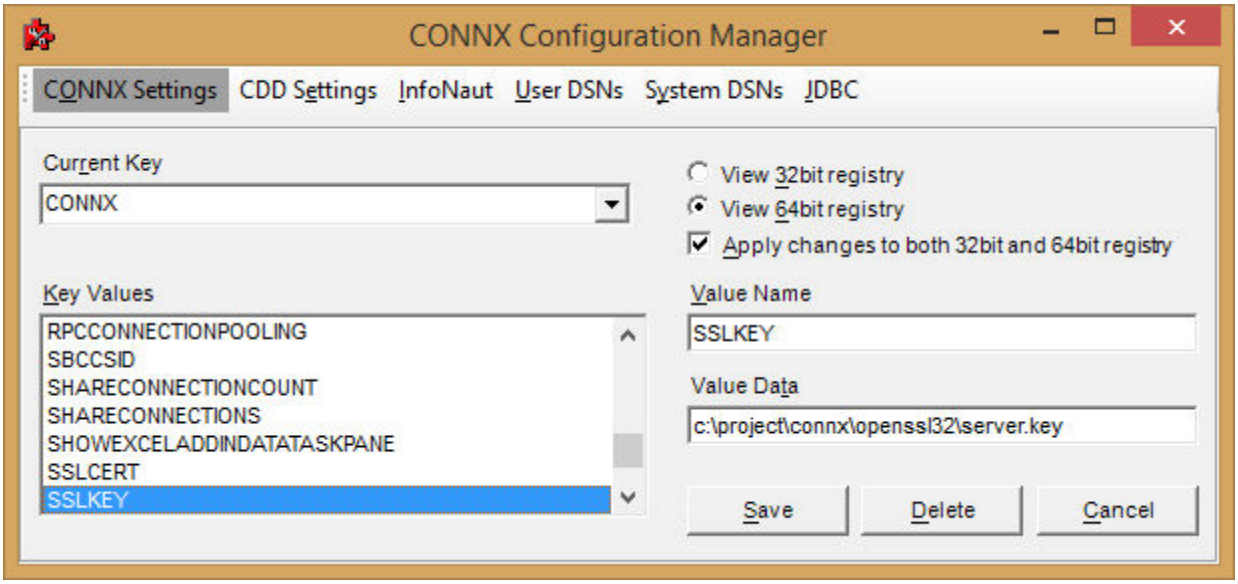
CONNX.CNXLISTENSSL must be set to 1



Transfer your SSL Certificate and key to the system running ESS.

Define two configuration settings that will point CONNX to the certificate and key called CONNX.SSLCERT and CONNX.SSLKEY.



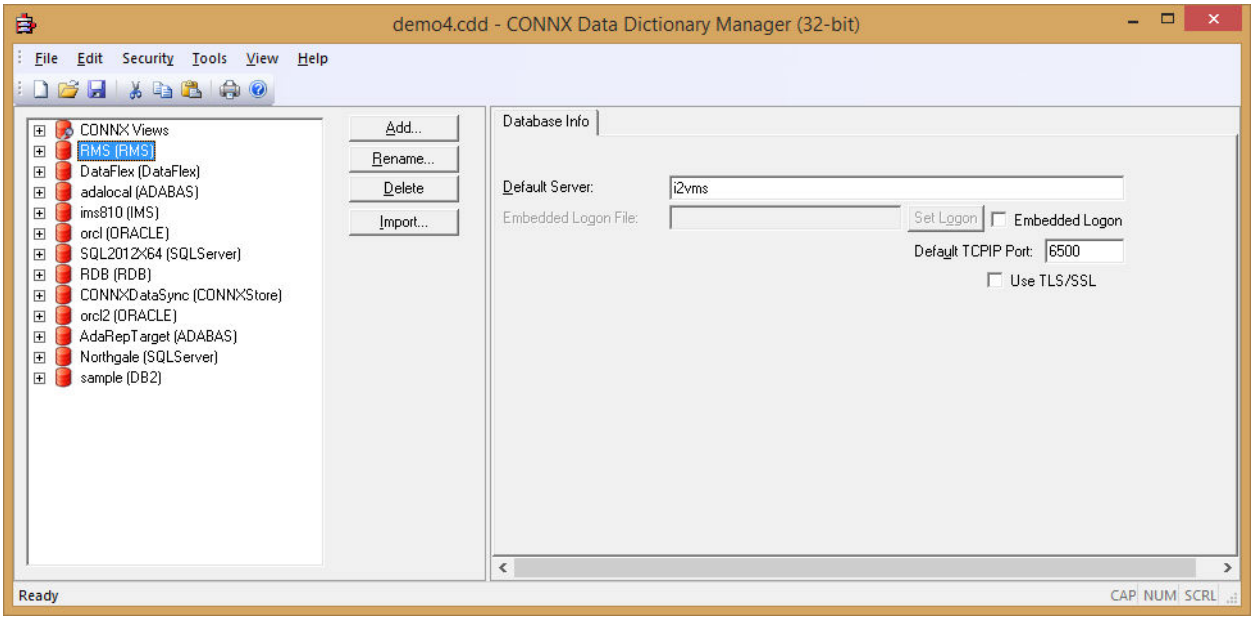


Once these setting changes are made, the EES Service must be restarted.

On the client side, the CONNX data dictionary must be configured to use SSL to connect to the server.

If you have an existing data dictionary, and you want to enable SSL for one or more databases in the CDD, there is also a "Use TLS/SSL" checkbox at the database panel for any database that supports SSL.

Use this checkbox to enable/disable SSL for the specified database connection.



It is possible to configure some databases in the CDD to use SSL, and others without SSL.

SSL to the CONNX JDBC Server Service

CONNX Supports SSL/TLS connections from the pure java JDBC Driver to the 32bit and 64bit JDBC Server service running on Windows or Unix.

In order to use CONNX with SSL when connecting with JDBC, the following requirements must be met.

1. The CONNX JDBC Server service must be configured to listen with SSL.
2. A SSL Certificate and Key file are required.

Creating your SSL Certificate and Key file

Use openssl to create your certificate and key file.

CONNX ships with a 64bit OpenSSL command line. It is located in the OpenSSL directory under the root of your 64bit CONNX installation.

Open a command prompt, and navigate to this directory.

Please refer to OpenSSL documentation for details on all of the possible settings when creating keys and certificates. Here is an example of a self-signed key an certificate creation:

```
openssl req -nodes -x509 -newkey rsa:4096 -keyout key.pem -out cert.pem -days 365 -config
openssl.cnf
```

In the above example, key.pem is the private key file, and cert.pem is the certificate file. Both of these files will be required when configuring the SSL Listener.

Importing the certificate into the Java security store

The certificated used in the JDBC server must be imported into the Java security store.

First, identify which Java jdk/sdk the application uses. Then, locate the "cacerts" security file under that directory structure.

Then use the keytool utility to import the certificate. The default password for a java security store is "changeit" - you will be prompted

for the password of the security store.

Here is an example of how to import a certificate into the java security store:

```
c:\Program Files (x86)\Java\jdk1.7.0_17>keytool -import -alias example -keystore  "C:\Program Files
(x86)\Java\jdk1.7.0_17\jre\lib\security\cacerts" -file c:\project\connx\OpenSSL32\server.crt
```

Enabling support for large cryptographic key lengths in Java

If you are not using the latest version of Java, you may be using a version that has a default policy of a limited cryptographic key length.

If you are not using one of the Java versions listed below, you will need to download and install the "Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy" for your version of Java.

As an example, for Java 8, it is located here:
http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html

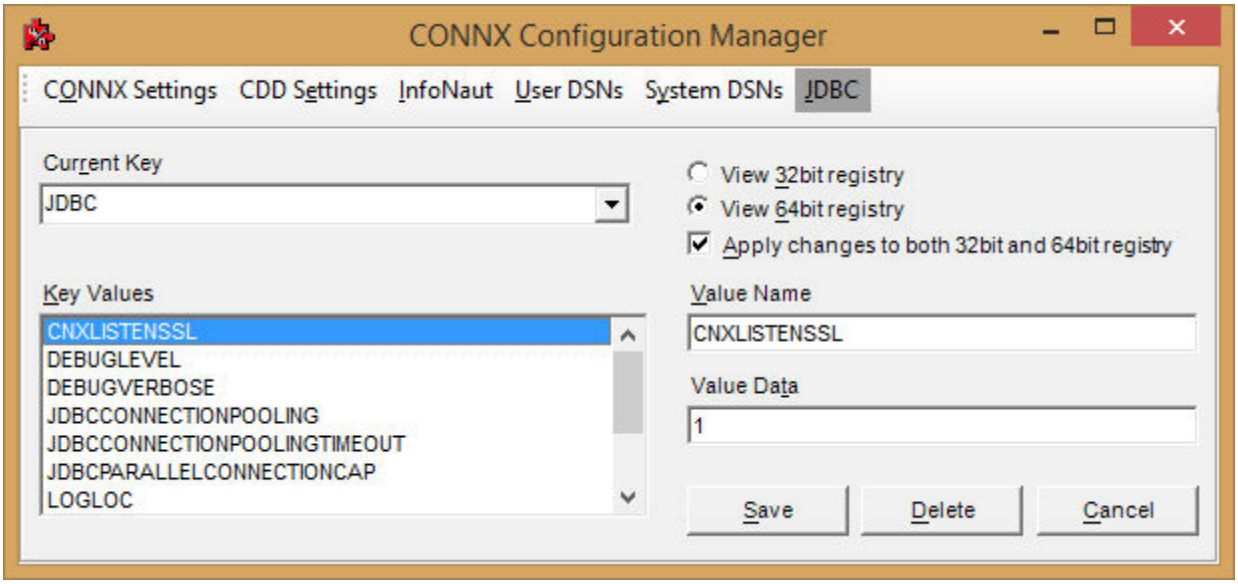
Versions of Java that have a default policy of unlimited cryptographic key length (and do not require any changes):

- Java 9 and above
- Java 8u161 or later
- Java 7u171 or later
- Java 6u181 or later

Enabling SSL for the CONNX JDBC Server Service

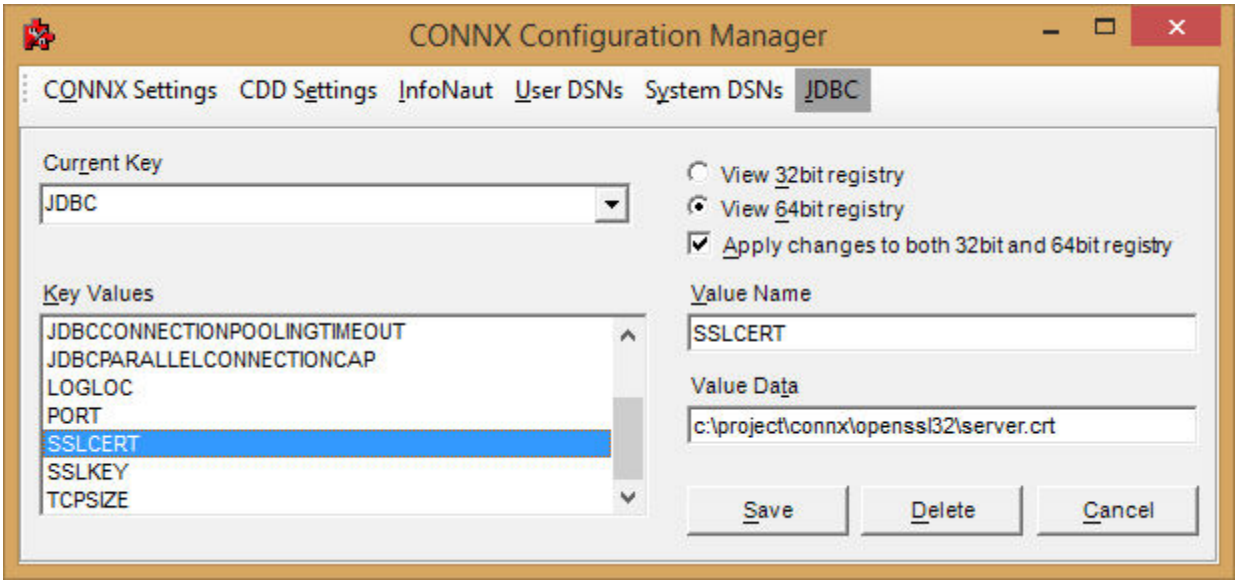
To Enable CONNX SSL for ESS, the following CONNX configuration setting must be set:

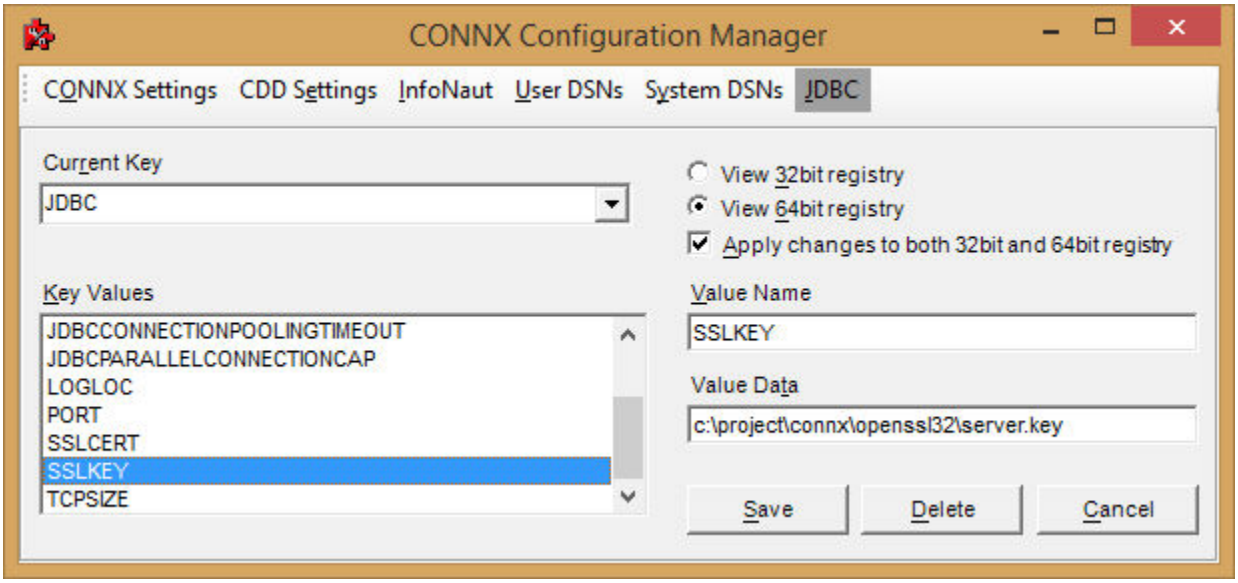
CONNX.JDBC.CNXLISTENSSL must be set to 1



Transfer your SSL Certificate and key to system running the CONNX JDBC Servers.

Define two configuration settings that will point CONNX to the certificate and key called CONNX.JDBC.SSLCERT and CONNX.JDBC.SSLKEY.





Once these setting changes are made, the JDBC server must be restarted.

On the client (the Java application), a new setting must be added to the connection string to enable SSL connections to the JDBC Server.

The setting name is SSL, and it should be set to a value of true.

Here is an example connection string with SSL enabled:
jdbc:connx:DD=oracle;Gateway=localhost;Port=7500;**ssl=true**

CONNX JDBC Sample Application

The CONNX JDBC Sample Application demonstrates CONNX JDBC in both a Windows and non-Windows environment. The sample application enables the user to formulate one- or two-parameter queries or to enter standard SQL statements directly from the keyboard.

With queries that use parameters, the application demonstrates data type conversion, which enables the user to cast a column of data from its native data type to a more convenient date type. For example, a Web page designer might want to retrieve all data as String, even though the native data types might include Integer, Float, etc.

Related Topics

The CONNX JDBC Driver

JDBC Driver Definition

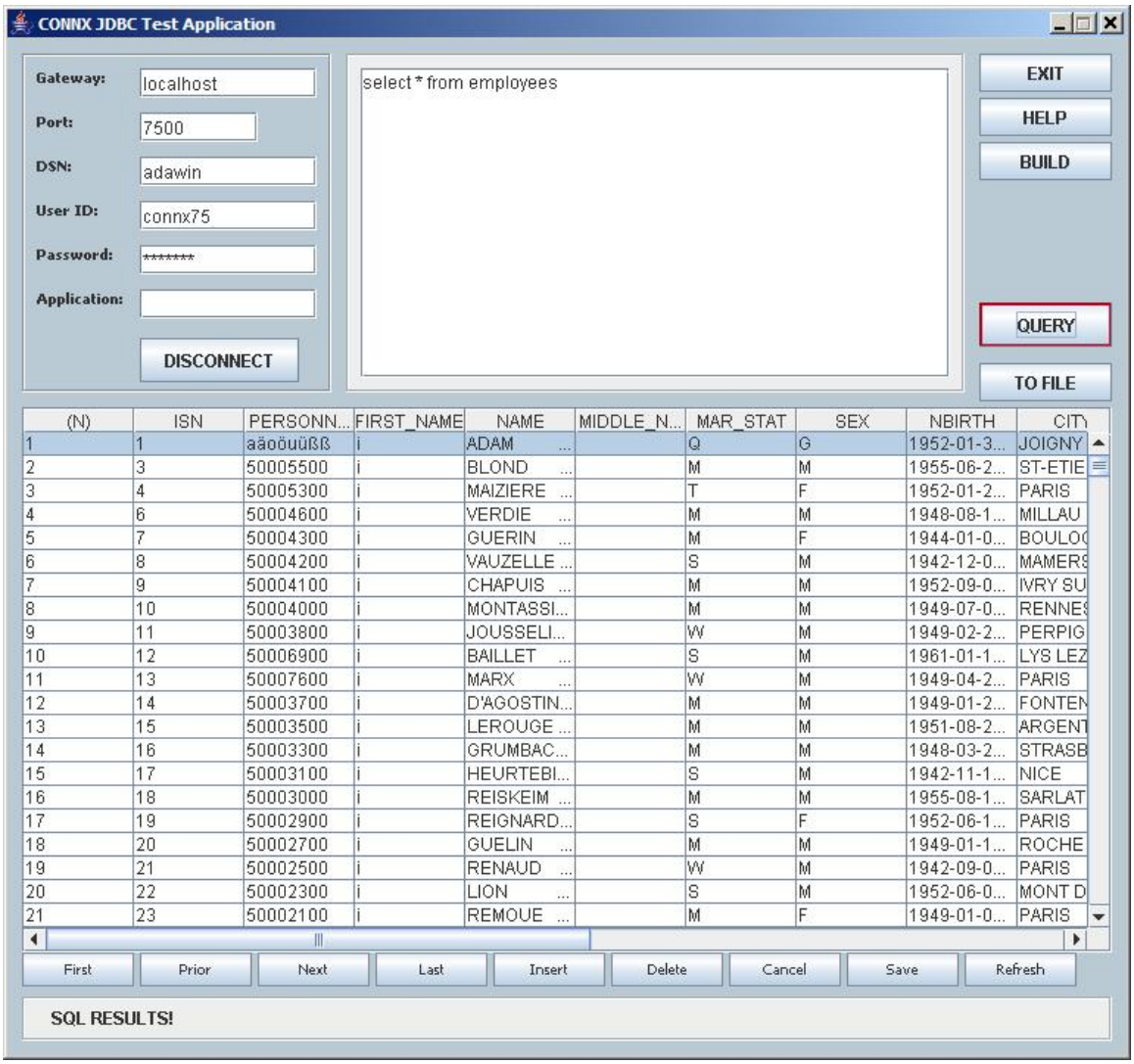
JDBC Driver Architecture

Using the CONNX JDBC Sample Application in a Microsoft Windows Environment

The following procedure describes how to start the CONNX JDBC Sample Application in a Microsoft Windows environment.

To start the CONNX JDBC Sample Application in a Windows environment

1. Click the **Start** button, and then point to **Programs.** Point to **CONNX Solutions,** point to **CONNX,** and then click **CONNX JDBC Sample Application.** The Query Test Application window appears.



Using the CONNX JDBC Sample Application in a non-Windows environment

The following procedure describes how to start the CONNX JDBC Sample Application in an environment other than Microsoft Windows.

To start the CONNX JDBC Sample Application in a non-Windows environment

1. Verify that the connxjdbcftp.jar file has been copied and unpacked to the non-Windows target. For information on how to transfer the connxjdbcftp.jar file, refer to CONNX JDBC in the CONNX Installation Guide.
2. Go to the <installation location>\CONNXJDBC\JAVA\SAMPLES directory. For example, in a Unix environment, type the following on the command line:

   ```
   cd java/samples
   ```

3. Press **<Enter>,** and then type the command

   ```
   sh sapplic_unix
   ```

   The Query Test Application window appears.
4. Follow the instructions on the following pages describing how to log on and run a query in the CONNX JDBC Sample Application.

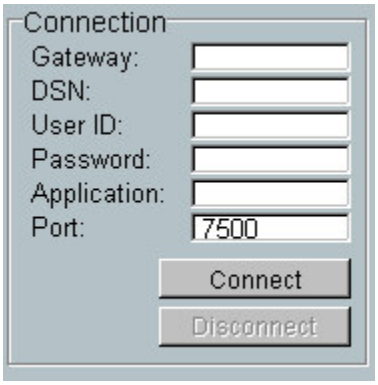
Running the CONNX JDBC Sample Application

The procedures listed below describe how to connect to a data source, choose an output type, select a query type, and run three different query types.

To connect to a data source
Navigator Bar

To connect to the data source

1. Type the following information in the **Connection** group box in the Query Test Application window:



- **Gateway:** The local server TCP/IP address or host name (for example, 127.0.0.1).
- **DSN (Data Source Name):** The data source name as registered with the DSNRegistry tool on the CONNX JDBC Server machine. When CONNX is installed, the CDD DSN connxSamples is automatically created and registered to enable access to the CONNX8Sample database. The DSNRegistry tool is described in To add a new data source name for the JDBC Driver.
- **User ID:** CONNX user name (for example, NickD).
- **Password:** CONNX password (appears as ********).
- **Application:** Database connection option (for example, Oracle or RMS). Database connection options are described in CONNX System Requirements. The Application text box may be left blank if all servers are running that contain the databases listed in the CONNX Data Dictionary. If any of the servers are down, or if the server running the application specified in the Application text box is down, the connection times out and the following message appears in the SQL Statement from Keyboard text box:

```
Unable to open database.

>>> STAT:2 ERR:-12505 SQLSTATE:HYT01 ODBC State = HYT01: Connection timeout
expired; TCP/IP Error = 10060: The connect request timed out.
```

1. **Port:** The port number (default is 7500).

2. Click the **Connect** button in the **Connection** group box. The following message appears in the **SQL Statement from Keyboard** group box:

   ```
   Attempting to connect
   ```

3. After the connection to the data source is established, the **Disconnect** button in the **Connection** group box and the **Query** and **Clear Query** buttons in the **SQL Statement from Keyboard** group box are activated. The following message appears in the **Miscellaneous Status Bar:**

```
Connection successful-please select an output type and a query type
```

Navigator Bar

Use the Navigator bar to examine or modify the data set. The data set represents a live link to the original data. Important: Any saved changes to the data set are saved to the original data source if the user has update privileges.

The Navigator bar contains the following keys:

- **First:** Moves the cursor to the first row of the data set.
- **Prior:** Moves the cursor to the previous row of the data set.
- **Next:** Moves the cursor to the next row of the data set. If focus is on the last row when the button is clicked, a new (blank) row opens.
- **Last:** Moves the cursor to the last row of the data set.
- **Insert:** Opens a new (blank) row to allow data entry (must be Saved to be made permanent). Produces an error if user does not have update privileges.
- **Delete:** Deletes the selected row (must be Saved to be made permanent). Produces an error if user does not have update privileges.
- **Cancel:** Cancels modifications made to the selected row of data. The data is restored to the way it was before the edit began or to the state it was since the last Save.
- **Save:** Saves all changes made since the last Save to the data source. This is a permanent change that affects the original data source. Produces an error if user does not have update privileges.
- **Refresh:** Refreshes the data set from the original data source.

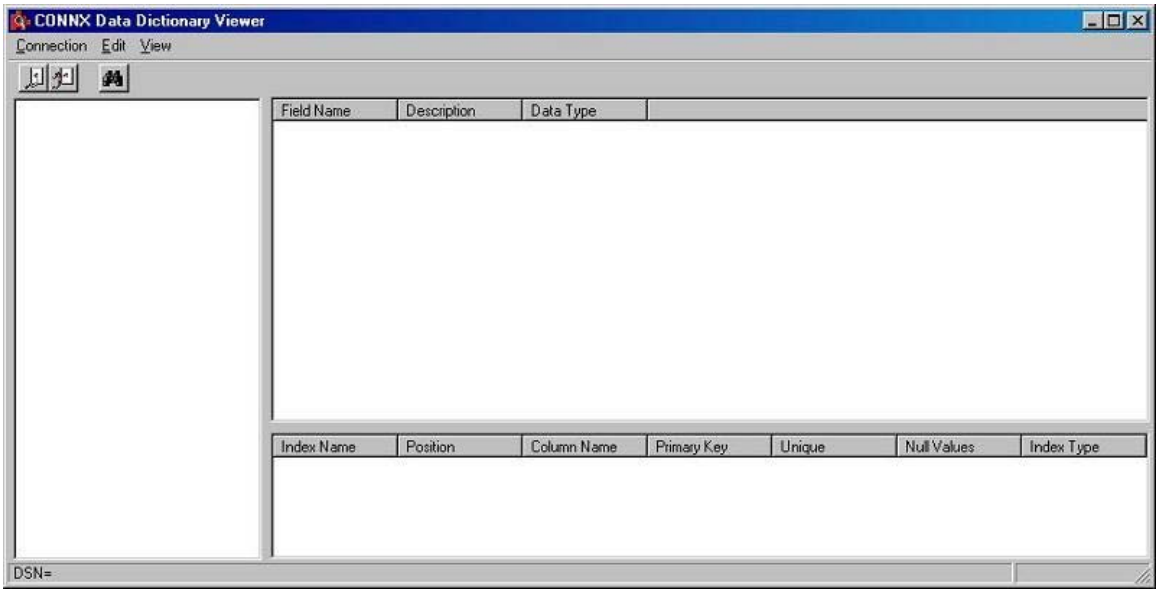
CONNX Data Dictionary Viewer

Introduction to the CONNX Data Dictionary Viewer

The CONNX Data Dictionary Viewer tool is included with CONNX for the benefit of users working on CONNX client machines who do not have access to the CONNX Administrator but who would need to search for and locate specific fields in tables in a CONNX Data Dictionary.

To establish a connection to the CONNX Data Dictionary Viewer

1. Click the **Start** button, and then point to **All Programs.** Point to **CONNX Driver** and then click **Data Dictionary Viewer Utility.** The CONNX Data Dictionary Viewer appears.

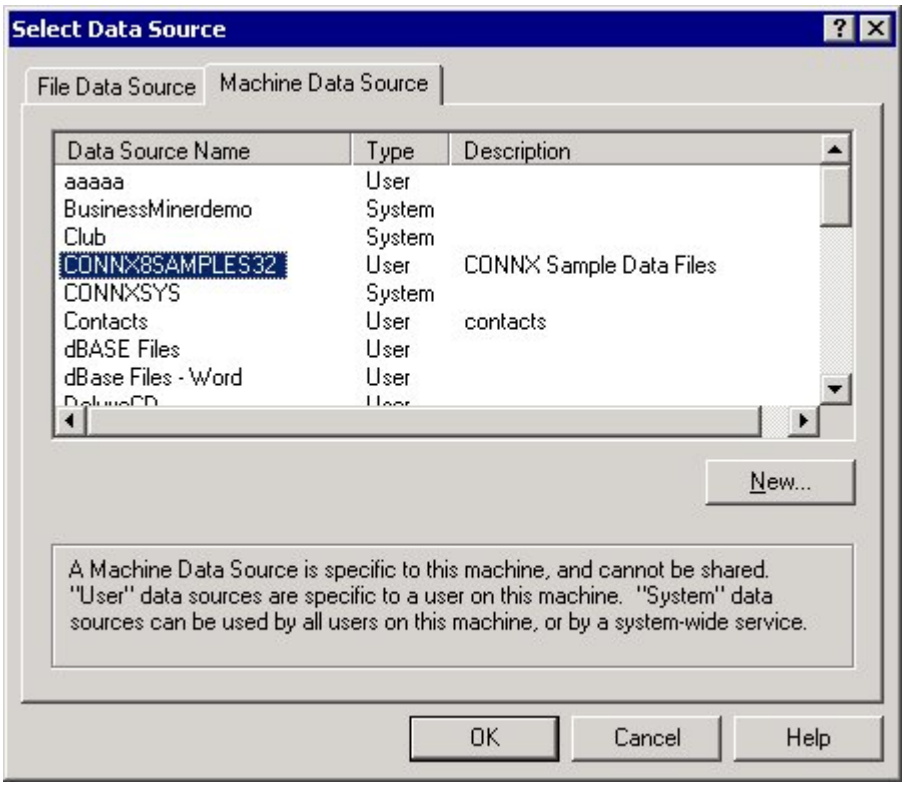


2. Table names are displayed in the left pane, their related fields in the top right pane, and their related indexes in the lower right pane. The index pane displays the primary index, unique values (if any), and the fields and order of the fields within the index.

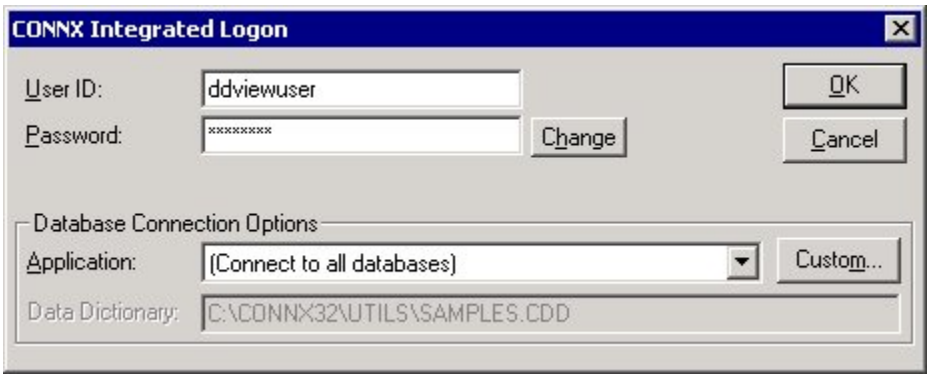
To open a connection to a data source name

1. On the **Connection** menu, click **New Connection.** You can also click the Connection button.

2. The **Select Data Source** dialog box appears.
3. Click the **Machine Data Source** tab, and then select a data source from the list.

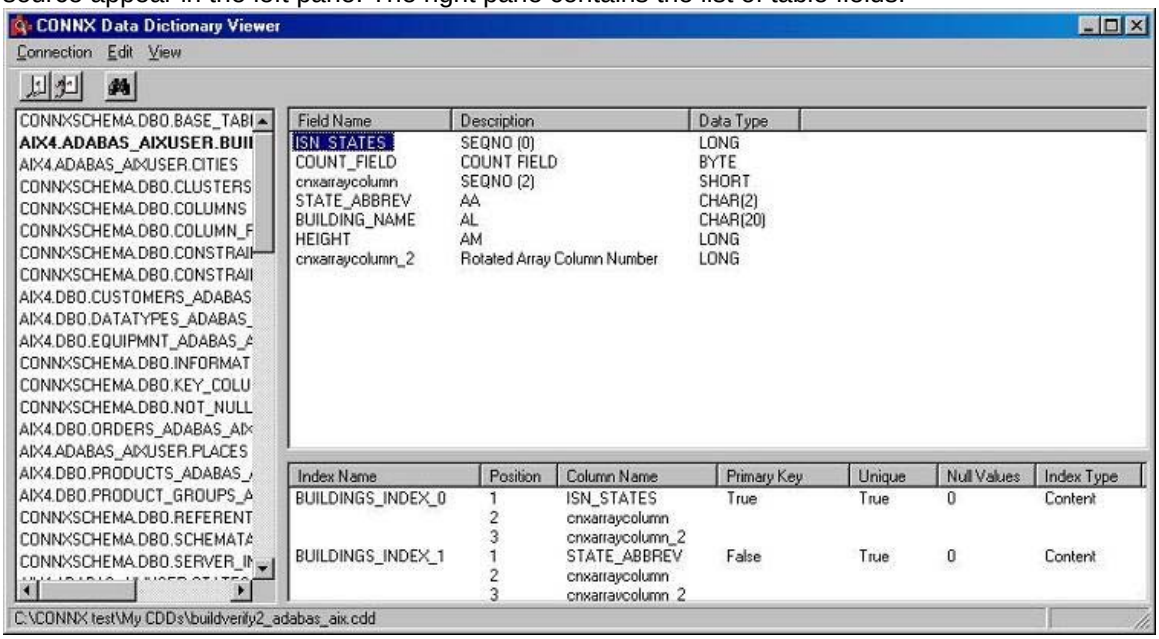


4. Click the **OK** button. The **CONNX Integrated Logon** dialog box appears.

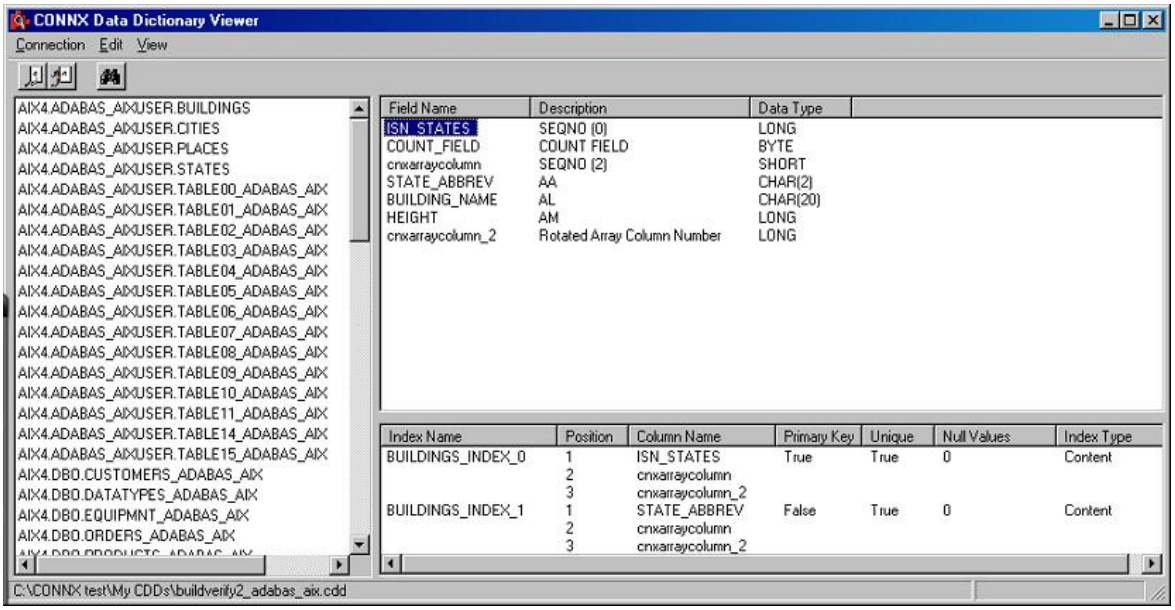


5. Type a user name and password for CONNX, and then click the **OK** button.

6. The **CONNX Data Dictionary Viewer** appears. The list of tables available in the selected data source appear in the left pane. The right pane contains the list of table fields.



7. The tables are sorted by table name. For an alternate view, on the Tools menu click Sort by Adabas File ID. The left pane will contain the same list of tables, now in Adabas File ID order.



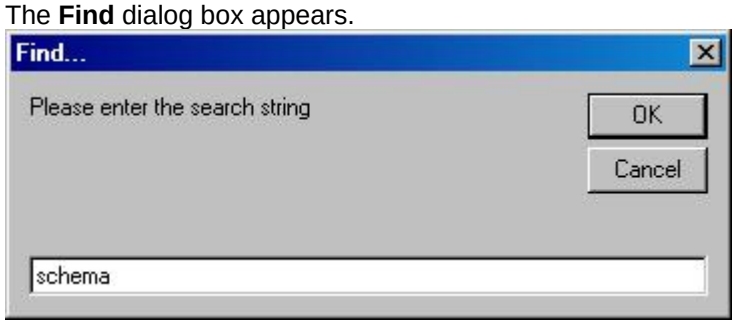
To locate a specific field or table

To locate a specific field:

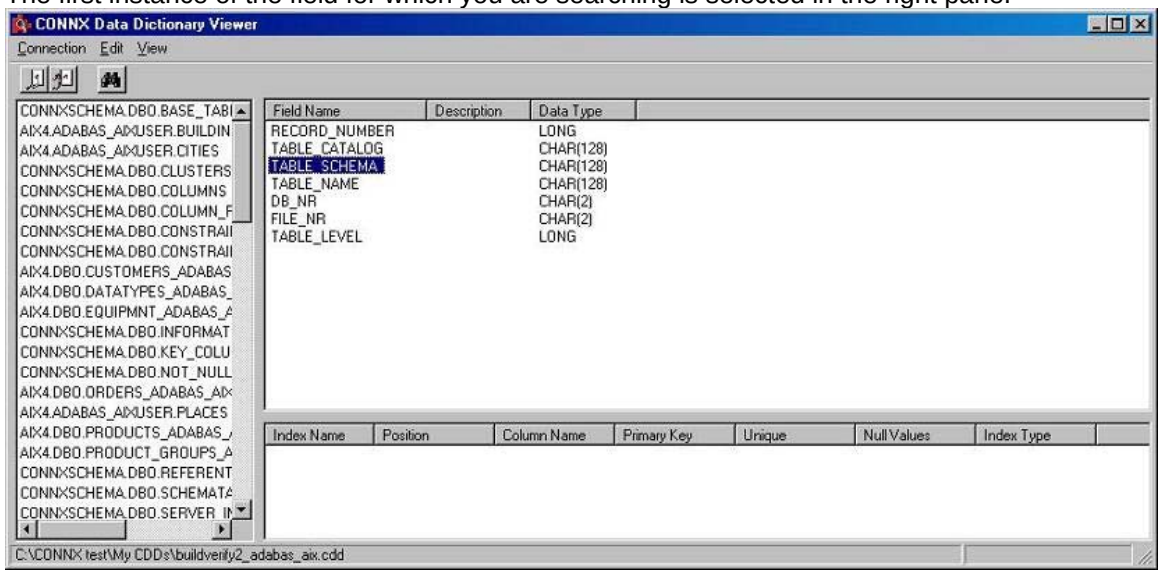
1. On the **Edit** menu, click **Find.** You can also click the **Find** button to open the **Find** dialog box.

Find button

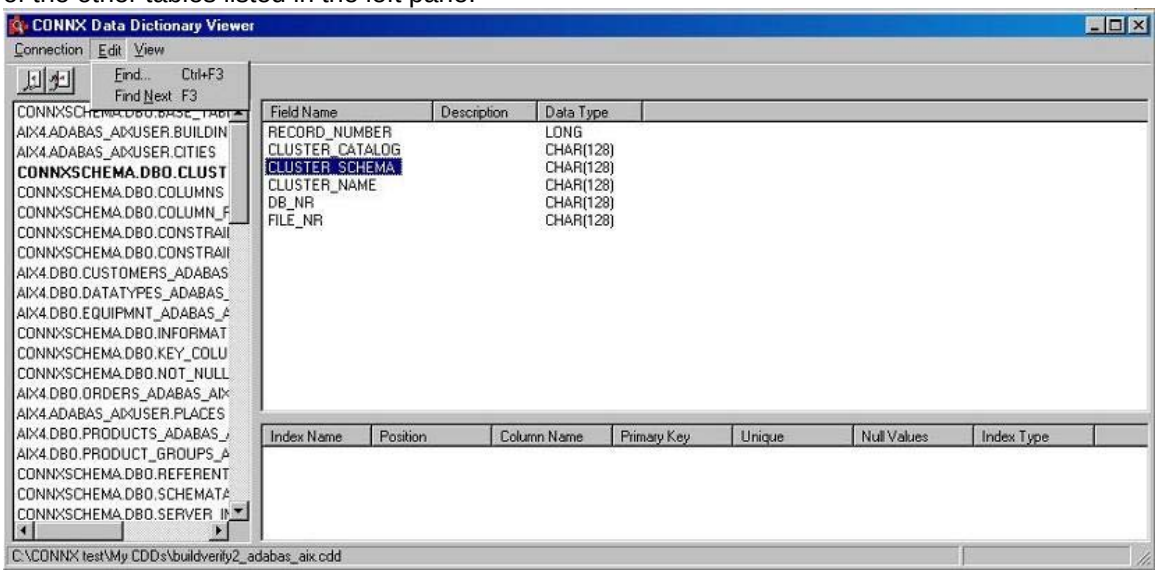
2. The **Find** dialog box appears.



3. Type the name of the field for which you are searching, and then click **OK**.
4. The first instance of the field for which you are searching is selected in the right pane.



5. On the **Edit** menu, click the **Find Next** button to continue searching for the same field name in all of the other tables listed in the left pane.



To locate a specific table:

To close a database connection

1. On the **Connection** menu, click **Close Connection.** You can also click the **Close Connection** button.
   
   Close Connection button
2. The data source connection closes.

Microsoft Access and Visual Basic Performance Tips

Both Microsoft Access and Microsoft Visual Basic support a rich feature set of SQL operations through the Microsoft® Jet Engine. Use of this robust set of capabilities without some background knowledge may yield less-than-satisfactory results. The following tips will help improve performance of both data access tools when used in conjunction with CONNX.

Dynasets vs. Snapshots in Microsoft Access

Microsoft Access has two methods of retrieving data: dynasets and snapshots.

- **DYNASETS**
  When data is retrieved with a dynaset, the primary key value for every record in your selection is returned first. Based on these key values, one additional query per row is issued to retrieve the selected columns for that row. The end result is that for a query that retrieves 100 rows, 101 SQL requests are sent to the server - one request to retrieve the primary keys for the query, and one additional request per row to retrieve the selected columns.

Advantages

- The user can quickly jump to the bottom of a result set without retrieving the data rows in the middle of the query.
- The user can refresh the viewed data by creating a second query based on the primary keys retrieved initially.
- The user can add, modify, or delete a row of data based on the primary keys retrieved initially.
- Rapid return of DAO dynasets.

Disadvantages

- Much slower than snapshots.
- Network traffic is greater than with snapshots.

- **SNAPSHOTS**
  When data is retrieved with a snapshot, the data is retrieved in one pass, and the results are stored locally on your computer.

Advantages

- Faster than dynasets because the data is retrieved in one pass.
- Network traffic is less than dynasets.

Disadvantages

- The user cannot refresh or modify the viewed data.
- The viewed data represents the state of the data at the time of retrieval.

More Access Tips

- **Read-only capability**
  When creating read-only forms based on ODBC queries, make sure the Allow Edits property of the form is disabled.
- **Creating reports**
  When viewing static ODBC data in Access, create a report based on the query instead of viewing the query directly. All Access reports are snapshots instead of dynasets and execute faster than their query counterpart.
- **Executing queries**
  When executing queries that take longer that 60 seconds, be sure to set the ODBC time-out property of the query to zero.

```
[ODBC]
QUERYTIMEOUT=0
```

With version 8.6, when you are executing queries, the default behaviour is to run the query asynchronously. This means that after every ODBC api call, control is returned to Access so that Access may repaint itself so that users do not get the impression that Access has stopped running. This handoff of control means queries take longer to execute and return all data.

To force CONNX to run the queries synchronously, a setting needs to be added via the Configuration Manager. Under the CONNX heading, a new value called ASYNCACCESS needs to be created with the value data of 0. To restore asynchronous functionality, change the value data to 1, or remove the ASYNCACCESS value.

[HKEY_LOCAL_MACHINE\SOFTWARE\CONNX SOLUTIONS\CONNX]

ASYNCACCESS=0

NUMBER (REG_DWORD)

With version 8.7, when you are executing queries, the default behavior is to run the query synchronously. This means that CONNX will not return control to Access until the query has completed execution. While the query is executing, Access will not respond to mouse clicks or key presses. This means that Access will not repaint its window and you will not be able to cancel the query with Ctrl+Break. When Access does not repaint the window, you may get the impression that Access has hung, but this is not the case.

To force CONNX to execute querues asynchronously, a setting needs to be added via the Configuration Manager. Under the CONNX heading, a new value called "ASYNCACCESS" needs to be created with the value data of "1". This will allow you to cancel the query with Ctrl+Break, and allows Access to repaint its window properly. There can be a severe impact to performance when the query is executing asynchronously, due to the excessive transferring of control between applications.

[HKEY_LOCAL_MACHINE\SOFTWARE\CONNX SOLUTIONS\CONNX]

ASYNCACCESS=1

NUMBER (REG_DWORD)

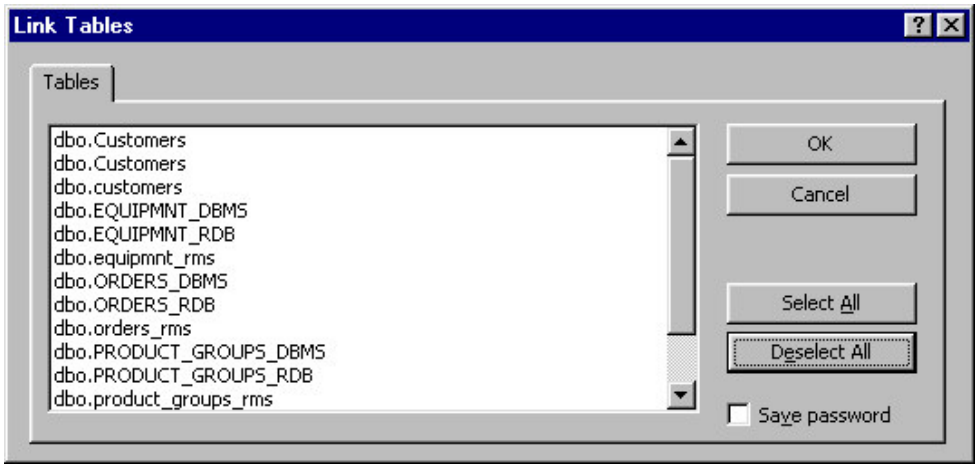
Another caveat of the query executing asynchronously is the fact that you will need to set the "ODBCTimeout" value for queries that require more than 60 seconds to execute.

[ODBC]

QUERYTIMEOUT=0

To restore synchronous functionality, change the value data of ASYNCACCESS to 0, or remove the value from the registry.

- **Catalog support**
  Microsoft Access supports two-part table names that include only the schema and object name.
  The three Customers tables in the Link Table dialog box below cannot be successfully linked because Access cannot identify the data source.



  Identify the data source by renaming your tables before attempting to link them within Microsoft Access.

Dynasets vs. Snapshots in Microsoft Visual Basic

Visual Basic uses the same two methods of retrieving data: dynasets and snapshots.

- **DYNASETS**
  When data is retrieved with a dynaset, the primary key value for every record in your selection is returned first. Based on these keys, one additional query per row is issued to retrieve the selected columns for that row. The end result is that for a query that retrieves 100 rows, 101 SQL requests are sent to the server - one request to retrieve the primary keys for the query, and one additional request per row to retrieve the selected columns.

Advantages

- The user can quickly jump to the bottom of a result set without retrieving the data rows in the middle of the query.
- The user can refresh the viewed data by creating a second query based on the primary keys retrieved initially.
- The user can add, modify, or delete a row of data based on the primary keys retrieved initially.

Disadvantages

- Much slower than snapshots.
- Network traffic is greater than snapshots.

- **SNAPSHOTS**
  When data is retrieved with a snapshot, the data is retrieved in one pass, and the results are stored locally on your computer.

Advantages

- Faster than dynasets because the data is retrieved in one pass.
- Network traffic is lower than with dynasets.

Disadvantages

- The user cannot refresh or modify the viewed data. The viewed data represents the state of the data at the time of retrieval.

More Visual Basic Performance Tips

- When creating read/only queries, use snapshots instead of dynasets.
- Take advantage of RDO (Remote Data Objects) available in Visual Basic 4 and Visual Basic 5.
- Remote Data Objects bypass the snapshot versus dynaset issues by sending SQL directly to the CONNX ODBC driver, without any Jet Engine processing. This results in optimum performance for Visual Basic applications that need to communicate with ODBC databases.
- **Visual Basic 4 or 5 with DAO**
  If using Visual Basic 4 or 5 with DAO, create the registry entry:

\Hkey_Local_Machine\Software\Microsoft\Jet\3.5\Engines\ODBC

In this key, add the numeric DWORD value QUERYTIMEOUT=0

- **Visual Basic 4 or 5 with DAO**
  Create the registry entry:

\Hkey_Local_Machine\Software\Microsoft\\Jet\3.5\Engines\ODBC

In this key, add the numeric DWORD value TRYJETAUTH=0

New Users and Microsoft Access Database Links

Microsoft Access tables are generally imported into CONNX through use of the default user name "Admin." Security levels remain at default levels during import.

When starting the import procedure for OLE DB/ODBC Providers using Microsoft Jet Engine 3.51 or 4.0, users should start the application using an appropriate DSN and log into CONNX. If it is the first time the user has opened the CDD, a CONNX dialog box appears with the following message:

"The user does not exist in the CONNX data dictionary. Would you like to create an integrated CONNX account for this user name?"

If this dialog box appears, the system fails in connecting to the Access database file, and the CONNX Database Login dialog box appears.

This error occurs because CONNX is attempting to use the CONNX user name and password for the Access database login. Since the Access database is expecting the user name "Admin" and no password to be entered, this initial attempt at logging on fails.

To correct this situation, change the displayed user name to Admin and leave the password blank. Do not change the server field.

To add a CONNX user manually through the CONNX administrator, you must enter the OLE DB provider name in the Server Name field, for example:

```
Microsoft.jet.oledb.4.0
```

OLE DB, Microsoft Access 97 Queries, and CONNX

Access 97 and Access 2000 databases contain both tables and queries. Some of the queries return tabular information. These queries can be treated as tables or views within CONNX with some limitations:Tables derived from queries cannot be edited. Care must be used with such tables if the resulting tabular data always varies, for example, if a query is designed to add a row to a table and then return a selection from that table. Every time that query is used, another row is added to that table, and it cannot be removed. Therefore, it is possible to issue an SQL statement that inadvertently accesses this table/query a number of times, yielding uncertain results.

Note: Queries that do not produce tabular data in Microsoft Jet 3.51 OLE DB Provider may be presented as importable tables. In version 4.0 , however, queries that do not produce tables are not presented as importable objects. Do not try to import non-tabular queries into CONNX using the older 3.51 version of the Access OLE DB provider.

OLE DB, Access 97, Access 2000, and CONNX

The newer Access OLE DB provider Microsoft Jet 4.0 OLE DB Provider can read both Access 97 and Access 2000 formatted database files (*.mdb). (The older 3.51 provider is limited to only Access 97 formatted databases.)

Normally, the Access 2000 databases cannot be used with Access 97. However, if you import Access 2000 tables into a CONNX CDD and create a DSN for this CDD,You can then create a linked table to that CONNX data source name within Access 97. CONNX insulates the differences between Access 97 and Access 2000 and enables Access 97 users to access Access 2000 database files.

Microsoft SQL Server

ODBC, Microsoft Access 97 for SQL Server 7.0, and CONNX

There is a standing incompatibility between Microsoft Access 97 and the Microsoft ODBC driver for SQL Server 7.0. If you link a table to a SQL Server 7.0 database through the ODBC driver, Access cannot read SQL Server NTEXT columns. (SQL Server 7.0 has an arbitrarily long UNICODE text field; the Access equivalent data type is MEMO.)

CONNX supports the SQL Server 7.0 NTEXT data type, which is transparently converted to a non-UNICODE string and is handled as a LONGVARCHAR (a CLOB or String BLOB)). This CONNX data type can be manipulated by Access 97. Using CONNX as a transition layer between the Access application and a SQL Server database corrects the incompatibility. Import the SQL Server 7.0 data into an CONNX CDD and create a DSN for it. Then in Access 97, create a linked table to the CONNX data source.

OLE DB Non-Indexed Tables and SQL Server 7.0

The CONNX Query Optimizer uses index information to process tables more efficiently. However, some SQL Server 7.0 tables do not contain index information. CONNX cannot access data stored in SQL Server 7.0 tables that do not contain index information and generates an error.

To prevent this error from occurring when importing SQL Server 7.0 tables, pass the SQL statement CREATE TABLE to the CONNX driver, and follow it with the CREATE UNIQUE INDEX.

Tools such as the Microsoft SQL Server 7.0 Enterprise Manager can be used to add index information to tables. If an index is added to a table after the table is imported into CONNX, the table metadata should be re-imported so that the index information can be accessed.

The Data Transformation Service (DTS) of Microsoft SQL Server 7.0 Enterprise Manager copies and transforms data from one data source to another, but does not replicate key information. Index information must be replaced or inserted in tables generated by DTS.

Troubleshooting OLE DB/ODBC-compliant Providers and the MDAC Configuration

A computer that uses CONNX OLE DB or ODBC data must have MDAC version 2.1 or later (Microsoft Data Access Components). If the CONNX CDD Administrator does not display selections for providers when importing from OLE DB or generates an error when selecting an ODBC DSN, it is possible that MDAC is not present or has been damaged. In most cases, reinstalling MDAC will solve this problem. A connection error is generated by any machine attempting to use a CONNX data source without the required MDAC.

If an error is issued during OLE DB import procedures when you are attempting to select a provider, it is possible that you are working with a Windows 95 system (Win95B, also known as Win95-OSR) installed with Internet Explorer 3.0, DCOM95, and MDAC 2.1. A key feature called Data Links that is used to import OLE DB objects may be missing from the setup.

To correct this problem, install Internet Explorer version 4.01 or a later version, and then reinstall MDAC version 2.1 or later.

Existing CONNX data sources can continue to be used without upgrading a Win95B machine to Internet Explorer 4.01.

Troubleshooting RMS Data Files

CONNX CDD warning message appears: "Warning: Column #XXX has a offset+length greater than the record size." (where #XXX is the column number).

This warning message appears whenever there is a discrepancy between RMS data file record length and the CONNX Data Dictionary table record length created for the data file. In the warning message box, the column number is displayed showing where the discrepancy begins.

IMPORTANT: *Any CONNX CDD table returning this warning will not yield correct results for any column beyond the column number specified in this warning message!*

1. Files with variable record length:

   A variable length record within an RMS file can be longer than the record length value displayed on the Table Properties tab in the CONNX Data Dictionary Manager window. Repetitive record segments (also known as arrays) are found within RMS file structures. Instead of a fixed number of repetitions, the number varies based on a counter field value stored within each record. Since the length of each record is calculated individually, the total record length can be different for each record in the RMS file.

   Any RMS file containing repeating segments using counter fields should be imported into separate CDD tables to ensure offset accuracy.

2. Files with fixed record length:

   The record imported or built in the CONNX Data Dictionary does not have the same record length as the actual RMS data file. To check the CONNX table record length, look at the record length value on the Table Properties tab in the CDD.

   To check the record length of the actual RMS file, perform a full directory of this file on your VAX or Alpha server. (At the VMS prompt type DIR/FULL YOURFILE.DAT and look for the "record format" line on the directory listing).

3. RMS file has a fixed record length and the two record lengths agree:

   Check the length of the CDD table against the record length in the tables properties tab. Add the Table Columns tab greatest offset value to its length to find the total record length for a fixed length record.

   If any field(s) were redefined or entered manually into the CDD, check those offsets for accuracy. If a partial record was defined, then the offset added to the length should not exceed the record length on the Tables Properties tab.

SCT-specific Troubleshooting

1. Where the data file is an SCT RMS file imported prior to CONNX version 8.6:

   The import has been enhanced to exclude unusable non-rotated records during import procedures. In earlier versions of CONNX, the records were imported by default and available for writing queries. Examples of such CONNX tables include the AAFILE, RTFILE, FSFILE, and SAFILE tables.

   Instead of building queries from the main table, choose just the segment of the table required for your query. For example, use the AAFILE_ROOT_SEG, AAFILE_BS, and AAFILE_AP for the AAFILE queries.

2. Where the "Exclude unusable non-rotated records" check box was not selected on the RMS SCT COBOL FD Files option in the Import CDD dialog box:

   The CONNX Data Dictionary Manager import option SCT COBOL FD IMPORT check box does not import unusable non-rotated records. If this check box is not selected, the import may include tables that cannot be used for queries.

Troubleshooting VSAM Data Files

CONNX CDD warning message appears: "Warning: Column #XXX has a offset+length greater than the record size." (where #XXX is the column number).

This warning message appears whenever there is a discrepancy between VSAM data file record length and the CONNX Data Dictionary table record length created for the data file. In the warning message box, the column number is displayed showing where the discrepancy begins.

IMPORTANT: *Any CONNX CDD table returning this warning will not yield correct results for any column beyond the column number specified in this warning message!*

1. Files with variable record length:

   A variable length record within a VSAM file can be longer than the record length value displayed on the Table Properties tab in the CONNX Data Dictionary Manager window. Repetitive record segments (also known as arrays) are found within VSAM file structures. Instead of a fixed number of repetitions, the number varies based on a counter field value stored within each record. Since the length of each record is calculated individually, the total record length can be different for each record in the VSAM file.

   Any VSAM data file containing repeating segments using counter fields should be imported into separate CDD tables to ensure offset accuracy.

2. Files with fixed record length:

   The record imported or built in the CONNX Data Dictionary does not have the same record length as the actual VSAM data file.

   You can use the IBM IDCAMS utility to check the length of a VSAM file. A demonstration of how to list the VSAM file attributes is shown in the following graphics which list the VSAM file attributes from the OS/390 ISPF option 3.4 (Figures 1 through 7) and the VSAM file attributes achieved via an IDCAMS command invoked via the TSO shell (Figures 8 through 9):

List VSAM File Attributes from the OS/390 ISPF Option 3.4

Figure 1

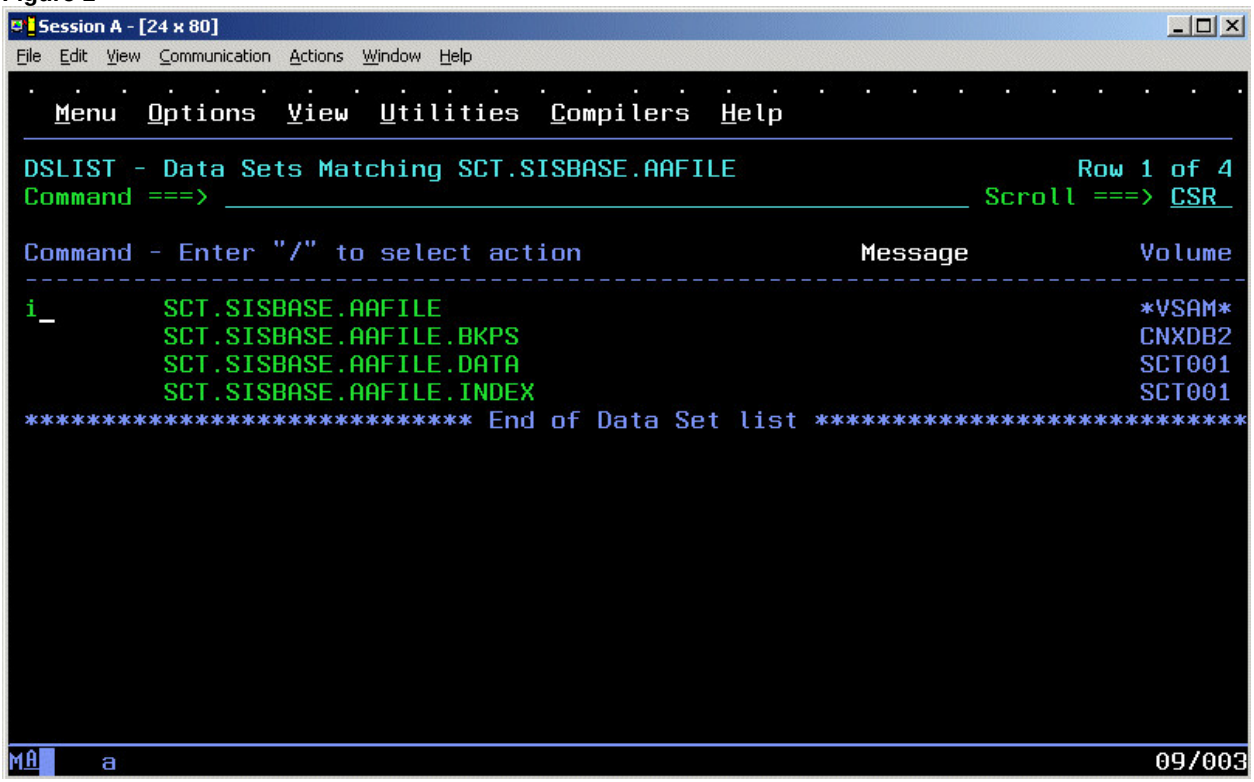


Figure 2

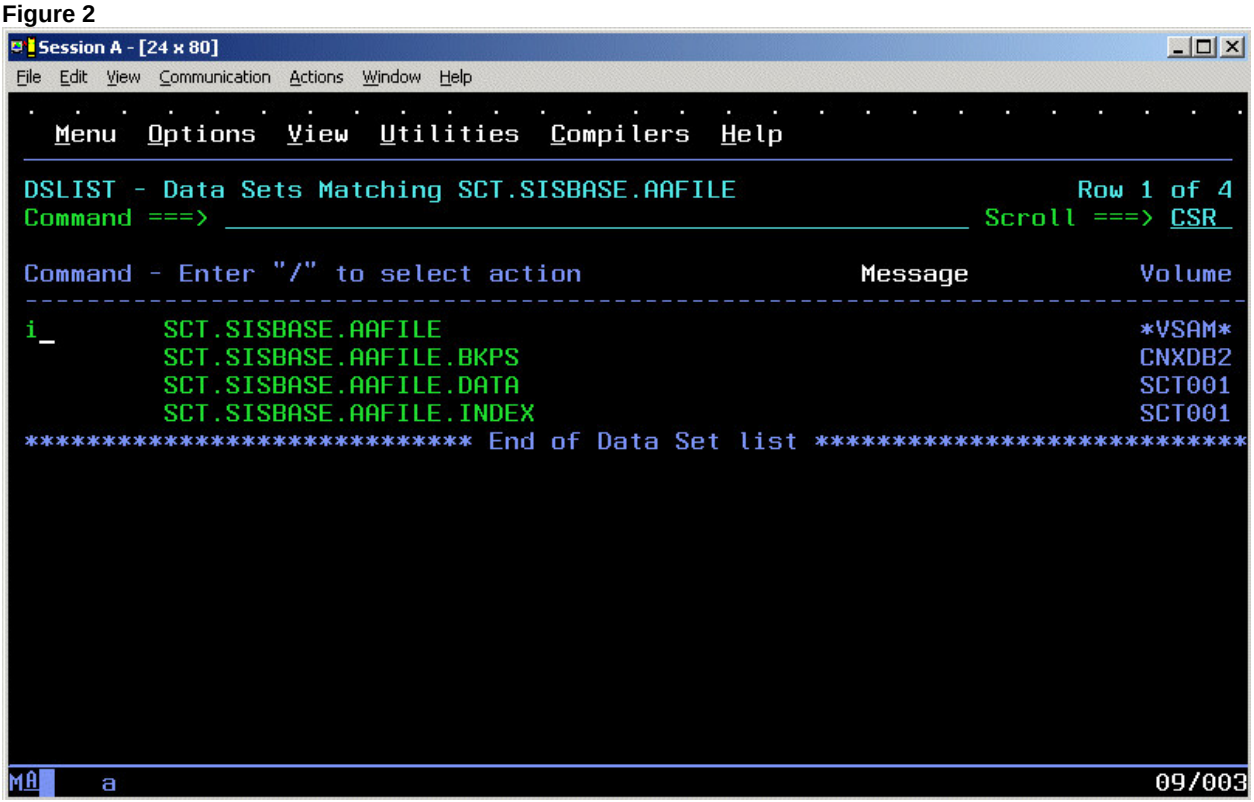


Figure 3

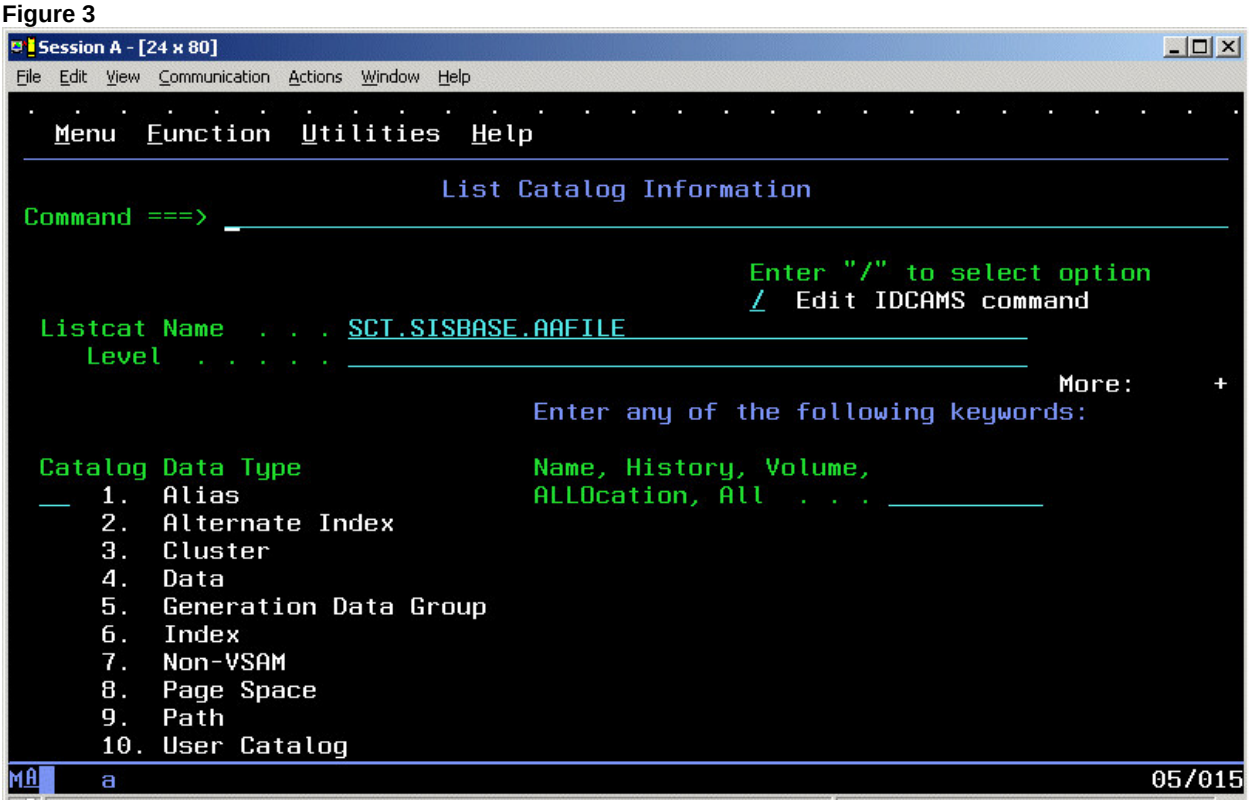


Figure 4

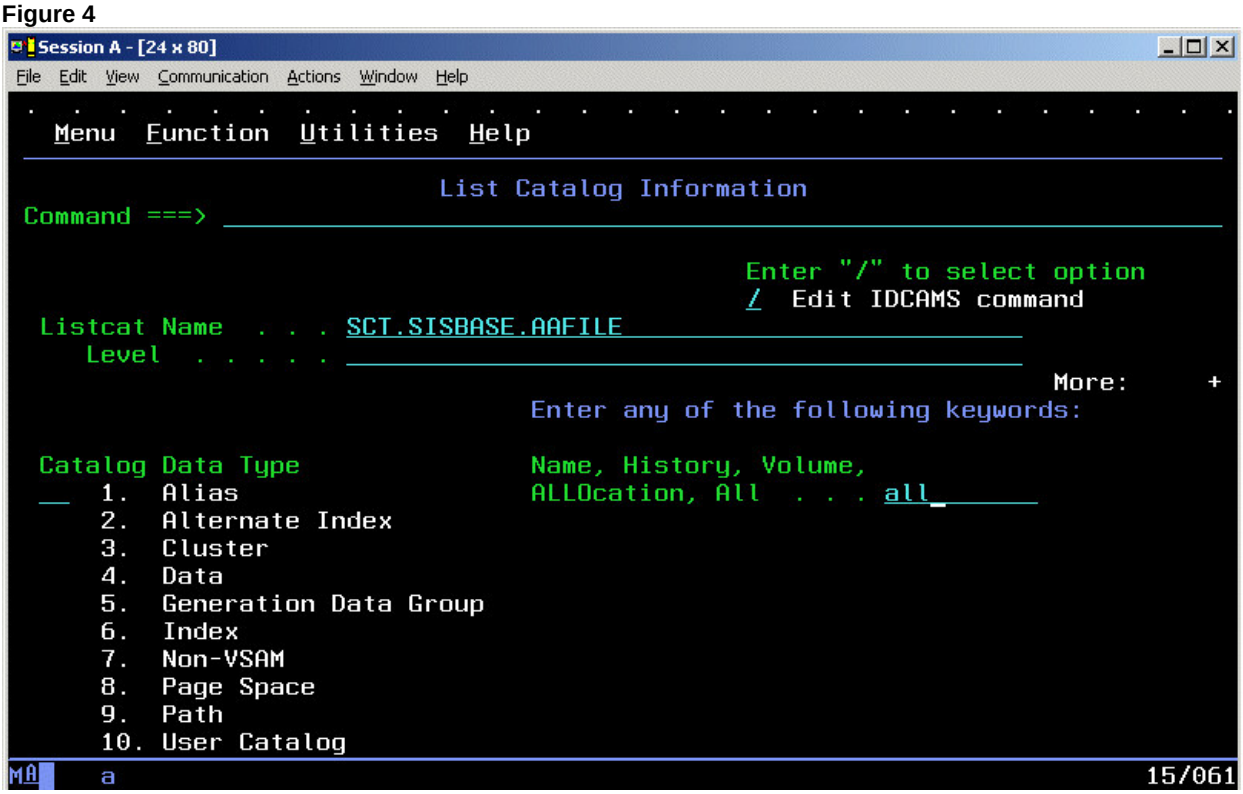


Figure 5

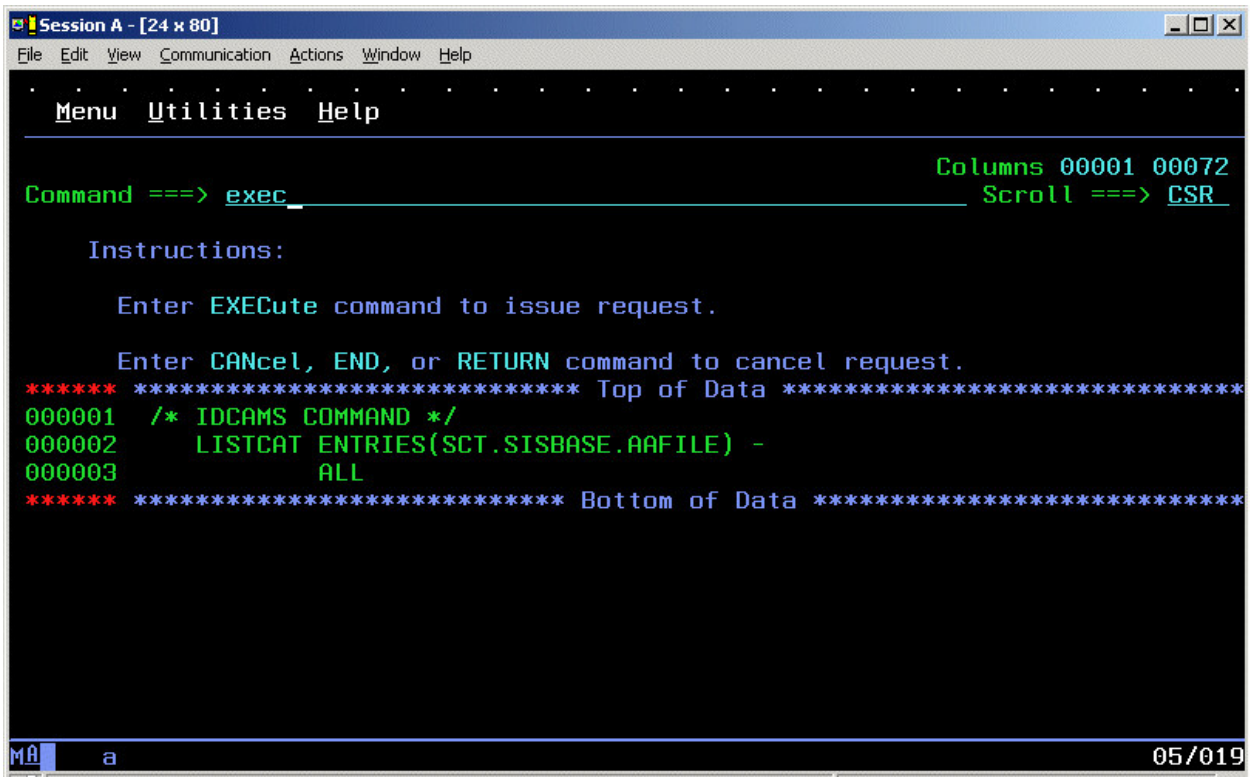


Figure 6

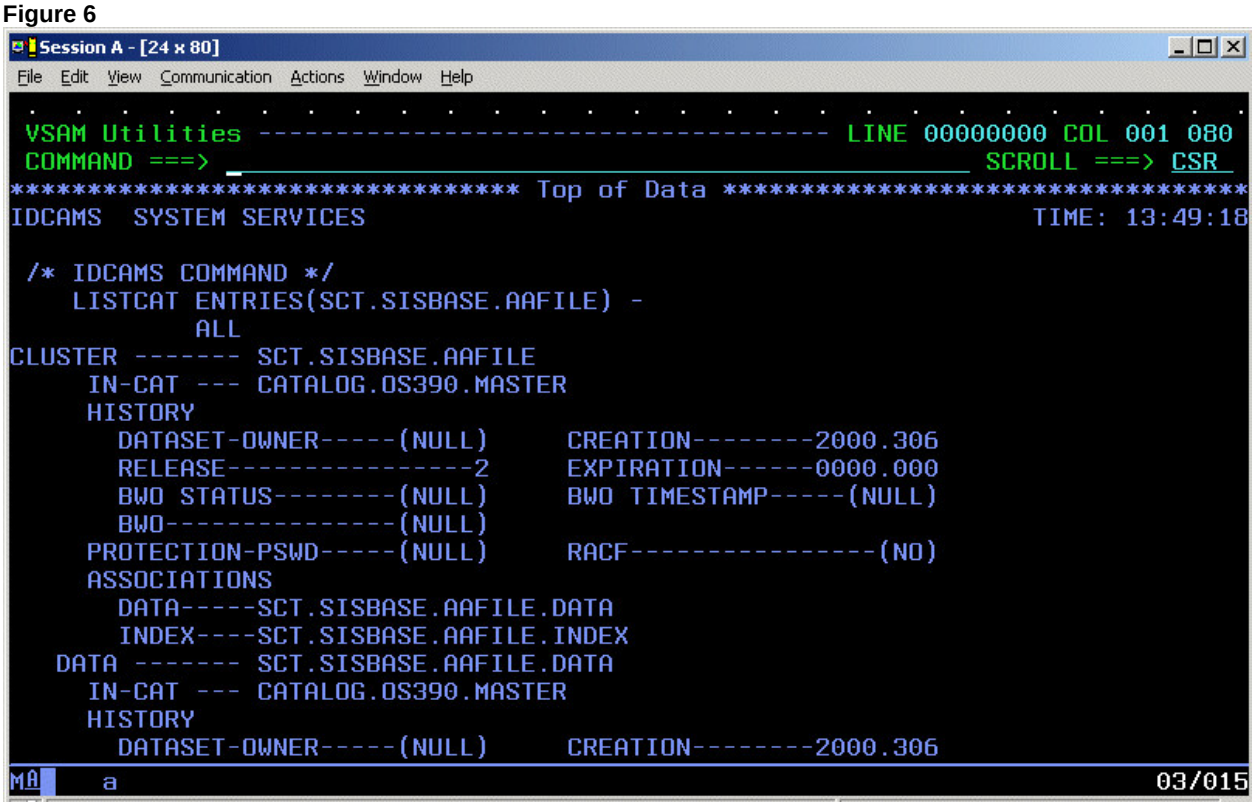
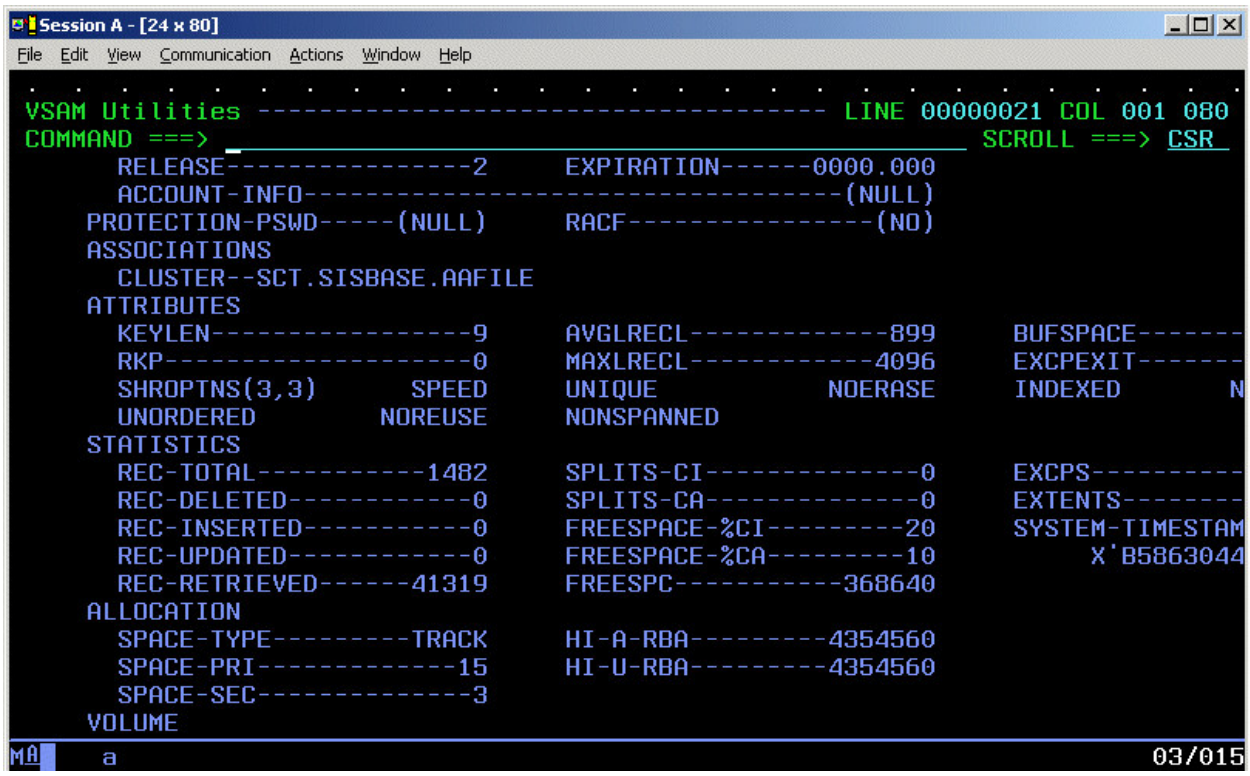


Figure 7



List VSAM File Attributes via a TSO Command
Figure 8

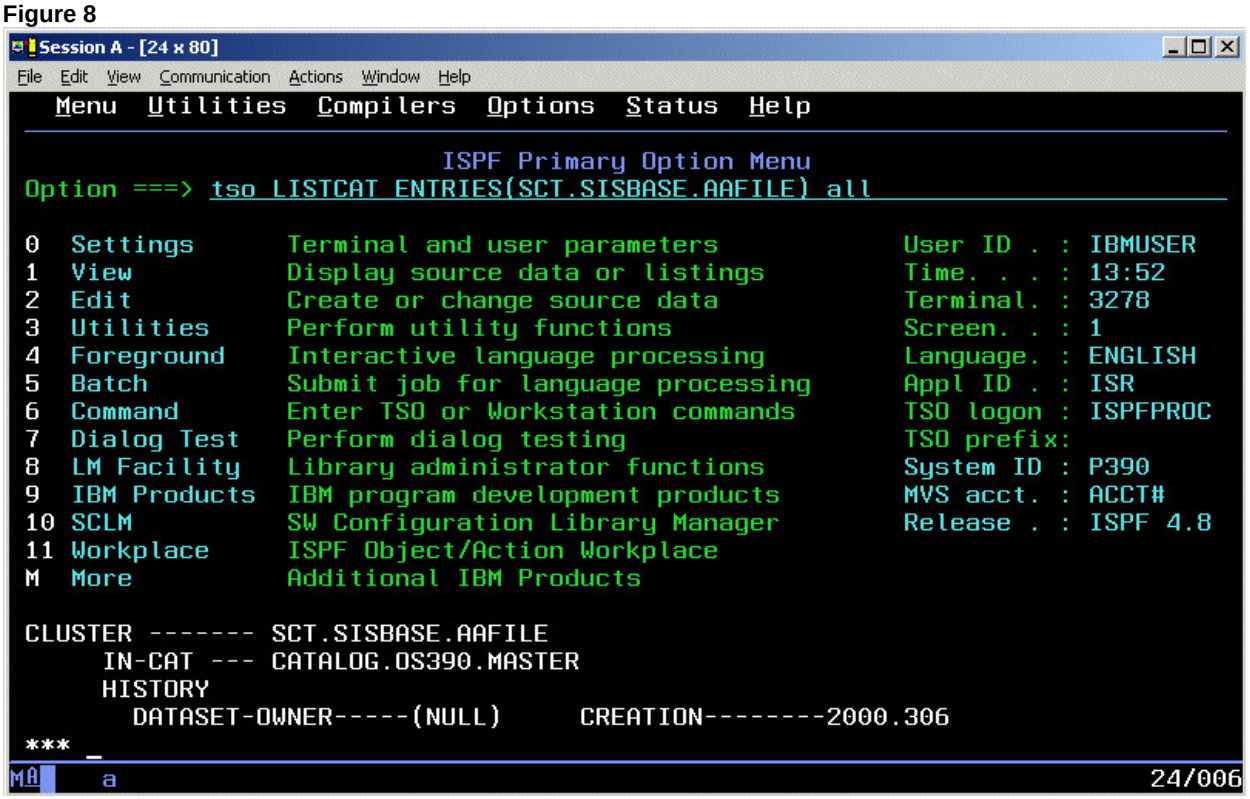
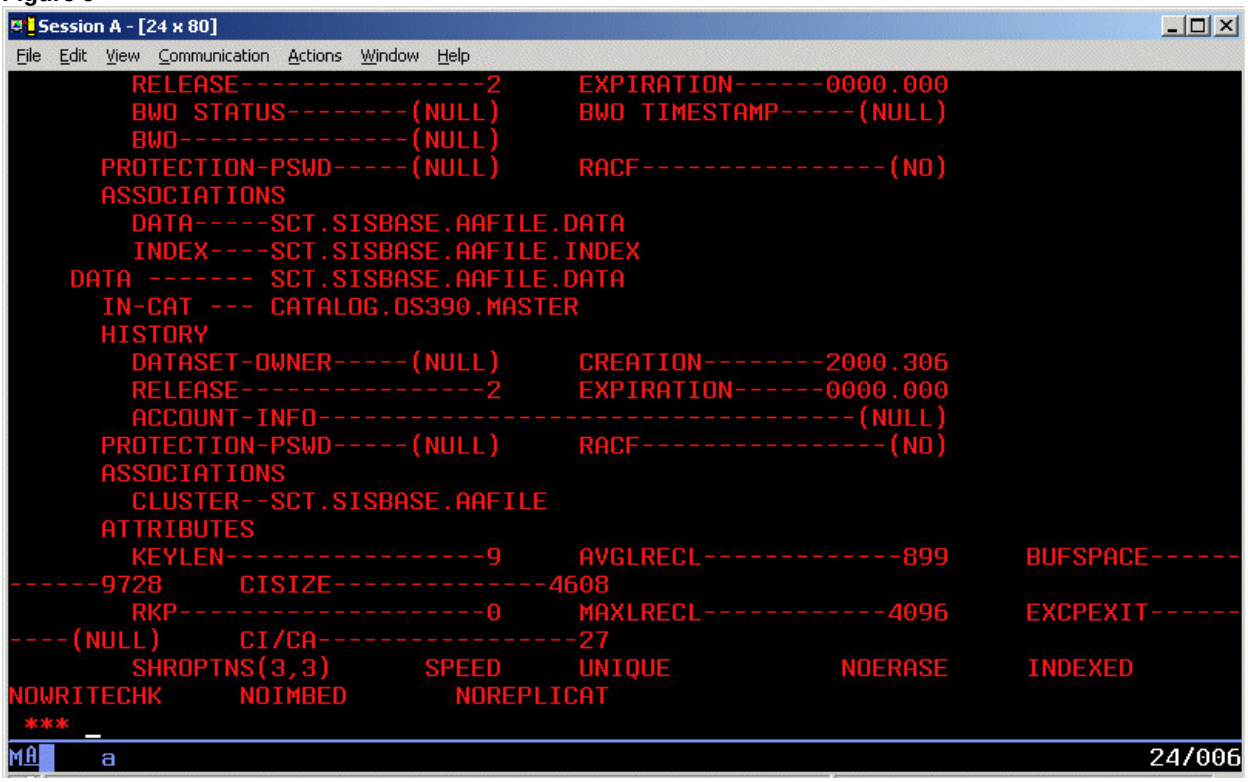


Figure 9



3. VSAM file has a fixed record length and the two record lengths agree:

You can verify that a VSAM file contains fixed length records by executing a LISTCAT command as in Step 2. The AVGLRECL and MAXLRECL values are equal for fixed-length VSAM files.

ODBC Driver - SQL Performance Tips

Indexes list keywords and other data that direct the user to the location of specific information stored in database tables. A database table can have one or more indexes associated with it. The efficiency of your searches for information stored in your tables can be improved dramatically if your program or query can use indexes and keys to access the information.

Indexes and keys related to your tables are defined in the CONNX Data Dictionary.

To get the most out of the CONNX ODBC driver, and to improve your search results, note the following hypothetical SQL substitutions:

When selecting a range of values:
(COL1 is an indexed column.)

Instead of

```
select * from <yourtable>
where col1 <= mAXvalue
and col1 >= mINvalue
```

Use

```
SELECT * FROM <YOURTABLE>
WHERE COL1 BETWEEN MINVALUE AND MAXVALUE
```

When selecting a set of values:

Instead of

```
SELECT * FROM <YOURTABLE>
WHERE COL1 = TESTVALUE1
OR COL1 = TESTVALUE2
OR COL1 = TESTVALUE3
```

Use

```
SELECT * FROM YOURTABLE
WHERE COL1 IN (TESTVALUE1, TESTVALUE2, TESTVALUE3)
```

When joining tables:

Use a constricting WHERE clause in all tables referenced in a join.

Instead of

```
SELECT * FROM <YOURTABLE1>,< YOURTABLE2>
WHERE YOURTABLE1.COL1 = TESTVALUE
AND YOURTABLE1.COL1 = YOURTABLE2.COL1
```

Use

```
SELECT * FROM <YOURTABLE1>, <YOURTABLE2>
WHERE YOURTABLE1.COL1 = TESTVALUE
AND YOURTABLE1.COL1 = YOURTABLE2.COL1
AND YOURTABLE2.COL1 = TESTVALUE
```

Adabas Performance Tuning

Sub/Super descriptor Handling

This topic only applies if you query a table containing a super or sub descriptor with the NU (Null Suppression) attribute or NC (SQL Null capable) attribute defined on one of the descriptor constituent fields.

To improve performance, descriptors that have NU or NC constituent fields could omit or skip records when the NU/NC column is blank. This descriptor behavior makes them unusable from an ANSI SQL perspective, because in SQL, records should always be returned even if a null value exists in a particular field. But not using these descriptors can significantly slow performance.

The following describes situations where NU/NC super descriptors will be used in SQL optimization, and what can be done to force the use of these performance enhancing descriptors.

CONNX does not allow partial key searches if an Adabas super descriptor has a NU (Null Suppressed) or NC (SQL Null capable) constituent field. Super descriptors that contain NU or NC constituent fields will be used only if criteria for every constituent fields is specified in the SQL statement. This applies to L3 calls, Sx calls and usage of super descriptors for ordering (Order by).

There are three ways to use a super descriptor when there are unknown criteria for some constituent fields:

1) Specify **column IS NOT NULL** on the constituent fields of the super descriptor where the value is unknown. This is the best solution.

The following example will illustrate how a super descriptor will be used by the Adabas SQL Gateway.

The example setup:

a) Create a table with three NC (SQL Null Capable) fields and create a super descriptor on those three fields.

```
create table nullsuppressiontest10 (col1 char(3), col2 char (3), col3 char(3) ,
index idx1 (col1, col2, col3))
```

b) Insert a value into one of the columns, and SQL NULL into the remaining fields.

```
insert into nullsuppressiontest10  values ('AAD', NULL, NULL)
```

Example #1:

```
select * from  nullsuppressiontest10 order by col1
```

The SQL statement above will not use the super descriptor to optimize the "order by" because the criteria has not been specified against the col2 and col3 fields, which are both NC. If the SQL Gateway used the super descriptor for ordering, records could be missed because they are not present in the super descriptor.

Example #2:

```
select * from  nullsuppressiontest10 where col1 is not null and col2 is not null
and col3 is not null order by col1
```

The SQL statement above will use the super descriptor to optimize ordering. It is safe to use the super descriptor because the criteria states we are not interested in rows that contain a null value.

Example #3:

```
select * from  nullsuppressiontest10 where col1 = 'AAD'
```

The SQL statement above will not use the super descriptor to optimize result retrieval, because the criteria hasn't been specified against the constituent fields col2 and col3. If records exist in the file where these fields are null, they will be skipped by Adabas and incorrect SQL results will be returned.

Example #4:

```
select * from  nullsuppressiontest10 where col1 = 'AAD' and col2 is not null and
col3 is not null
```

The SQL statement above will the super descriptor to optimize result retrieval. It is safe to use the super descriptor because the criteria states we are not interested in rows that contain a null value.

2) Suffix the escape clause {forceadanukey} to the end of the SQL statement. The SQL Gateway will use the super descriptor even though it may not return the correct "SQL" results.

The example setup:

a) Create a table with three NC (SQL Null Capable) fields and create a super descriptor on those three fields.

```
create table nullsuppressiontest10 (col1 char(3), col2 char (3), col3 char(3) ,
index idx1 (col1, col2, col3))
```

b) Insert a value into one of the columns, and SQL NULL into the remaining fields.

```
insert into nullsuppressiontest10  values ('AAD', NULL, NULL)
```

Example #1:

```
select * from  nullsuppressiontest10 where col1 = 'AAD'
```

The SQL statement above will not use the super descriptor to optimize result retrieval, because the criteria hasn't been specified against the constituent fields col2 and col3. If records exist in the file where these fields are null, they will be skipped by Adabas and incorrect SQL results will be returned.

Example #2:

```
select * from  nullsuppressiontest11 where col1 = 'AAD' {forceadanukey}
```

The SQL statement above will the super descriptor to optimize result retrieval. Even though no criteria was specific for constituent columns col2 and col3, the SQL Gateway will use the super descriptor anyway because of the {forceadanukey} clause. This SQL statement may not return the correct ANSI SQL results, so care should be taken when using this escape clause.

3) Enable the global configuration setting FORCEADANUKEY in the CONNX Configuration Manager or the SQLRegistry. This will cause the SQL Gateway to use super descriptors regardless of null suppression status for all queries. Once you enable this global setting, use the escape clause {forceadanonukey} to turn off this behavior for specific queries.

Chapter 21 - Record Locking and Transactions

Transaction Support

CONNX supports coordinated transactions for Oracle, Oracle Rdb, IBM DB2, CICS/VSAM, DBMS, and some OLE DB/ODBC Providers that support transactions through the ODBC and OLE-DB specification. This is accomplished through the ODBC SQLTransact API. Most applications have higher level wrappers around the ODBC and OLE API and methods for committing and rolling back transactions. Consult the documentation for your front-end application for details on the use of transactions.

Transactions are an SQL tool used to maintain database integrity. They are started before an SQL statement is executed. All SQL operations are recorded and are undone if a ROLLBACK transaction operation occurs. If a COMMIT transaction operation occurs, the database changes are made permanent, and the transaction log is cleared.

When a COMMIT command is issued, all of the statements in the transaction are applied to the database at once. When a ROLLBACK is issued, the statements are reversed, and the database is returned to its original state.

You can issue a ROLLBACK anytime before issuing a COMMIT command. If CONNX is exited in mid-operation, all commands are automatically rolled back. Issuing a COMMIT command saves any changes made up to that point.

Note: For CONNX for CICS/VSAM, COMMIT/ROLLBACK is implemented via the traditional EXEC SYNCPOINT and EXEC CICS SYNCPOINT ROLLBACK commands. The efficacy of these commands is contingent on whether the target VSAM files have been defined with journals or transaction logs.

Related Topics

- Dataflex
- POWERflex
- RMS
- Oracle Rdb
- DBMS
- DB2
- Oracle

Adabas

Record Locking

Record locking in Adabas is triggered if a command from ODBC, ADO, ADO.NET, JDBC, or OLE DB issues a lock. When a record is being updated, the record is locked, and then released when the transaction ends.

Transactions

Transactions are supported by the Adabas SQL Gateway (CONNX for Adabas). A transaction call causes changes to be committed in Adabas.

DataFlex

Record locking

Record locking in DataFlex occurs at the file level. This means that when a lock is required, the entire file must be locked, not just a single record. Because of this limitation, locking of records must be considered carefully when accessing DataFlex files.

Locking occurs automatically during updates or deletions of records through the use of UPDATE or DELETE SQL statements.

If data is selected with the intention of updating it through the use of the SELECT ___ FOR UPDATE SQL statement, or by setting the SQL_CONCUR_LOCK property of the statement handle, then CONNX will return an error if an attempt is made to read a table locked by another user.

Record locking in DataFlex files can be triggered by setting the lock property of the statement handle to SQL_CONCUR_LOCK when issuing an SELECT statement.

Implementation of this feature varies in each product.

Here is an example of an SQL statement used for DataFlex record locking in Visual Basic ADO:

```
rs.Open "select customerid from customers_dataflex", conn,
adOpenKeyset, adLockPessimistic
```

Transactions

Transactions are not supported by CONNX for DataFlex.

POWERflex

Record locking

Record locking in POWER*flex* occurs at the file level. This means that when a lock is required, the entire file must be locked, not just a single record. Because of this limitation, locking of records must be considered carefully when accessing POWER*flex* files.

Locking occurs automatically during updates or deletions of records through the use of UPDATE or DELETE SQL statements.

If data is selected with the intention of updating it through the use of the SELECT ___ FOR UPDATE SQL statement, or by setting the SQL_CONCUR_LOCK property of the statement handle, then CONNX will return an error if an attempt is made to read a table locked by another user.

Record locking in POWER*flex* files can be triggered by setting the lock property of the statement handle to SQL_CONCUR_LOCK when issuing an SELECT statement.

Implementation of this feature varies in each product.

Here is an example of an SQL statement used for POWER*flex* record locking in Visual Basic ADO:

```
rs.Open "select customerid from customers_powerflex", conn,
adOpenKeyset, adLockPessimistic
```

Transactions

Transactions are not supported by CONNX for POWER*flex*.

RMS

Record locking

RMS record locking occurs at the record level, and works well with high performance and high volume applications.

Record locking in RMS is automatically used when updating or deleting records with the UPDATE or DELETE SQL statement.

During record selection, CONNX uses the RMS GET REGARDLESS flag which allows CONNX to read records locked by other applications.

Use of Get Regardless enables the user to select data currently locked by other applications. However, if data is selected with the intent to update it through the use of the SELECT .. FOR UPDATE SQL Statement, or by setting the SQL_CONCUR_LOCK property of the statement handle, an error is returned if an attempt is made to read a record locked by another user.

Locking can be triggered when issuing a SELECT statement by setting the lock property of the statement handle to SQL_CONCUR_LOCK.

Implementation of this feature varies in each product:

Here is an example of an SQL statement used for RMS record locking in Visual Basic ADO:

```
rs.Open "select customerid from customers_rms", conn, adOpenKeyset,
adLockPessimistic
```

Locking notes for RMS programmers:

By default, CONNX will open all RMS files in using GET(FAB$M_GET) access, and

ALL(FAB$M_SHRGET|FAB$M_SHRUPD|FAB$M_SHRPUT|FAB$M_SHRDEL)

share access.

CONNX will open a file using:

UPDATE(FAB$M_GET|FAB$M_UPD),

DELETE(FAB$M_GET|FAB$M_DEL) or

INSERT(FAB$M_GET|FAB$M_PUT) access

with a share access of

ALL(FAB$M_SHRGET|FAB$M_SHRUPD|FAB$M_SHRPUT|FAB$M_SHRDEL)

ONLY when performing an insert, update or delete.

Additionally, CONNX uses the RMS READ_REGARDLESS flag when it issues read/only selects, which allows CONNX to read records that are locked by another application if the intent is NOT to update the data.

Transactions

CONNX can support RMS transactions if the RMS file is flagged for RMS journaling. All applications that update the file must make additional RMS system calls to control the transaction. However, if journaling is enabled on an existing RMS file used by a VMS application that has not made the program changes, the VMS application will no longer be able to update the RMS file. This feature should only be used if RMS journaling is currently in use by existing VMS applications.

To activate this feature, the following system logical must be defined in the SYS$STARTUP:SYSTARTUP_VMS.COM file:

```
$DEFINE /SYSTEM CNXRMSJOURNAL 1
```

To enable full transaction support for all RMS files marked Recovery Unit Journaling, the following must be added to the registry file:

```
[RMS]
TRANSACTION = 1
```

RMS files can be marked for Recovery Unit Journaling with the following command:

```
$SET FILE/RU_JOURNAL rmsfilename
```

Refer to your RMS Journaling Manual for further details about journaling.

Related Topics

 CONNX Registry File Settings

Oracle Rdb

Record locking

Rdb record locking occurs at the record level. This level of locking granularity works well with high performance and high volume applications.

Locking is automatically used when updating or deleting records using the UPDATE or DELETE SQL statement.

When selecting records with no intent to update, CONNX issues a SQL statement in a READ ONLY transaction. This minimizes any locks placed on the data. However, if data is selected with the intention to update through the use of the SELECT .. FOR UPDATE SQL statement or by setting the SQL_CONCUR_LOCK property of the statement handle, CONNX will issue a read/write transaction. An error message is returned if an attempt is made to read a record locked by another user.

Locking can be triggered when issuing a SELECT statement by setting the lock property of the statement handle to SQL_CONCUR_LOCK.

The implementation of this feature varies in each product:

Example in Visual Basic ADO:

```
rs.Open "select customerid from customers_rdb", conn, adOpenKeyset,
adLockPessimistic
```

Locking notes for Rdb programmers:

Locking for Rdb should be controlled by the use of transactions. The CONNX Rdb module supports transactions which guarantee that a complete unit of work is performed. If any part of that unit work generates an error, the complete unit can be rolled back.

By default, CONNX is in "Automatic Transaction" mode.

This means that each SQL statement is placed in a separate transaction.

To control the use of transactions in your application, change to Manual Transaction mode.

The implementation of this feature varies in each product:

Example in Visual Basic ADO:

```
Conn.BeginTrans
rs.Open "select customerid from customers_rdb", conn, adOpenKeyset,
adLockPessimistic
conn.CommitTrans
```

Records accessed in a transaction are locked against reads and updates. The locks are released when the transaction is committed or rolled back.

DBMS

Record locking

DBMS locking occurs at the record level. This level of locking granularity works well with high performance and high volume applications.

Locking is automatically used when updating or deleting records with the UPDATE or DELETE SQL statement.

When selecting records with no intent to update, CONNX issues a SQL statement in a READ ONLY transaction. This minimizes any locks placed on the data. However, if data is selected with the intention to update through the use of the SELECT .. FOR UPDATE SQL statement or by setting the SQL_CONCUR_LOCK property of the statement handle, CONNX issues a read/write transaction. An error message is returned if an attempt is made to read a record locked by another user.

Locking can be triggered when issuing a SELECT statement by setting the lock property of the statement handle to SQL_CONCUR_LOCK.

The implementation of this feature varies in each product:

Example in Visual Basic ADO:

```
rs.Open "select customerid from customers_dbms", conn, adOpenKeyset,
adLockPessimistic
```

Locking notes for DBMS programmers:

Locking for DBMS should be controlled by the use of transactions. The CONNX DBMS module supports transactions. Transactions guarantee that a complete unit of work is performed. If any part of that unit work generates an error, the complete unit can be rolled back.

By default, CONNX is in an Automatic Transaction mode. This means that each SQL statement is placed in a separate transaction.

To control the use of transaction in your application, change to Manual Transaction mode.

The implementation of this feature varies in each product:

Example in Visual Basic ADO:

```
Conn.BeginTrans
rs.Open "select customerid from customers_dbms", conn, adOpenKeyset,
adLockPessimistic
conn.CommitTrans
```

Records accessed in a transaction are locked against reads and updates. The locks are released when the transaction is committed or rolled back.

DB2

Record locking

DB2 record locking occurs at the row level. This level of locking granularity works well with high performance and high volume applications.

Locking is automatically used when updating or deleting records with the UPDATE or DELETE SQL Statement.

When selecting records with no intent to update, CONNX issues a SQL statement in a READ ONLY transaction. This minimizes any locks placed on the data. However, if data is selected with the intention to update through the use of the SELECT .. FOR UPDATE SQL statement or by setting the SQL_CONCUR_LOCK property of the statement handle, CONNX issues a read/write transaction. An error message is returned if an attempt is made to read a record locked by another user.

Locking can be triggered when issuing a SELECT statement by setting the lock property of the statement handle to SQL_CONCUR_LOCK.

The implementation of this feature varies in each product:

Example in Visual Basic ADO**:**

```
rs.Open "select customerid from customers_db2", conn, adOpenKeyset,
adLockPessimistic
```

Locking notes for DB2 programmers:

Locking for DB2 should be controlled by the use of transactions. The CONNX DB2 module supports transactions. Transactions guarantee that a complete unit of work is performed. If any part of that unit work generates an error, the complete unit can be rolled back.

By default, CONNX is in an Automatic Transaction mode. This means that each SQL statement is placed in a separate transaction. To control the use of transaction in your application, change to Manual Transaction mode.

The implementation of this feature varies in each product:

Example in Visual Basic ADO:

```
Conn.BeginTrans
rs.Open "select customerid from customers_db2", conn, adOpenKeyset,
adLockPessimistic
conn.CommitTrans
```

Records accessed in a transaction are locked against reads and updates. The locks are released when the transaction is committed or rolled back.

Oracle

Record locking

Oracle record locking occurs at the record level. This level of locking granularity is good for high performance & high volume applications.

Locking is automatically used when updating or deleting records with the UPDATE or DELETE SQL Statement.

When selecting records with no intent to update, CONNX issues a SQL statement in a READ ONLY transaction. This minimizes any locks placed on the data. However, if data is selected with the intention to update through the use of the SELECT .. FOR UPDATE SQL statement or by setting the SQL_CONCUR_LOCK property of the statement handle, CONNX issues a read/write transaction. An error message is returned if an attempt is made to read a record locked by another user.

Locking can be triggered when issuing a SELECT statement by setting the lock property of the statement handle to SQL_CONCUR_LOCK.

The implementation of this feature varies in each product:

Example in Visual Basic ADO:

```
rs.Open "select customerid from customers_oracle", conn, adOpenKeyset,
adLockPessimistic
```

Locking notes for Oracle programmers:

Locking for Oracle should be controlled by the use of transactions. The CONNX Oracle module supports transactions. Transactions guarantee that a complete unit of work is performed. If any part of that unit work generates an error, the complete unit can be rolled back.

By default, CONNX is in an Automatic Transaction mode. This means that each SQL statement is placed in a separate transaction. To control the use of transaction in your application, change to Manual Transaction mode.

The implementation of this feature varies in each product:

Example in Visual Basic ADO:

```
Conn.BeginTrans
rs.Open "select customerid from customers_oracle", conn, adOpenKeyset,
adLockPessimistic
conn.CommitTrans
```

Records accessed in a transaction are locked against reads and updates. The locks are released when the transaction is committed or rolled back.

VSAM

Record Locking

Locking occurs automatically during updates or deletions of records through the use of UPDATE or DELETE SQL statements, which are implemented via the CICS READ, UPDATE, REWRITE, DELETE, and UNLOCK commands.

If data is selected with the intent to update it through the use of the SELECT ___ FOR UPDATE SQL statement, or by setting the SQL_CONCUR_LOCK property of the statement handle, then CONNX returns an error if an attempt is made to read a VSAM file, record, or set of records within the file which has been previously locked by one or more users.

Record locking in VSAM can be triggered by setting the lock property of the statement handle to SQL_CONCUR_LOCK when issuing a SELECT command.

Implementation of this feature varies in each product.

Here is an example of an SQL statement used for VSAM record locking in Visual Basic ADO:

```
rs.open "select customerid from customers_vsam", conn, adOpenKeyset,
   adLockPessimistic
```

Transactions

Transactions are supported by CONNX for VSAM via the CICS SYNCPOINT and SYNCPOINT ROLLBACK commands.

Note: These commands have no effect unless the VSAM files accessed via CONNX within the current unit of work are defined to CICS as recoverable resources.

C-ISAM, DISAM, and Micro Focus

Record locking

C-ISAM, DISAM, and Micro Focus, record locking occurs at the record level. This level of locking granularity is good for high performance & high volume applications.
Locking is automatically used when updating or deleting records with the UPDATE or DELETE SQL Statement.

When selecting records with no intent to update, CONNX issues a SQL statement in a READ ONLY transaction. This minimizes any locks placed on the data. However, if data is selected with the intention to update through the use of the SELECT .. FOR UPDATE SQL statement or by setting the SQL_CONCUR_LOCK property of the statement handle, CONNX issues a read/write transaction. An error message is returned if an attempt is made to read a record locked by another user.

Locking can be triggered when issuing a SELECT statement by setting the lock property of the statement handle to SQL_CONCUR_LOCK.

The implementation of this feature varies in each product:

Example in Visual Basic ADO:

```
rs.Open "select customerid from customers_cisam", conn, adOpenKeyset,
adLockPessimistic
```

Locking notes for C-ISAM, DISAM, and Micro Focus programmers:

Locking for C-ISAM, DISAM, and Micro Focus should be controlled by the use of transactions. The CONNX C-ISAM, DISAM, and Micro Focus module supports transactions. Transactions guarantee that a complete unit of work is performed. If any part of that unit work generates an error, the complete unit can be rolled back.

By default, CONNX is in an Automatic Transaction mode. This means that each SQL statement is placed in a separate transaction. To control the use of transaction in your application, change to Manual Transaction mode.

The implementation of this feature varies in each product:

Example in Visual Basic ADO, :

```
Conn.BeginTrans
rs.Open "select customerid from customers_cisam", conn, adOpenKeyset,
adLockPessimistic
conn.CommitTrans
```

Records accessed in a transaction are locked against reads and updates. The locks are released when the transaction is committed or rolled back.

Transactions

Important: *It should be noted that C-ISAM and DISAM performs all transactions at the Process level and that any actions taken in separate threads are all pooled into the single Process transaction. Consequently, transactions are not recommended in instances where multiple users are connecting through a JDBC server.*

TCP/IP Codes/States

This table summarizes potential ODBC State information messages which can be returned to CONNX.

TCP/IP Code/State	Message Text	Recommended Action
10004 (WSAEINTR)	Interrupted function call: a blocking operation was interrupted.	None required.
10013 (WSAEACCES)	Permission denied: Socket access attempt is prohibited by its access permissions.	Notify network administrator.
10014 (WSAEFAULT)	Bad pointer address detected in a socket function call pointer argument.	Notify CONNX Technical Support.
10022 (WSAEINVAL)	Winsock function called with an invalid argument.	Notify CONNX Technical Support.
10024 (WSAEMFILE)	No file descriptors are available: too many open sockets.	Close some connections and retry the connect; notify CONNX Technical Support if problem recurs.
10035 (WSAEWOULDBLOCK)	The requested operation would block a non-blocking socket.	Retry the operation.
10036 (WSAEINPROGRESS)	Blocking sockets operation in progress.	Retry the operation.
10037 (WSAEALREADY)	Operation already in progress. A second operation was attempted on a nonblocking socket.	Notify CONNX Technical Support.
10038 (WSAENOTSOCK)	Socket operation on non-socket.	Notify CONNX Technical Support.
10039 (WSAEDESTADDRREQ)	Destination address required.	Double-check the TCP/IP destination address and port number; notify CONNX Technical Support if problem recurs.
10040 (WSAEMSGSIZE)	Message too long.	Notify CONNX Technical Support.
10041 (WSAEPROTOTYPE)	Protocol wrong type for socket.	Notify CONNX Technical Support.
10042 (WSAENOPROTOOPT)	Unknown / unsuppported winsock option.	Notify CONNX Technical Support.
10043 (WSAEPROTONOSUPPORT)	Protocol not supported.	Probable configuration error; notify network administrator.
10044 (WSAESOCKTNOSUPPORT)	Socket type not supported.	Notify network administrator and CONNX Technical Support.
10045 (WSAEOPNOTSUPP)	Operation not supported.	Notify CONNX Technical Support.
10046 (WSAEPFNOSUPPORT)	Protocol family not supported.	Notify network administrator.
10047 (WSAEAFNOSUPPORT)	Address family not supported by protocol family.	Notify CONNX Technical Support.
10048 (WSAEADDRINUSE)	Requested address is in use. Only one usage of each socket address (protocol/IP address/port) is permitted.	Notify network and DB2 administrators.
10049 (WSAEADDRNOTAVAIL)	Cannot assign requested address.	Double-check TCP/IP destination address and port number; retry.

10050 (WSAENETDOWN)	Network subsystem failed.	Notify network administrator.
10051 (WSAENETUNREACH)	The requested network is currently unreachable.	Retry connect attempt later; also try pinging the destination address.
10052 (WSAENETRESET)	Remote host has reset the connection.	Retry connection; notify CONNX Technical Support if problem recurs.
10053 (WSAECONNABORTED)	Network dropped the connection because of a timeout or related failure.	Notify CONNX Technical Support.
10054 (WSAECONNRESET)	Connection was reset by the partner.	Retry connection; notify network and DB2 administrators if problem persists.
10055 (WSAENOBUFS)	No buffer space available: too many connections.	Close some client applications and try again; notify CONNX Technical Support if problem persists.
10056 (WSAEISCONN)	Socket is already connected.	Notify CONNX Technical Support.
10057 (WSAENOTCONN)	Socket is not connected.	Notify CONNX Technical Support.
10058 (WSAESHUTDOWN)	Cannot send after socket shutdown.	Notify CONNX Technical Support.
10060 (WSAETIMEDOUT)	The connect request timed out.	Verify that TCP/IP is supported and started on the target server, and that the IP address and port are correct.
10061 (WSAECONNREFUSED)	Connection refused.	Verify that TCP/IP is supported and started on the target server, and that the IP address and port are correct.
10064 (WSAEHOSTDOWN)	Host is down: socket operation failed.	Retry connection; notify network and DB2 administrators if problem persists.
10065 (WSAEHOSTUNREACH)	Requested network is currently unreachable from this host.	Retry connect attempt later; also try pinging the destination address.
10067 (WSAEPROCLIM)	Too many processes. The maximum number of simultaneous applications using a socket has been reached.	Close some client connections and retry; notify CONNX Technical Support if problem persists.
10091 (WSASYSNOTREADY)	Network subsystem is not ready for communication.	Probable client configuration problem; notify network administrator.
10092 (WSAVERNOTSUPPORTED)	The requested Windows Sockets version is not supported by the winsock DLL.	Probable client configuration problem; notify network administrator.
10093 (WSANOTINITIALISED)	Successful WSAStartup not yet performed.	Notify CONNX Technical Support.
10094 (WSAEDISCON)	Graceful shutdown in progress.	Retry connection.
11001 (WSAHOST_NOT_FOUND)	Host not found.	Double-check the TCP/IP address and the port number of the target server.
11002 (WSATRY_AGAIN)	Host not found; try again. The name server did not return the hostname IP address.	Retry the connection.
11003 (WSANO_RECOVERY)	A non-recoverable error occurred during a database lookup. One or more HOSTS, SERVICES, or PROTOCOLS files could not be found, or a DNS server returned a severe error.	Probable client configuration problem; notify network administrator.
11004 (WSANO_DATA)	Valid name, no data record of requested type. The hostname is	Probable network configuration problem; notify network administrator.

	not defined at the name server, or in the hosts file.	

C-ISAM

C-ISAM and DISAM: Error Messages

Error Code	Description	Action
100	An attempt to add a duplicate key index.	User Error - Cannot add duplicate keys.
101	An attempt to perform an action without opening first.	Contact Technical Support.
102	Bad Arguments.	Contact Technical Support.
103	Bad Key	Contact Technical Support.
104	Too many open files**.**	Increase the amount of file handles that can be opened on your system.
105	CISAM file corruption.	Use CISAM utility bcheck.
106	Exclusive Access Open Error.	Contact Technical Support.
107	Record locked.	Must wait until it is unlocked.
108	Duplicate key.	A duplicate key was attempted to be added.
109	Primary key deletion.	Primary Keys cannot be deleted.
110	End of file.	Contact Technical Support.
111	Record not found.	Contact Technical Support.
112	No current record.	Contact Technical Support.
113	File locked.	Must wait until File is unlocked.
114	File name too long.	Contact Technical Support.
115	Lock file cannot be created.	Contact Technical Support.
116	Memory problems.	Contact Technical Support.
117	Bad collation.	Contact Technical Support.
118	Log read errors.	Delete log file and create a new recovery.log file in the CONNX directory.
119	Bad log.	Delete log file and create a new recovery.log file in the CONNX directory.
120	Log open.	Create a new recovery.log file in the CONNX directory.
121	Log write.	Delete log file and create a new recovery.log file in the CONNX directory.
122	Not in transaction.	Contact Technical Support.
124	Begin transaction not found.	Contact Technical Support.
125	No NFS.	Check your Network File Server.
126	Bad record number.	Contact Technical Support.
127	No primary key.	Contact Technical Support.
128	No logging	Contact Technical Support.
129	Too many users.	You have exceeded amount of users.
131	No free disk space.	Free up some disk space.
132	Record too long.	If record is being created by CONNX through Create Table, reduce the length.
133	Audit trail exists.	Contact Technical Support.
134	No locks.	Contact Technical Support.
150	Demo limits exceeded.	Contact Technical Support.
153	No manual mode.	Contact Technical Support.
171	Bad format.	Might not be C-ISAM file. Recheck this before contacting

Technical Support.

Adabas Error Messages

CONNX for Adabas error messages are covered in the Adabas documentation CD under "Nucleus Error Messages and Nucleus Response Codes." For more information, contact CONNX Technical Support or your Adabas SQL Gateway (CONNX for Adabas) sales representative.

DB2

DB2: SQL States

This table summarizes potential SQL States error and informational messages that can be returned to CONNX by a DB2 target server.

State	Error Message Text	Recommended Action
00000	SQL statement execution successful.	Informational: No action required.
01002	Disconnect error.	Informational: No action required.
01003	Null values removed from column function argument(s).	Informational: No action required.
01004	The host variable string value was truncated.	Informational: No action required.
01503	The result column count is greater than the number of host variables.	Informational: No action required.
01504	UPDATE or DELETE without a WHERE clause.	Informational: No action required.
01506	An arithmetic error in the date portion of a DATE or TIMESTAMP was corrected.	Informational: No action required.
01509	Cursor blocking disabled due to insufficient storage.	If this message recurs, notify your DBA.
01515	Null value has been assigned to a host variable.	Informational: No action required.
01517	Substitute character used for a character which could not be converted.	Informational: No action required.
01519	Numeric value out of range: null value assigned to host variable.	Informational: No action required.
01520	Host variable cannot be assigned a string value.	Revise SQL statement to assign a compatible data type to the host variable.
01524	Column function result omits null arithmetic values.	Informational: No action required.
01526	Isolation level escalated.	Informational: No action required.
01539	Connection successful; use only single byte character set (SBCS) characters.	Informational: No action required.
01543	Duplicate constraint ignored.	Informational: No action required.
01545	A one-part column name was processed as a correlated reference.	Informational: No action required.
01550	The index create failed because the index already exists.	Informational: No action required.
01564	Division by zero: null value assigned to a host variable.	Revise SQL statement and retry as required.
01589	Redundant statement specifications ignored.	Informational: No action required.
01595	Invalidated view definition	Informational: No action required.

	replaced.	
01596	Long string data type-based comparison functions were not created.	Informational: No action required.
01599	Bind/REBIND options ignored.	Informational: No action required.
01602	Optimization level reduced.	Informational: No action required.
01604	SQL statement explained.	Informational: No action required.
01607	Read-only transaction time exceeds the defined threshold.	Notify your DBA.
01609	Stored procedure returned too many result sets.	Informational: No action required.
01610	Stored procedure returned ad-hoc result set(s).	Informational: No action required.
01611	Previously closed cursor reopened on the next result set.	Informational: No action required.
01615	Bind option ignored.	Informational: No action required.
01616	CPU resource limit exceeded.	Resubmit SQL request; notify your DBA if this message recurs.
01622	System error after successful statement completion.	Informational: No action required.
01632	Number of entitled concurrent connections exceeded.	Notify your DBA.
02000	End of result set.	Informational: No action required.
07001	Host variable count is not equal to number of parameter markers.	Revise SQL statement and/or bound parameters and retry.
07002	Invalid parameter list or control block.	Revise SQL statement and/or bound parameters and retry.
07003	An EXECUTE of a SELECT failed, or the statement is not in prepared state,	Contact CONNX Tech Support.
07005	Statement cursor name does not match any prepared statement.	Contact CONNX Tech Support.
07006	Input host variable unused due to invalid data type.	Revise SQL statement and/or bound parameters and retry.
08001	DRDA AR (application requester) could not connect to the target.	Verify connection parameters; retry; contact CONNX Tech Support.
08002	Connection already exists.	Verify connection parameters; retry; contact CONNX Tech Support.
08003	Connection does not exist.	Verify connection parameters; retry; contact CONNX Tech Support.
08004	DRDA AS (application server) rejected the connection request.	Verify connection parameters; retry; contact CONNX Tech Support.
09000	Triggered SQL statement failed.	Notify your DBA.
0A001	Invalid CONNECT statement: process not in the connectable state.	Verify connection parameters; retry; Contact CONNX Tech Support.
0A501	The connect attempt failed because the DRDA application	Verify connection parameters; retry; contact CONNX Tech Support.

	server security mechanism is not supported by the DRDA application requester (CONNX for DB2).	
0A502	Action/operation disabled.	Notify your DBA; contact CONNX Tech Support.
21000	Result contains more than one row or subquery result contains more than one value.	Revise SQL statement and retry.
21501	Invalid multi-row INSERT into self-referencing table.	Revise SQL statement and retry.
21502	Invalid multi-row UPDATE of a primary key.	Revise SQL statement and retry.
21504	Invalid multi-row DELETE.	Revise SQL statement and retry.
21505	Row function must not return multiple rows.	Revise SQL statement and retry.
22001	Character data was truncated.	Revise SQL statement and retry.
22003	Numeric value out of range.	Revise numeric value and retry.
22007	Invalid datetime format.	Correct datetime literal or parameter marker value and retry.
22008	Datetime field overflow.	Revise datetime arithmetic function or expression and retry.
22011	Substring error.	Revise SUBSTR function and retry.
22012	Division by zero.	Revise DIVIDE syntax and retry.
22018	Invalid scalar function character value.	Correct character literal and retry.
22019	Invalid escape character in LIKE predicate.	Correct escape character and retry.
22021	A character is undefined for the coded character set.	Remove character from host variable or character literal and retry.
22024	Input host variable or parameter did not contain a NUL terminator.	Resubmit with NULL-terminated host variable.
22025	Invalid escape character in LIKE predicate.	Correct escape character and retry.
22501	Invalid variable string length.	Truncate string and retry.
22502	Arithmetic exception error occurred.	Revise arithmetic expression and retry.
22506	Datetime special register reference is invalid.	Remove reference to special register and retry.
22522	Invalid CCSID value.	Contact CONNX Tech Support.
23001	A RESTRICT UPDATE/DELETE rule prevented the UPDATE/DELETE of a parent key.	Revise SQL statement and retry.
23502	Attempt to INSERT/UPDATE NULL into a non-null column.	Remove NULL values from INSERT/UPDATE statement and retry.
23503	Invalid value for foreign key INSERT/UPDATE.	Revise INSERT/UPDATE values and retry.
23504	A NO ACTION	Revise UPDATE/DELETE and retry.

	UPDATE/DELETE rule prevented a parent key UPDATE/DELETE.	
23505	Unique index/constraint violation.	Revise INSERT/UPDATE and retry.
23511	A check constraint prevented the deletion of a parent row.	Revise DELETE statement and retry.
23512	A check constraint add failed: one or more base table rows conflict.	Revise INSERT statement and retry.
23513	A check constraint prevented an INSERT/UPDATE.	Revise INSERT/UPDATE and retry.
23515	Create of unique index/constraint failed because of duplicate key values in base table.	Revise CREATE INDEX/CONSTRAINT statement or delete duplicate key values from base table.
24501	The cursor is not open.	Contact CONNX Tech Support.
24502	The cursor is already open.	Contact CONNX Tech Support.
24504	The cursor is not positioned on a row.	Contact CONNX Tech Support.
24506	PREPARE failed: the target statement has an open cursor.	Contact CONNX Tech Support.
24514	Cursor disabled by previous error.	Contact CONNX Tech Support.
24516	Cursor already assigned.	Contact CONNX Tech Support.
25000	Update operation invalid for read-only connection	Connect in read/write mode and retry.
25501	This statement can only begin a unit of work.	End the current unit of work with a COMMIT/ROLLBACK and retry.
26501	Statement is invalid or missing.	Revise SQL statement and retry.
28000	Invalid authorization (user) name.	Re-enter user name in correct case and retry.
2D521	COMMIT/ROLLBACK invalid for this environment.	Probable cause: transaction logging/journalling is not active.
2D528	Dynamic COMMIT invalid for this environment.	Probable cause: transaction logging/journalling is not active.
2D529	Dynamic ROLLBACK invalid for this environment.	Probable cause: transaction logging/journalling is not active.
34000	Invalid cursor name.	Revise SQL syntax and retry; contact CONNX Tech Support.
38001	External function is not allowed to execute SQL.	Notify your DBA.
38002	External function not defined as MODIFIES SQL DATA.	Notify your DBA.
38003	Statement not permitted in a function/procedure.	Notify your DBA.
38004	External function not defined as READS SQL DATA.	Notify your DBA.
38501	Error occurred during call to a function, procedure, or trigger.	Retry; notify your DBA; contact CONNX Tech Support.
38502	The external function is not	Notify your DBA.

	allowed to execute SQL statements.	
38503	User-defined function abnormally terminated.	Notify your DBA.
38504	User-defined function interrupted by the user.	Notify your DBA.
38552	A function in the IBM SYSFUN schema has abnormally terminated.	Notify your DBA.
39001	Invalid SQLSTATE returned from a user-defined function.	Notify your DBA.
39004	Null value invalid for IN / INOUT argument.	Revise SQL statement and/or parameters and retry.
40001	Deadlock / timeout occurred with automatic rollback.	Retry; notify your DBA.
40003	Statement completion is unknown.	Retry; notify your DBA.
40504	Unit of work rolled back due to a system error.	Retry; notify your DBA.
40506	Current transaction rolled back due to an SQL error.	Revise SQL statement; retry.
42501	Your user ID does not have the privilege to perform the specified operation on the identified object.	Request the necessary privilege from your DBA.
42502	You are not authorized to perform the specified operation.	Request the necessary privilege from your DBA.
42505	The application server failed to authenticate the end user during connection processing.	Verify userid/password and retry.
42509	SQL statement not authorized due to STATICRULES option.	Notify your DBA.
42601	Invalid/missing character, token, or clause.	Revise SQL syntax and retry.
42602	Invalid character detected in a name.	Revise SQL name and retry.
42603	Unterminated string constant.	Revise SQL string constant and retry.
42604	Invalid numeric or string constant.	Revise constant and retry.
42605	Invalid number of scalar function arguments.	Revise SQL syntax and retry.
42606	Invalid hexadecimal constant.	Revise SQL syntax and retry.
42607	Invalid column function operand.	Revise SQL syntax and retry.
42608	Invalid use of NULL or DEFAULT in VALUES clause.	Revise VALUES clause SQL syntax and retry.
42609	All operands are parameter markers.	Revise SQL syntax and retry.
42610	Parameter marker not allowed.	Revise SQL syntax and retry.
42611	Invalid column/argument definition.	Revise SQL syntax and retry.

42612	SQL statement is not acceptable in the current context.	Revise SQL syntax and retry.
42613	Mutually exclusive clauses.	Remove one or more clauses and retry.
42614	Duplicate keyword.	Remove duplicate keyword and retry.
42615	Invalid alternative.	Revise SQL syntax and retry.
42617	Blank or empty SQL statement.	Revise SQL syntax and retry.
42618	Host variable not allowed.	Revise SQL syntax and retry.
42622	Name/label is too long.	Revise SQL name/label and retry.
42701	Duplicate column name in INSERT/UPDATE.	Revise SQL syntax and retry.
42702	Duplicate or ambiguous column reference.	Revise SQL syntax and retry.
42703	Undefined column, attribute, or parameter name.	Revise SQL syntax and retry.
42704	Undefined object or constraint name.	Revise SQL syntax and retry.
42707	Column name in ORDER BY is not in the SELECT clause.	Add column name to SELECT, or remove it from the ORDER BY; retry.
42710	Duplicate object or constraint name.	Revise CREATE TABLE/INDEX name and retry.
42711	Duplicate column/attribute name in object definition or ALTER statement.	Revise CREATE TABLE/INDEX column name and retry.
42712	Duplicate table designator in the FROM clause.	Revise SQL syntax and retry.
42713	Duplicate object in object list.	Revise SQL syntax and retry.
42723	Duplicate function signature exists in the schema.	Revise CREATE PROCEDURE syntax and retry.
42724	Unable to access an external program.	Verify program/procedure name; retry; notify your DBA.
42727	No default primary tablespace exists for the new table.	Notify your DBA.
42728	Duplicate node in the nodegroup.	Notify your DBA.
42729	Undefined node.	Notify your DBA.
42730	Container name in use by another tablespace.	Notify your DBA.
42731	Container name in use by this tablespace.	Notify your DBA.
42742	Subtable/view already exists in the typed table/view hierarchy.	Revise CREATE TABLE/VIEW syntax and retry.
42802	Number of INSERT/UPDATE values does not match the number of columns.	Revise VALUES clause and retry.
42803	Invalid column reference in SELECT/HAVING clause.	Revise SELECT/HAVING; retry.
42805	An integer in the ORDER BY clause does not map to a result column.	Correct ORDER BY integer; retry.

42806	Invalid host variable assignment: incompatible data types.	Revise host variable to compatible data type; retry.
42807	INSERT/UPDATE/DELETE not permitted on this object.	Request privilege from your DBA.
42808	Column in the INSERT/UPDATE statement is not updateable.	Remove column from statement; retry.
42809	Statement cannot be applied to/executed on the identified object.	No action required.
42811	Number of columns does not match the number of columns in the SELECT.	Correct SQL syntax; retry.
42815	Invalid data type, length, scale, value, or CCSID.	Correct SQL syntax; retry.
42816	Invalid datetime value or duration.	Correct datetime literal; retry.
42818	Incompatible operator/function operands.	Correct SQL syntax; retry.
42819	Non-numeric operand in arithmetic operation or function.	Correct operand; retry.
42820	Invalid numeric constant.	Correct constant; retry.
42821	UPDATE/INSERT value is incompatible with the target column.	Revise UPDATE/INSERT VALUE to a compatible data type; retry.
42823	Subquery SELECT clause contains multiple columns.	Rewrite subquery to contain 1 column; retry.
42824	LIKE operand is not a character string.	Revise operand; retry.
42827	The target table of the UPDATE/DELETE does not match the target table of the WHERE CURRENT OF cursor.	Revise SELECT statement to point to target table.
42828	The target table of the UPDATE/DELETE WHERE CURRENT OF statement is read-only, or the cursor is read-only.	Revise SELECT statement to include FOR UPDATE OF; request read/write access to target table.
42829	FOR UPDATE OF is invalid, because the cursor result table is read-only.	Revise SELECT statement to include FOR UPDATE OF.
42832	This operation is not permitted on system objects.	Remove system object(s) from SQL statement; retry.
42854	Data type mismatch between a select list result column and a typed view or summary table.	Revise SELECT column list; retry.
42877	Column name must be unqualified.	Revise SQL column name syntax; retry.
42878	Invalid EXTERNAL function/procedure name.	Revise SQL syntax; retry.
42882	Specific instance qualifier must match the function name qualifier.	Notify your DBA; contact CONNX Tech Support.
42883	No function found with matching	Notify your DBA; contact CONNX Tech

	signature.	Support.
42884	No function/procedure found which matched the name and/or arguments.	Notify your DBA; contact CONNX Tech Support.
42889	Target table already has a primary key.	Revise CREATE syntax; retry.
42893	Object/constraint not dropped: dependent objects exist.	Notify your DBA.
42895	Input host variable data type does not match the parameter of a procedure or user-defined function.	Revise parameter value and/or type; retry.
42901	Column function does not include column name.	Revise SQL syntax; retry.
42903	Invalid reference in a WHERE / SET clause.	Revise SQL syntax; retry.
42907	String is too long.	Revise string constant; retry.
42908	Required column list not included in SQL statement.	Revise SQL SELECT column list; retry.
42911	Invalid decimal divide: result scale is negative.	Revise DIVIDE syntax; retry.
42912	Cannot update column: it is not in the FOR UPDATE OF clause of the SELECT statement.	Add column to FOR UPDATE OF clause; retry.
42917	Cannot explicitly drop object.	Notify your DBA.
42939	Cannot create object: identifier reserved for system use.	Revise CREATE syntax and object name; retry.
42969	Package not created and current unit of work rolled back due to internal limitations or invalid section number.	Contact CONNX Tech Support.
51002	Package not found.	Ask your DBA to BIND the CONNX SQL packages; retry; contact CONNX Tech Support.
51003	Package consistency tokens do not match.	Ask your DBA to BIND the CONNX SQL packages; retry; contact CONNX Tech Support.
51005	Function disabled by previous system error.	Notify your DBA.
51015	Attempt to execute an SQL package section which returned a bind time error.	Contact CONNX Tech Support.
51017	User not logged on.	Retry logon; contact CONNX Tech Support.
51021	Cannot execute SQL statements until application executes a rollback.	Execute rollback; retry.
51028	SQL package marked inoperative; cannot be used.	Ask your DBA to BIND the CONNX SQL packages; retry; contact CONNX Tech Support.
54001	Statement too long or complex.	Simplify SQL statement; retry.

54002	String constant too long.	Truncate string constant; retry.
54004	Too many table names / items in SELECT / INSERT.	Revise SQL syntax; retry.
54006	Concatenation result too long.	Revise concatenation operands; retry.
54008	Key too long or has too many columns.	Revise CREATE TABLE/INDEX syntax; retry.
54010	Table record length too long.	Revise CREATE TABLE/INDEX syntax; retry.
54011	Too many columns specified for table / view.	Revise CREATE TABLE/INDEX syntax; retry.
54032	Maximum table size has been reached.	Notify your DBA.
54047	Exceeded maximum table space size.	Notify your DBA.
54048	Temporary table space with sufficient page size does not exist.	Notify your DBA.
55001	Database must be migrated.	Notify your DBA.
55002	Explanation table not defined properly.	Notify your DBA.
55006	Cannot drop object: currently in use by the same application process.	Revise application logic; retry; contact CONNX Tech Support
55019	Table is in an invalid state for the operation.	Table is not logged/journalled; retry with No Commit isolation level or ask your DBA to start logging/journalling for the table.
55025	Database must be restarted.	Notify your DBA.
55039	Access / state transition not allowed: tablespace not in an appropriate state.	Notify your DBA.
56033	Insert / update long string column value must be a host variable or NULL.	Revise SQL syntax to use a parameter marker.
56084	DRDA does not support LOB data.	Contact CONNX Tech Support.
56092	Cannot determine authorization type: authorization name is both a user id and group id.	Notify your DBA.
56095	Invalid bind option; bind operation terminated.	Contact CONNX Tech Support.
56096	Conflicting bind options; bind operation terminated.	Contact CONNX Tech Support.
56098	Error during implicit rebind/prepare.	Contact CONNX Tech Support.
56099	Target database does not support REAL data type.	Revise SQL syntax; retry.
57001	Table is unavailable: no primary index defined.	Notify your DBA.
57007	Cannot use object: DROP / ALTER pending.	Retry; notify your DBA.

57009	Virtual storage / resource temporarily unavailable.	Retry; notify your DBA.
57011	Virtual storage / resource unavailable.	Retry; notify your DBA.
57012	Non-database resource unavailable; SQL statements can be successfully executed.	Retry; notify your DBA.
57013	Non-database resource unavailable; SQL statements cannot be successfully executed.	Retry; notify your DBA.
57014	Processing canceled as requested.	No action required.
57016	Table is inactive: it cannot be accessed.	Retry; notify your DBA.
57017	Character conversion undefined.	Contact CONNX Tech Support.
57019	The statement was unsuccessful, because of a problem with a resource.	Retry; notify your DBA.
57022	Table could not be created: statement authorization ID does not own any suitable dbspaces.	Notify your DBA.
57030	Connection attempt to application server exceeds the installation-defined limit.	Notify your DBA.
57033	Deadlock / timeout without automatic rollback.	Retry; notify your DBA.
57046	Cannot start new transaction: database or instance is quiesced.	Notify your DBA.
57047	Cannot create an internal database file: the directory is not accessible.	Notify your DBA.
57049	Operating system process limit reached.	Notify your DBA.
57051	CPU cost estimate exceeds resource limit.	Notify your DBA.
57055	Temporary table space with sufficient page size unavailable.	Notify your DBA.
57056	Package unavailable: database is in NO PACKAGE LOCK mode.	Notify your DBA.
57057	A prior condition in a DRDA chain of SQL statements prevented execution of the SQL statement.	Contact CONNX Tech Support.
58004	System error occurred: SQL statements can be successfully executed.	No action required.
58005	System error occurred: SQL statements cannot be successfully executed.	Notify your DBA; contact CONNX Tech Support.
58008	Execution failed: distributed protocol error will not prevent successful execution of Distributed Data Management	Contact CONNX Tech Support.

	(DDM) commands or SQL statements.	
58009	Execution failed: distributed protocol error caused deallocation of the conversation.	Contact CONNX Tech Support.
58010	Execution failed: distributed protocol error will affect successful execution of Distributed Data Management (DDM) commands or SQL statements.	Contact CONNX Tech Support.
58011	The Distributed Data Management (DDM) command is invalid while the bind process in progress.	Contact CONNX Tech Support.
58012	The bind process with the specified package name and consistency token is not active.	Contact CONNX Tech Support.
58014	The Distributed Data Management (DDM) command is not supported.	Contact CONNX Tech Support.
58015	The Distributed Data Management (DDM) object is not supported.	Contact CONNX Tech Support.
58016	The Distributed Data Management (DDM) parameter is not supported.	Contact CONNX Tech Support.
58017	The Distributed Data Management (DDM) parameter value is not supported.	Contact CONNX Tech Support.
58018	The Distributed Data Management (DDM) reply message is not supported.	Contact CONNX Tech Support.
58023	System error: current program has been canceled.	Notify your DBA.
58028	Commit requested, but the unit of work was rolled back.	Notify your DBA.
58030	I/O error occurred.	Notify your DBA.
58031	System error: connection unsuccessful.	Notify your DBA; contact CONNX Tech Support.

DB2: Distributed Data Management

This table summarizes potential Distributed Data Management (DDM) error and informational messages that can be returned to CONNX by a DB2 target server.

Message Code	Possible Problem	Solution
ABNUOWRM	An Abnormal End Unit of Work Condition Reply Message was received from the target server. The current logical unit of work ended abnormally because of some action at the target server. This can be caused by a deadlock resolution, by an operator intervention, or by some similar situation that caused the relational database (RDB) to rollback the current logical unit of work.	Retry the operation.
ACCATHRM	A Not Authorized to Use Access Method Reply Message was received from the target server. The user is not authorized to use the specified access method.	Obtain the required authority from the DBA.
ACCRDBRM	An Access to RDB Completed Reply Message was received from the target server. An instance of the SQL application manager has been created and is bound to the specified relational database (RDB).	Informational: No action required.
AGNPRMRM	A Permanent Agent Error Reply Message was received from the target server. The command requested could not be completed because of a permanent error condition detected at the target system.	Notify your DBA; retry the operation; contact CONNX Tech Support.
BGNBNDRM	A Begin Bind Error Reply Message was received from the target server, which indicates that the package binding process could not be initiated because an error condition exists.	Contact CONNX Tech Support.
CMDATHRM	A Not Authorized to Command Reply Message was received from the target server. The user is not authorized to perform the requested command.	Request authority to the command from your DBA or systems administrator.
CMDCHKRM	A Command Check Reply Message was received from the target server. The requested command encountered an unarchitected and implementation-specific condition for which there is no architected message.	Contact CONNX Tech Support.
CMDCMPRM	A Command Processing Completed Reply Message was received from the target server. The command processing was successfully completed.	Informational: No action required.
CMDNSPRM	A Command Not Supported Reply Message was received from the target server. The specified command is not recognized or is not supported for the specified target object.	Contact CONNX Tech Support.
CMDVLTRM	A Command Violation Reply Message	Contact CONNX Tech Support.

	was received from the target server, which indicates that a Distributed Data Management (DDM) command violating the processing capabilities of the conversation has been received.	
CMMRQSRM	A Commitment Request Reply Message was received from the target server, which indicates that a dynamic commit or rollback was attempted at the target relational database.	Informational: No action required.
DSCINVRM	An Invalid Description Reply Message was received from the target server. A target server manager was unable to assemble a valid Formatted Data Object Content Architecture (FD:OCA) descriptor for the data being sent.	Contact CONNX Tech Support.
DTAMAPRM	A Data Mapping Error Reply Message was received from the target server. The target server cannot insert, modify, or retrieve a record due to a data mapping error.	Contact CONNX Tech Support.
DTAMCHRM	A Data Descriptor Mismatch Reply Message was received from the target server, which indicates that the data received did not match the received descriptor. That is, the amount of data received did not match the amount of data expected.	Contact CONNX Tech Support.
ENDQRYRM	The End of Query Reply Message indicates that the query process has terminated in such a manner that the query is now closed. It cannot be resumed with the CNTQRY command or closed with the CLSQRY command.	Informational: No action required.
ENDUOWRM	The End Unit of Work Condition Reply Message specifies that the logical unit of work has ended as a result of the last command.	Informational: No action required.
INTTKNRM	An Interrupt Token Invalid Reply Message was received from the target server. The target SQL Application Manager (SQLAM) has determined that the specified RDB Interrupt Token (RDBINTTKN) value is invalid.	Contact CONNX Tech Support.
INVRQSRM	An Invalid Request Reply Message was received from the target server.	Contact CONNX Tech Support.
MGMATHRM	A Not Authorized to Management Class Reply Message was received from the target server. The requester (CONNX) is not authorized to the named management class on the target system.	Contact CONNX Tech Support.
MGMCNFRM	A Management Class Conflict Reply Message was received from the target server. The management class specified on the CRTAIF command conflicts with the management class of the base file.	Contact CONNX Tech Support.
MGMNFNRM	A Management Class Not Found Reply Message was received from the target server. The named	Contact CONNX Tech Support.

	management class cannot be found on the target system.	
MGRDEPRM	A Manager Dependency Error Reply Message was received from the target server. A request has been made to use a manager, but the requested manager requires specific support from some other manager that is not present.	Contact CONNX Tech Support.
MGRLVLRM	A Manager Level Conflict Reply Message was received from the target server. The manager levels specified in the Manager Level List (MGRLVLLS) conflict among themselves or with previously specified manager levels.	Contact CONNX Tech Support.
OBJNSPRM	An Object Not Supported Reply Message was received from the target server. The target server does not recognize or support the object specified as data in an OBJDSS for the command associated with the object.	Contact CONNX Tech Support.
OPNQFLRM	An Open Query Failure Reply Message was received from the target server, which indicates that the OPNQRY command failed to open the query.	Contact CONNX Tech Support.
OPNQRYRM	Open Query Complete Reply Message indicates to the requester that the OPNQRY command completed normally, and that the query process has been initiated.	Informational: No action required.
PKGBNARM	An RDB Package Binding Not Active Reply Message was received from the target server, which indicates that a Bind SQL Statement (BNDSQLSTT) or an End Bind (ENDBND) command was issued when the package binding process was not active for the specified package name.	Contact CONNX Tech Support.
PKGBPARM	An RDB Package Binding Process Active Reply Message was received from the target server. The command cannot be issued when the relational database package binding process is active. The active package binding process must be terminated before the command can be issued.	Contact CONNX Tech Support.
PRCCNVRM	A Conversational Protocol Error Reply Message was received from the target server.	Contact CONNX Tech Support.
PRMNSPRM	A Parameter Not Supported Reply Message was received from the target server. The specified parameter is not recognized or is not supported for the specified command.	Contact CONNX Tech Support.
QRYNOPRM	A Query Not Open Reply Message was received from the target server. A Continue Query (CNTQRY) or a Close Query (CLSQRY) command was	Contact CONNX Tech Support.

	issued for a query that is not open.	
QRYPOPRM	A Query Previously Opened Reply Message was received from the target server. The server sends this message when an Open Query (OPNQRY) command is issued for a query that is already open. A previous OPNQRY command might have opened the query which may not be closed.	Contact CONNX Tech Support.
RDBACCRM	An RDB Currently Accessed Reply Message was received from the target server, which indicates that the Access Relational Database (ACCRDB) and the Interrupt Relational Database Request (INTRDBRQS) commands cannot be issued because the requester (CONNX) currently has access to a relational database.	Contact CONNX Tech Support.
RDBAFLRM	An RDB Access Failed Reply Message was received from the target server, which specifies that the relational database (RDB) failed the attempted connection.	Verify that UserID, Password, and RDBNAME are correctly entered; retry; Contact CONNX Tech Support.
RDBATHRM	A Not Authorized to RDB Reply Message was received from the target server. The requester is not authorized to access the specified relational database.	Verify that UserID, Password, and RDBNAME are correctly entered; retry; Contact CONNX Tech Support.
RDBNACRM	An RDB Not Accessed Reply Message was received from the target server, which indicates that the access relational database command (ACCRDB) was not issued prior to a command requesting RDB services.	Contact CONNX Tech Support.
RDBNFNRM	An RDB Not Found Reply Message was received, which indicates that the target server cannot find the specified relational database.	Verify that the RDBNAME entered in the CONNX CDD entry field matches the RDBNAME or location defined on the host; retry; Contact CONNX Tech Support.
RDBUPDRM	RDB Update Reply Message indicates that a DDM command resulted in an update at the target relational database (RDB).	Informational: No action required.
RSCLMTRM	A Resource Limits Reached Reply Message was received from the target server. The requested command could not be completed due to insufficient target server resources.	Notify your DBA or systems administrator; contact CONNX Tech Support.
SQLERRRM	An SQL Error Condition Reply Message was received from the target server, which indicates that an SQL error has occurred.	Informational: Refer to the SQL State/Code for specifics.
SYCMMGNM	A System Command Manager Name message was received from the target server.	Contact CONNX Tech Support.
SYNTAXRM	A Data Stream Syntax Error Reply Message was received from the target server. The data sent to the target agent does not structurally conform to the requirements of Distributed Data Management (DDM) Architecture.	Contact CONNX Tech Support.

SYSCMDRM	A System Command Reply Message was received from the target server.	Contact CONNX Tech Support.
TRGNSPRM	A Target Not Supported Reply Message was received from the target server. The object specified as a command target parameter is not an object of a class that the target server supports.	Contact CONNX Tech Support.
VALNSPRM	A Parameter Value Not Supported Reply Message was received from the target server. The parameter value specified either is not recognized or is not supported for the specified parameter.	Contact CONNX Tech Support.

DB2: APPC Primary Return Codes

This table lists potential APPC Primary Return Codes returned by the CONNX DB2 Module when connected via supported SNA APPC/LU 6.2 vendor software.

APPC Primary Return Code	Message Text	Recommended Action
OK	OK	None Required
Allocation Error	The attempt to allocate an APPC conversation failed.	Refer to secondary return code; retry; notify CONNX Tech Support.
Canceled	The CONNX DB2 local server issued one of the following verbs: DEALLOCATE, SEND_ERROR or TP_ENDED.	None Required; restart connection as needed.
Communications Subsystem Abended	The APPC communications software running on the local client machine ended abnormally, or the connection from the client to the server ended due to a LAN error.	Restart the vendor SNA engine; retry connection.
Communications Subsystem Not Loaded	The APPC communications software on the local client machine could not be loaded (executed) or has ended. Notify the system administrator.	Usually a registry or PATH problem; check vendor SNA engine PATH/registry entries and retry.
Conversation Busy	Only one APPC verb can be processed at a time by each conversation.	Retry; notify CONNX Tech Support.
Conversation Ended	---	None Required
Conversation Style Mixed	---	Notify CONNX Tech Support.
Conversation Type Mixed	The CONNX DB2 local server has issued both basic and mapped conversation verbs. Only one type of verb is permitted per conversation.	Notify CONNX Tech Support.
CNOS Local Race Reject	The APPC engine is currently processing a Change Number of Sessions (CNOS) request from a local Logical Unit (LU).	Retry after a short wait.
CNOS Partner Logical Unit Reject	The partner Logical Unit rejected a Change Number of Sessions (CNOS) request from the local Logical Unit.	Refer to secondary return code.
Conversation Failure No Retry	The APPC conversation with the partner Transaction Program (the DRDA Application Server) ended due to a permanent error. Notify the system administrator. Do not retry the conversation.	Notify DBA and CONNX Tech Support.
Conversation Failure Retry	The APPC conversation with the partner Transaction Program (the DRDA Application Server) ended due to a temporary error. Restart the connection.	Retry; notify DBA and CONNX Tech Support.
Deallocate Abend	The conversation has been deallocated.	Issued by the host DRDA program; retry; notify DBA and CONNX Tech Support.
Deallocate Abend Program	The APPC conversation has been deallocated.	Issued by the host DRDA program; retry; notify DBA and CONNX Tech Support.
Deallocate Abend Service	The APPC basic conversation was deallocated by the partner Transaction Program (the DRDA Application Server) with dealloc_type set to AP_ABEND_SVC.	Issued by the host DRDA program; retry; notify DBA and CONNX Tech Support.
Deallocate Abend Timer	The APPC basic conversation was deallocated by the partner Transaction Program (the DRDA	Issued by the host DRDA program; retry; notify DBA and CONNX Tech Support.

	Application Server) with dealloc_type set to AP_ABEND_TIMER.	
Deallocate Normal	The partner Transaction Program (the DRDA Application Server) has deallocated the conversation.	None Required
Invalid APPC verb	---	Notify CONNX Tech Support.
Invalid Verb Segment	The APPC Verb Control Block (VCB) extended beyond the data segment boundary.	Notify CONNX Tech Support.
Parameter Check	The APPC verb failed because of a parameter error.	Refer to secondary return code; correct the invalid parameter.
Program Error No Truncation	The partner Transaction Program (the DRDA Application Server) has issued a SEND_ERROR while the conversation was in SEND state. No data was lost.	Retry; notify CONNX Tech Support.
Program Error Purging	---	Retry; notify CONNX Tech Support.
Program Error Truncation	The partner Transaction Program (the DRDA Application Server) issued a SEND_ERROR after sending a partial logical record. Data was lost.	Retry; notify CONNX Tech Support.
State Check	The APPC verb failed because it was invoked from an invalid state.	Refer to secondary return code; retry; notify CONNX Tech Support.
Service Program Error No Truncation	The partner Transaction Program (the DRDA Application Server) or the partner LU issued SEND_ERROR. No data was lost.	Retry; notify CONNX Tech Support.
Service Program Error Purging	The partner Transaction Program (the DRDA Application Server) issued a SEND_ERROR. Data was lost.	Retry; notify CONNX Tech Support.
Service Program Error Truncation	The partner Transaction Program (the DRDA Application Server) issued a SEND_ERROR after sending a partial logical record. Data was lost.	Retry; notify CONNX Tech Support.
Thread Blocking	The calling thread is already blocked.	Retry; notify CONNX Tech Support.
Transaction Program Busy	The CONNX DB2 local server issued a call to the APPC engine while it was processing another call for the same transaction program.	None Required; retry.
Unsuccessful	No data was returned from a RECEIVE_IMMEDIATE request to the partner Transaction Program (the DRDA Application Server).	None Required; retry.

DB2: APPC Secondary Return Codes

This table lists potential APPC Secondary Return Codes returned by the CONNX DB2 Module when connected via supported SNA APPC/LU 6.2 vendor software.

APPC Secondary Return Code	Message Text	Recommended Action
Allocation Failure No Retry	The APPC conversation was not allocated due to a permanent error. Notify the system administrator.	Local and/or target machine configuration problem; notify CONNX Tech Support.
Allocation Failure Retry	The APPC conversation was not allocated due to a temporary error. Retry the connection.	Possible transient link problem; retry; notify CONNX Tech Support.
Invalid APPC conversation identifier	---	Connection failure; retry connection; notify CONNX Tech Support.
Bad conversation type	The value specified for parameter conv_type in the APPC ALLOCATE verb control block is invalid.	Notify CONNX Tech Support.
Bad Logical Length	The logical record length field is invalid.	Notify CONNX Tech Support.
Bad LU Alias	The logical unit alias is not defined in the APPC configuration.	Verify configuration LU Alias and CONNX CDD/Data source entry; retry; notify CONNX Tech Support.
Bad Partner LU Alias	Check the CONNX data source entry and the APPC configuration value, and retry the connection.	Notify CONNX Tech Support.
Bad Return control	The value specified for rtn_ctl in the APPC ALLOCATE verb control block is invalid.	Notify CONNX Tech Support.
Bad Return Status with Data	The return status parameter (rtn_status) of the APPC verb control block is invalid.	Notify CONNX Tech Support.
Bad Security Value	The value specified for security in the APPC ALLOCATE verb control block is invalid.	Notify CONNX Tech Support.
Bad Sync Level	The value specified for sync_level in the APPC ALLOCATE verb control block is invalid.	Notify CONNX Tech Support.
Bad Transaction Program identifier.	---	Connection failure: retry connection; notify CONNX Tech Support.
Bad type	---	Notify CONNX Tech Support.
Conversation Type Mismatch	The DRDA AS (application server) transaction program at the target host does not support the conversation_type parameter of the APPC allocate verb control block.	Notify CONNX Tech Support.
Dealloc Bad Type	The dealloc_type parameter in the APPC DEALLOCATE verb control block is invalid.	Notify CONNX Tech Support.
Dealloc Confirm Bad State	The conversation was not in SEND state, and the TP attempted to flush the send buffer and send a confirmation request.	Notify CONNX Tech Support.
Dealloc Flush Bad State	The dealloc_type parameter in the APPC DEALLOCATE verb control block = AP_FLUSH, but the program is not in	Notify CONNX Tech Support.

	SEND state.	
Dealloc Not LL BDY	The conversation was in SEND state, but the TP did not send a complete logical record.	Notify CONNX Tech Support.
Destination Address Equals Own Address	---	Configuration error. Verify that the local adapter address does not equal any destination adapter address.
DLC (Data Link Control) failure	---	Consult vendor error log(s).
DLC not defined	---	Configuration error: define Data Link Control profile for adapter; notify CONNX Tech Support.
Flush Not Send State	The conversation is not in SEND state.	Notify CONNX Tech Support.
Invalid Alias	--	Possible mismatch between CONNX CDD/data source and APPC configuration.
Invalid Auto-Activate for number of sessions	--	Notify CONNX Tech Support.
Invalid Conversation Security Requested	--	Notify CONNX Tech Support.
Invalid Conversation Security Verification	--	Notify CONNX Tech Support.
Invalid Conversation Type	--	Notify CONNX Tech Support.
Invalid Control Point Name	--	Configuration error; verify Control Point name.
Invalid Control Point NAU (Network Address Unit) Address	---	Configuration error. Verify that APPC configuration entry for NAU is between 0 and 254.
Invalid Destination Address Length	---	Configuration error. Verify host destination address and check local APPC configuration.
Invalid Fully Qualified Control Point Name	---	Possible configuration error. Check APPC configuration against CONNX CDD/Data source; notify CONNX Tech Support.
Invalid Fully Qualified Owning Control Point Name	---	Configuration error. Check APPC configuration; notify CONNX Tech Support.
Invalid Fully Qualified Logical Unit Name	---	Configuration error. Possible mismatch between CONNX CDD/data source entry and APPC configuration.
Invalid Fully Qualified Partner LU Name	---	Configuration error. Possible mismatch between CONNX CDD/data source entry and APPC configuration.
Invalid LU Name	---	Configuration error. Check the LU name(s) defined in the APPC configuration against the CONNX CDD/data source entries.
Invalid LU NAU (Network Address Unit) Address	---	APPC configuration error; enter correct Network Address Unit; contact Network Administrator.

Invalid Mode Name	---	Configuration error. CONNX CDD/data source entry does not match any configured APPC mode name, or mode is undefined on the target host.
Invalid Partner LU Name	---	Configuration error. Check the Partner LU name(s) defined in the APPC configuration against the CONNX CDD/data source entries.
Invalid Password	---	Possible typo; re-enter password (use correct upper/lower case for DB2 UDB targets) and retry connection.
Invalid Session ID	---	Notify CONNX Tech Support.
Invalid Sync Level	---	Notify CONNX Tech Support.
Invalid Transaction Program Name	--	Configuration Error. If TP name is non-blank, check CONNX CDD against APPC configuration; ask Network Administrator to verify that the host TP name is defined; notify CONNX Tech Support.
Invalid User ID	---	Possible typo; re-enter user id (use correct upper/lower case for DB2 UDB targets) and retry connection.
Link deactivation is in progress	---	None required.
Local LU Name equals Partner LU name	---	Change CONNX CDD/data source entries and APPC configuration for Local LU.
Local LU is detached	--	None required.
Minimum Greater than Total	The sum of the APPC CNOS verb parameters min_conwinners_source and min_conwinners_target is greater than the partner_lu_mode_session_limit parameter.	Revise APPC configuration.
Mode Closed	The APPC mode name has been disabled at the host remote LU by setting the local maximum session limit to zero.	Notify DBA/Network Administrator; revise host mode definition; notify CONNX Tech Support.
Mode Name Reject	The APPC CNOS verb failed because the partner LU does not recognize the mode name.	Check local APPC configuration and CONNX CDD/data source entries; define mode name on host as required.
Prepare To Receive Invalid Type	The ptr_type parameter of the APPC prepare_to_receive verb control block is invalid.	Notify CONNX Tech Support.
Prepare To Receive Not LL BDY	The CONNX DB2 local server did not send a complete logical record.	Notify CONNX Tech Support.
Prepare To Receive Not Send State	The conversation is not in SEND state.	Notify CONNX Tech Support.
Partner LU name equals Local LU Name	---	Configuration Error. Revise Local/Partner LU names.
Receive and Wait Bad Fill	The fill parameter of the APPC RECEIVE_AND_WAIT verb is invalid.	Notify CONNX Tech Support.

Receive and Wait Bad State	The CONNX DB2 local server issued an APPC RECEIVE_AND_WAIT verb, but the APPC conversation was not in RECEIVE or SEND state.	Notify CONNX Tech Support.
RCV_AND_WAIT_NOT_LL_BDY	The conversation was in SEND state, and the CONNX DB2 local server sent an incomplete logical record.	Notify CONNX Tech Support.
Receive Immediate Bad Fill	The fill parameter of the APPC RECEIVE_IMMEDIATE verb is invalid.	Notify CONNX Tech Support.
Receive Immediate Bad State	The CONNX DB2 local server issued an APPC RECEIVE_IMMEDIATE verb, but the APPC conversation was not in RECEIVE state.	Notify CONNX Tech Support.
Request To Send Bad State	The CONNX DB2 local server issued an APPC REQUEST_TO_SEND verb from an invalid state.	Notify CONNX Tech Support.
Security Not Valid:	The user ID/password combination was rejected by the target host.	Re-enter userid/password. For DB2 UDB targets, be sure to use the correct lower/upper case characters.
Send Data Invalid Type	The type parameter of the APPC SEND_DATA verb control block is invalid.	Notify CONNX Tech Support.
Send Data Not Send State	The CONNX DB2 local server issued an APPC SEND_DATA verb, but the conversation is not in send state.	Notify CONNX Tech Support.
Send Data Not LL BDY	The CONNX DB2 local server sent an incomplete logical record.	Notify CONNX Tech Support.
Sync Level Not Supported	The DRDA AS (application server) transaction program at the target host does not support the sync_level parameter of the APPC allocate verb control block.	Possible configuration error; notify CONNX Tech Support.
Table error	---	Configuration error. The APPC vendor ASCII/EBCDIC translation table is not installed.
Transaction Program Name Not Recognized	The remote Logical Unit does not recognize the TP name. Check the TP Name entered in the CONNX data source configuration.	Configuration error.
Transaction Program Not Available; No Retry	The DRDA AS (application server) transaction program at the target host is permanently unavailable.	Notify the network/system administrator.
Transaction Program Not Available; Retry	The DRDA AS (application server) transaction program at the target host is temporarily unavailable.	Retry the connection.
Undefined Transaction Program Name	Check the CONNX entry against the APPC configuration.	Configuration Error.
Unknown partner mode name	The value of mode_name in the APPC ALLOCATE verb control block is invalid	Configuration Error: Verify entries for Partner LU alias and/or mode name in APPC configuration against those in CONNX data source/CDD.

DB2: ODBC States

This table summarizes potential ODBC State informational messages that can be returned by the CONNX DB2 Module.

ODBC State	Message Text
00000	Success
01000	General warning
01002	Disconnect error
01003	NULL value eliminated in set function
01004	String data, right truncated
01S07	Fractional truncation
07005	Prepared statement not a cursor-specification
07006	Restricted data type attribute violation
08001	Client unable to establish connection
08002	Connection name in use
08003	Connection does not exist
08004	Server rejected the connection
08007	Connection failure during transaction
08S01	Communication link failure
22001	String data, right truncated
22002	Indicator variable required but not supplied
22003	Numeric value out of range
22007	Invalid datetime format
22008	Datetime field overflow
22012	Division by zero
22018	Invalid character value for cast specification
22019	Invalid escape character
22025	Invalid escape sequence
22026	String data, length mismatch
23000	Integrity constraint violation
24000	Invalid cursor state
25000	Invalid transaction state
25S03	Transaction is rolled back
28000	Invalid authorization specification
34000	Invalid cursor name
40001	Serialization failure
40003	Statement completion unknown
42000	Syntax error or access violation
42S11	Index already exists

42S12	Index not found
44000	WITH CHECK OPTION violation
HY000	General error
HY008	Operation canceled
HY090	Invalid string or buffer length
HY093	Invalid parameter number

VSAM

VSAM: File Open Error Messages

Text	Cause	Action
File Open Error 01: Read access requested, but target file does not have read privileges.	CONNX attempted to read the target file, but the file attributes as defined in CICS RDO (Resource Definition Online) or the FCT (File Control Table) do not permit read access.	Ask the CICS administrator to change the target file attributes to include READ access.
File Open Error 02: Read/Write access requested, but target file does not have add/delete/update privileges.	CONNX attempted to open the target file for update processing, but the file attributes do not permit read, add, delete, or update access.	Either open the target file for read only access, or ask the CICS administrator to change the target file attributes to include read, add, delete, and update access.
File Open Error 03: CICS INQUIRE FILE	CONNX issued a CICS INQUIRE FILE against the target file to determine its OPEN and ENABLED status. The INQUIRE FILE command failed; refer to the reported response and response2 fields for more information.	Use the CEMT INQUIRE FILE command to verify that the CICS file name is defined on the target region. Example: CEMT INQ FI(CUSTOMER) STATUS: RESULTS - OVERTYPE TO MODIFY Fil(CUSTOMER) Vsa Clo Ena Rea Upd Add Bro Del Sha Dsn (CONNX.CUSTOMER) If the file is defined, ask the CICS administrator to authorize the CONNX server transaction (NXS0) and/or the CICS userid to use the CICS INQUIRE FILE command.
File Open Error 04: CICS SET FILE ENABLED	CONNX issued an unsuccessful CICS SET FILE ENABLED command against the target file.	Ask the CICS administrator to verify that the target file exists; also request that the CONNX CICS/VSAM server transaction (NXS0) and/or the CICS userid be authorized to execute the CICS SET FILE command. Refer to the response and response2 codes for this message.
File Open Error 05: InquireFileAttributes()	CONNX issued an unsuccessful CICS INQUIRE FILE command against the target file.	Ask the CICS administrator to verify that the target file exists; also request that the CONNX CICS/VSAM server transaction (NXS0) and/or the CICS userid be

		authorized to execute the CICS INQUIRE FILE command. Refer to the response and response2 codes for this message.
File Open Error 06: CICS SET FILE CLOSED WAIT	CONNX issued an unsuccessful CICS SET FILE CLOSED WAIT command against the target file.	Ask the CICS administrator to verify that the target file exists; also request that the CONNX CICS/VSAM server transaction (NXS0) and/or the CICS userid be authorized to execute the CICS SET FILE command. Refer to the response and response2 codes for this message.
File Open Error 07: CICS SET FILE READABLE	While opening the target file for read-only access, CONNX issued an unsuccessful CICS SET FILE READABLE command.	Ask the CICS administrator to verify that the target file has the readable access attributes. Example: Determine the file attributes via the CEMT INQUIRE FILE command. INQ FI(CUSTOMER) STATUS: RESULTS - OVERTYPE TO MODIFY Fil(CUSTOMER) Vsa Clo Ena Rea Upd Add Bro Del Sha Rea = Readable attribute
File Open Error 08: CICS SET FILE ADDABLE DELETABLE READABLE UPDATABLE	While opening the target file for read/write/update access, CONNX issued an unsuccessful CICS SET FILE ADDABLE DELETABLE READABLE UPDATABLE command.	Ask the CICS administrator to verify that the target file has the addable, deletable, readable, and updatable access attributes. Example: Determine the file attributes via the CEMT INQUIRE FILE command. INQ FI(CUSTOMER) STATUS: RESULTS - OVERTYPE TO MODIFY Fil(CUSTOMER) Vsa Clo Ena Rea Upd Add Bro Del Sha Rea = Read attribute Upd = Update attribute Add = Add attribute Del = Delete attribute
File Open Error 09: CICS SET FILE OPEN	While opening the target file, CONNX issued an unsuccessful CICS SET FILE OPEN command.	Ask the CICS administrator to verify that the target file exists; also request that the CONNX CICS/VSAM server transaction (NXS0) and/or CICS userid be authorized to execute the CICS SET FILE command.

VSAM: File Read Error Messages

Text	Cause	Action
GetFirstRecord() Error 01: Invalid key access method specified	There is a logic error in the CONNX 8.8 CICS/VSAM server program.	Notify CONNX Technical Support.
GetFirstRecord() Error 02: CICS START BROWSE failed.	The CONNX CICS/VSAM server executed an unsuccessful CICS STARTBR command against the target file.	Ask the CICS administrator to verify that the target file has the browseable attribute via the CEMT INQUIRE FILE command: Example: INQ FI(CUSTOMER) STATUS: RESULTS - OVERTYPE TO MODIFY Fil(CUSTOMER) Vsa Clo Ena Rea Upd Add Bro Del Sha Where Bro = the browseable attribute. If the file is browseable and this message recurs, examine the response and response2 return codes for more information. If the file does not have the browseable attribute, ask the CICS administrator to add the browseable attribute to the target file, and retry the request.
GetFirstRecord() Error 03: CICS READNEXT or READPREV failed.	After successfully starting a BROWSE against the target file, CONNX executed an unsuccessful CICS READNEXT or READPREV command.	Examine the response and response2 return codes; report the error to CONNX Technical Support.
GetFirstRecord() Error 04: Invalid READ direction specified.	There is a logic error in the CONNX 8.8 CICS/VSAM server program.	Notify CONNX Technical Support.
GetFirstRecord() Error 05: Invalid operator specified.	There is a logic error in the CONNX 8.8 CICS/VSAM server program.	Notify CONNX Technical Support.
GetFirstRecord() Error 06: A GetByKey or GetByValOffset access method was requested, but the target file is not a VSAM Key Sequenced Data Set (KSDS), Relative Record Data Set (RRSD), or an Entry-Sequenced Data Set	There is a logic error in the CONNX 8.8 CICS/VSAM server program	Notify CONNX Technical Support.

(ESDS).		
CICS DoRead() Error 01: Target VSAM file must be a Key-Sequenced Data Set (KSDS) or a Relative Record Data Set (RRDS).	There is a logic error in the CONNX 8.8 CICS/VSAM server program.	Notify CONNX Technical Support.
CICS DoReadNext() Error 01: m_pRidField class member variable is null.	There is a logic error in the CONNX 8.8 CICS/VSAM server program.	Notify CONNX Technical Support.
CICS DoReadPrev() Error 01: m_pRidField class member variable is null.	There is a logic error in the CONNX 8.8 CICS/VSAM server program.	Notify CONNX Technical Support.

VSAM: File Write/Rewrite/Delete Error Messages

Text	Cause	Action
CICS InsertRecord() Error 01: Target VSAM file must be a Key Sequenced Data Set (KSDS) or a Relative Record Data Set (RRDS), or an Entry-Sequenced Data Set (ESDS).	There is a logic error in the CONNX 8.8 CICS/VSAM server program, probably caused by providing the wrong information to the CONNX 8.8 CDD import utility.	Reimport the CICS file and COBOL copybook and retry the DELETE. If the error persists, notify CONNX Technical Support.
CICS DeleteCurrentRecord() Error 01: Target VSAM file must be a Key Sequenced Data Set (KSDS) or a Relative Record Data Set (RRDS).	There is a logic error in the CONNX 8.8 CICS/VSAM server program, probably caused by providing the wrong information to the CONNX 8.8 CDD import utility.	Reimport the CICS file and COBOL copybook and retry the DELETE. If the error persists, notify CONNX Technical Support.
CICS DeleteCurrentRecord() Error 02: Read for Update failed; Delete bypassed.	The current record was originally read without an update lock. Prior to the delete attempt, CONNX unsuccessfully attempted to read the record again with an update lock.	This message is returned when there is a file locking conflict with another CICS transaction. Retry the DELETE; if the problem persists, notify your CICS administrator.
CICS UpdateCurrentRecord() Error 01: The requested update record length is greater than the maximum record length for the target file.	There is a logic error in the CONNX 8.8 CICS/VSAM server program, probably caused by providing the wrong information to the CONNX 8.8 CDD import utility.	Reimport the CICS file and COBOL copybook and retry the UPDATE. If the error persists, notify CONNX Technical Support.
CICS UpdateCurrentRecord() Error 02: Attempt to change the target VSAM file key field during an Update/REWRITE is not permitted; retry as an Insert/Delete.	The current SQL UPDATE statement includes one or more key fields in the SET clause; VSAM does not support an UPDATE (REWRITE) which changes the key of a KSDS.	Change the SQL Update statement to an INSERT of the new key/data followed by a DELETE of the current key.
CICS UpdateCurrentRecord() Error 03: Read for Update failed; Update bypassed.	The current record was originally read without an update lock. Prior to the Update (REWRITE) attempt, CONNX unsuccessfully attempted to read the record again with an update lock.	This message is returned when there is a file locking conflict with another CICS transaction. Retry the UPDATE; if the problem persists, notify your CICS administrator.
CICS UpdateCurrentRecord() Error 04: Target VSAM file is open for	There is a logic error in the CONNX VSAM server	Notify CONNX technical support.

Update, but there is no Update token.	program.	
CICS UpdateCurrentRecord() Error 05: Target VSAM file is not open for update, or there is no current record.	There is a logic error in the CONNX VSAM server program.	Notify CONNX technical support.

VSAM: File Transaction Error Messages

Text	Cause	Action
EndTransaction() Error 01: CICS Syncpoint failed.	The CONNX 8.8 CICS/VSAM server program issued an unsuccessful CICS SYNCPOINT command to end the current unit of work.	Examine the response and response2 return codes for further information; report error to CONNX Technical Support.
EndTransaction() Error 02: CICS Rollback failed.	The CONNX 8.8 CICS/VSAM server program issued an unsuccessful CICS SYNCPOINT ROLLBACK command to roll back the current unit of work.	Examine the response and response2 return codes for further information; report error to CONNX Technical Support.
EndTransaction() Error 03: Function invoked with invalid transaction type.	There is a logic error in the CONNX 8.8 CICS/VSAM server program.	Notify CONNX Technical Support.

VSAM: CICS Response2 Error Messages

Response2	Text	Cause	Action
1	File name not found in the file resource definition or the FCT (File Control Table).	CONNX has attempted to open a CICS 1-8 character file name which is not defined to CICS via RDO or an FCT entry.	Verify that the CICS file name exists via a CICS CEMT INQUIRE FILE command. Example: INQ FI(CUSTOMER) STATUS: RESULTS - OVERTYPE TO MODIFY Fil(CUSTOMER) Vsa Clo Ena Rea Upd Add Bro Del Sha Dsn(CONNX.CUSTOMER)
12	Record truncated: the specified length exceeds the maximum target file record size.	Probably an error caused by providing the wrong information to the CONNX CDD import utility.	Verify that the CICS short file name and the COBOL copybook used in the CDD import refer to the correct VSAM file; reimport the VSAM file and COBOL copybook; delete the truncated record, and re attempt the INSERT or UPDATE. If the problem persists, notify CONNX Technical Support.
13	The READ, READNEXT, or READPREV command LENGTH option specified an incorrect length for a file with fixed length records.	Refer to Response2 = 12	Refer to Response2 = 12
14	Incorrect length specified for a file with fixed length records.	Refer to Response2 = 12	Refer to Response2 = 12
20	FCT entry specification or RDO definition does not permit the requested BROWSE, READ, DELETE or UPDATE operation.	CONNX has attempted one of the above CICS commands, but the target CICS file definition does not include the required attribute.	Verify the current file attributes via the CICS CEMT INQUIRE FILE command. Example: INQ FI(CUSTOMER) STATUS: RESULTS - OVERTYPE TO MODIFY Fil(CUSTOMER) Vsa Clo

			Ena Rea Upd Add Bro Del Sha If the required attribute does not exist, temporarily add it and retry the operation. If the operation succeeds, request a permanent change to the file attributes. If the operations fails, notify CONNX Technical Support.
21	DELETE command issued against a VSAM ESDS (Entry-Sequenced Data Set).	Probably an error caused by providing the wrong information to the CONNX CDD import utility.	Re-import the CICS file and COBOL copybook and retry the DELETE. If the file is a VSAM ESDS, and the import is successful, this message should not display; instead, the error message should display as: CICS DeleteCurrentRecord() Error 01: Target VSAM file must be a KeySequenced Data Set (KSDS) or a Relative Record Data Set (RRDS). If neither message displays, and the target VSAM file is an ESDS, notify CONNX Technical Support.
24	READPREV command issued for a file for which the previous STARTBR or RESETBR command has the GENERIC option.	This is a logic error in the CONNX CICS/VSAM server.	Notify CONNX Technical Support.
25	Both KEYLENGTH and GENERIC options are specified, and the KEYLENGTH option is greater than or equal to the full key length.	Probably an error caused by providing the wrong information to the CONNX CDD import utility.	Reimport the CICS file and COBOL copybook and retry the SELECT. If the error recurs, notify CONNX Technical Support.
26	KEYLENGTH	Probably an error caused by	Reimport the CICS file and

	option is specified, and the length does not equal the target file key length.	providing the wrong information to the CONNX CDD import utility.	COBOL copybook and retry the SELECT. If the error recurs, notify CONNX Technical Support.
31	DELETE command without the RIDFLD option was issued against a file which has not been previously READ with UPDATE.	There is a logic error in the CONNX CICS/VSAM server.	Notify CONNX Technical Support.
33	A STARTBR (Start Browse) attempt was made with a REQID (Request ID) already in use.	There is a logic error in the CONNX CICS/VSAM server.	Notify CONNX Technical Support.
34	The REQID (Request ID) specified by a READNEXT command does not match any active STARTBR (Start Browse) REQID.	There is a logic error in the CONNX CICS/VSAM server.	Notify CONNX Technical Support.
35	The ENDBR (End Browse) REQID (Request ID) does not match any REQID from a STARTBR (Start Browse) command.	There is a logic error in the CONNX CICS/VSAM server.	Notify CONNX Technical Support.
37	A READNEXT or a READPREV command changed the type of record identification (for example, key or relative byte address) during the browse.	There is a logic error in the CONNX CICS/VSAM server.	Notify CONNX Technical Support.
41	The REQID	There is a logic error in the	Notify CONNX Technical

	specified by a READPREV command does not match any active STARTBR (Start Browse) REQID.	CONNX CICS/VSAM server.	Support.
47	A DELETE, REWRITE, or UNLOCK instruction token does not match any Read for UPDATE token.	There is a logic error in the CONNX CICS/VSAM server.	Notify CONNX Technical Support.
50	The target file is disabled, either by initial definition or by a SET FILE or a CEMT SET FILE command.	The CONNX CICS/VSAM server failed to OPEN and ENABLE the target file; another program disabled the file; or the file was disabled via the CEMT SET FILE command.	Manually open the file via the CEMT SET FILE command; retry the operation. If the request succeeds, ask the CICS administrator to change the initial file attributes, and/or authorize the CONNX CICS/VSAM server transaction and program (NXS0 and CNXVSAM) to execute the CICS SET FILE OPEN and ENABLED commands.
60	The target file is CLOSED and UNENABLED or OPEN and in use by other transactions.	The requested file is CLOSED and UNENABLED. Another CICS program or operator has issued a CLOSE request against the target file, or the file is defined as (CLOSED,UNENABLED), and OPENTIME(FIRSTREF) is omitted from the file resource definition or the requested file is OPEN and in use, but another transaction or operator has issued a CLOSE request against the file or the target file is quiesced, or is being quiesced, as a result of a SET DSNAME QUIESCED or IMMQUIESCED command.	Manually open and enable the target file via the CEMT INQUIRE / SET FILE commands; retry the request. Example: INQ FI(CUSTOMER) STATUS: RESULTS - OVERTYPE TO MODIFY Fil(CUSTOMER) Vsa Clo Dis Rea Upd Add Bro Del Sha Dsn(CONNX.CUSTOMER) SET FI(CUSTOMER) STATUS: RESULTS - OVERTYPE TO MODIFY Fil(CUSTOMER) Vsa Ope Ena Rea Upd Add Bro Del Sha Dsn(CONNX.CUSTOMER)

			If successful, ask the CICS administrator to change the file attributes.
70	An unknown failure occurred on a remote system.	The target CICS file is defined as a remote file, and the file I/O request has failed.	Ask the CICS administrator to verify that connectivity exists between the CICS region on which CONNX is installed and the remote region on which the target file is defined; retry the request.
80	The STARTBR (Start Browse), READ, DELETE or REWRITE attempt using the provided search argument returned a NOT FOUND condition.	This is additional information for the NOT FOUND CICS response; the request has failed because the search argument does not exist on the target VSAM file.	Modify the search argument (e.g. the SQL WHERE clause), and retry the request.
90	An end of file condition was detected during the browse.	Additional information for the CICS ENDFILE response; the browse has finished reading all records which meet the search criteria.	None; this is an informational message.
100	No space is available on the direct access device for adding the updated record to the data set.	The target VSAM data set has exceeded its primary and secondary extent allocations, or the DASD drive is full.	Notify the CICS administrator and systems programmer.
101	A resource security check failed on the target file.	Additional information for the NOTAUTH (Not Authorized) CICS response. The CICS user ID or transaction (the CONNX CICS/VSAM server: NXS0) is not authorized to perform the requested action against the target file.	Request the CICS administrator to change the authority for the CICS user ID and/or CONNX transaction and retry the request.
106	A READ, READNEXT, READPREV, DELETE, or REWRITE command specified the RIDFLD or UPDATE option, but a retained	Another CICS task or program has locked the record.	Retry the request later.

	lock exists against this key.		
109	Record changed since it was read for update. Repeat the read and retry the DELETE or REWRITE	Another program has changed the record.	Retry the SELECT/DELETE or the SELECT/UPDATE requests.
110	Undefined VSAM error occurred.	Additional information for the CICS ILLOGIC response; potentially a logic error in the CONNX CICS/VSAM server.	Retry the request; if the problem persists, notify CONNX Technical Support.
120	An I/O or hardware error occurred during the file command.	There is a file I/O or hardware error which cannot be addressed within CICS.	Notify the CICS administrator and the systems programmer.
150	A REWRITE attempt to a VSAM file caused a key collission for its UNIQUE alternate index.	The REWRITE to the target file caused a REWRITE to the upgrade set, resulting in a DUPREC response.	Informational; as required, delete the alternate index entry and retry the request.

VSAM: Started Task VSAM / QSAM / PDS: File Open Error Messages

Text	Cause	Action
File Open Error 1001: Read/Write access requested, but the target file does not have the read attribute.	Your user id does not have read access to the target VSAM, QSAM or PDS file.	Ask your security administrator to grant your user id read access to the target file.
File Open Error 1002: Read/Write access requested, but the target file does not have the write attribute.	Your user id does not have write or change access to the target VSAM or QSAM file.	Ask your security administrator to grant your user id write or change access to the target file.
File Open Error 1003: afopen() or afreopen() returned a NULL file pointer.	The attempt to open the target file failed, probably because of a mismatch in the imported CONNX *.cdd file JCL DDNAME or fully-qualified data set name, or because another process has exclusive access to the target file.	Review the DDNAME or fully-qualified data set name of the target file in the CONNX Data Dictionary (*.cdd) file. If the file name is a JCL DDNAME, verify that the DDNAME in the *.cdd file matches a DDNAME defined in the execution JCL for the CONNX Started Task or batch job. If the the file is a fully-qualified data set name, verify that the target data set exists and can be opened and viewed using standard TSO / ISPF utilities such as Browse or DITTO. If there is a data set sharing conflict with another process, the view attempt will fail with an appropriate message.
File Open Error 1004: osddinfo() return code != 0.	During file import or open, CONNX requested attribute information for the file corresponding to a user-entered JCL DDNAME in the File Name text box in the Table Properties tab of the CONNX Data Dictionary Manager.	Verify that the DDNAME in the *.cdd file matches a DDNAME defined in the execution JCL for the CONNX Started Task or batch job.
File Open Error 1005: CnxRACF::CheckDSNAccess() < 0 or > 4.	The CONNX data set security interface has returned an error condition during the attempted open, read, write, or delete operation against the target file. If the return code is < 0, the DASD volume serial of the target data set could not be located. If the return code is > 4, the logged on user id does not have the requested file read,	Verify that the target data set exists, and that your userid has the requested read, write, update, or delete file access privilege(s). If your userid has the necessary file access privileges, follow procedure A01.

	write, update, or delete privilege(s).	
File Open Error 1007: fattr() returned a NULL pointer.	After a call to afopen() or afreopen(), CONNX attempted to determine file attributes via the fattr() function call, which returned a NULL pointer.	Follow procedure A01.
File Open Error 1008: Read access requested, but the target file does not have the read attribute.	Your user id does not have read access to the target VSAM, QSAM or PDS file.	Ask your security administrator to grant your user id read access to the target file.
File Open Error 1012: osdsinfo() return code != 0.	The CONNX file open logic invoked to the osdsinfo() function using the fully-qualified dataset name corresponding to the JCL DDNAME stored in the CONNX Data Dictionary file. This function returns necessary information such as data set organization, record format, length, and blocksize. The file is either already open, or there is a data set sharing conflict with another subtask, task, or job.	Follow procedure A01.
File Open Error 1013: The physical data set name is not a VSAM, QSAM or PDS file.	CONNX attempted to open a file with an unsupported file access format.	Verify that the JCL DDNAME or the actual physical data set name used in the import or open request is a valid type of file (VSAM, QSAM physical sequential, or partitioned data set + member name.
File Open Error 1015: Read/Write access is not supported for partitioned data sets.	The invoking application has requested that CONNX open a fully-qualified PDS(member) data set name for read/write access. CONNX only supports read access for PDS members.	Revise the invoking application to issue read-only SELECTS against the fully-qualified PDS(member) data set name or JCL DDNAME.
File Open Error 1016: Unsupported data set record format.	CONNX detected an unsupported record format (RECFM) during import or open() of a physical sequential or a partitioned data set (member). Supported record formats are F (Fixed) and FB (Fixed Block). Unsupported record formats include FBS, VB, VBS, and U (Fixed Block	None. CONNX cannot import or open the designated file.

	Spanned, Variable Blocked, , Variable Blocked Spanned, and Undefined).	
File Open Error 1017: osopen() return code != 0.	During import or open() processing, CONNX unsuccessfully attempted to open the target file using the low-level osopen() BSAM (Basic Sequential Access Method) record-oriented interface function.	Follow procedure A01.
GetFirstRecord() Error 1021: Invalid key access method specified.	This is an internal CONNX logic error.	Report the error message to CONNX Technical Support.
GetFirstRecord() Error 1023: Invalid READ direction specified.	This is an internal CONNX logic error.	Report the error message to CONNX Technical Support.
GetFirstRecord() Error 1024: Invalid search operator specified.	This is an internal CONNX logic error.	Report the error message to CONNX Technical Support.
GetFirstRecord() Error 1027: ksearch() / fseek() failed.	File-positioning function ksearch (VSAM) or fseek() (QSAM or PDS) failed; the file I/O request is unsuccessful.	Follow procedure A01.
GetFirstRecord() Error 1029: kretrv() failed.	The file I/O function kretrv() failed to read the target VSAM file.	Follow procedure A01.
GetFirstRecord() Error 1031: seek to start-of-file failed.	The function kseek() (VSAM) or fseek() (QSAM / PDS) failed to set the file pointer to start-of file.	Follow procedure A01.
GetFirstRecord() Error 1032: The target file is not searchable or seekable.	The target VSAM, QSAM, or PDS file does not have the saerchable / seekable attribute.	Re-import the file into the CONNX CDD and re-try the SQL statement. If the error persists, follow procedure A01.
GetFirstRecord() Error 1033: The file pointer is NULL.	The target VSAM, QSAM, or PDS file pointer is NULL, due to an open() error, a dataset sharing conflict with another job, process, or task, or an internal logic error.	Re-try the SQL statement. If the error persists, follow procedure A01.
GetFirstRecord() Error 1037: The target file is not a VSAM data set.	Either there was a CONNX file import error, or the DDNAME or fully-qualified data set name in the CONNX execution JCL points to a non-VSAM data set.	Re-import the file into the CONNX CDD and re-try the SQL statement. If the error persists, follow procedure A01.
DoReadNextPrev() Error 1051: afread() error: 0 bytes returned	The afread() file I/O function returned 0 bytes plus an error	If the error is reproducible, follow procedure A01.

and ferror() != 0.	return code.	
DoReadNextPrev() Error 1052: afread() error: bytes returned > 0 and ferror() != 0.	The afread() file I/O function returned a positive byte count and an error return code.	If the error is reproducible, follow procedure A01.
DoReadNextPrev() Error 1053: afread() error: return code < 0.	The afread() file I/O function returned an error code.	If the error is reproducible, follow procedure A01.
DoReadNextPrev() Error 1054: osget() error: return code != 0 (OK) or -1 (EOF).	An attempt to read the target file using the SAS/C BSAM record-oriented interface function returned an error code which was not 0 (OK) or -1 (EOF).	If the error is reproducible, follow procedure A01.
DoReadNextPrev() Error 1055: The target file is not a VSAM data set.	An attempt to read the target file failed because it is not a VSAM data set. Either the file attributes of the data set have been changed since it was last imported into the CONNX CDD, or this is an internal logic error.	Verify that the DDNAME or physical data set name stored in the CONNX CDD file points to a VSAM KSDS, ESDS, or RRDS dataset. If the target dataset is VSAM, then report the error to CONNX Technical Support.
DoReadNextPrev() Error 1056: The target file is not a QSAM dataset or a PDS member.	An attempt to read the target file failed because it is not a QSAM physical sequential data set or a fully-qualified PDS(member). Either the file attributes of the data set have been changed since it was last imported into the CONNX CDD, or this is an internal logic error.	Verify that the DDNAME or physical data set name stored in the CONNX CDD file points to a QSAM physical sequential data set or a fully-qualified PDS(member). If the target dataset organization (QSAM or PDS(member)) is correct, then report the error to CONNX Technical Support.

VSAM: Started Task VSAM / QSAM / PDS: File Read Error Messages

Text	Cause	Action
GetFirstRecord() Error 1021: Invalid key access method specified.	This is an internal CONNX logic error.	Report the error message to CONNX Technical Support.
GetFirstRecord() Error 1023: Invalid READ direction specified.	This is an internal CONNX logic error.	Report the error message to CONNX Technical Support.
GetFirstRecord() Error 1024: Invalid search operator specified.	This is an internal CONNX logic error.	Report the error message to CONNX Technical Support.
GetFirstRecord() Error 1027: ksearch() / fseek() failed.	File-positioning function ksearch (VSAM) or fseek() (QSAM or PDS) failed; the file I/O request is unsuccessful.	Follow procedure A01.
GetFirstRecord() Error 1029: kretrv() failed.	The file I/O function kretrv() failed to read the target VSAM file.	Follow procedure A01.
GetFirstRecord() Error 1031: seek to start-of-file failed.	The function kseek() (VSAM) or fseek() (QSAM / PDS) failed to set the file pointer to start-of file.	Follow procedure A01.
GetFirstRecord() Error 1032: The target file is not searchable or seekable.	The target VSAM, QSAM, or PDS file does not have the saerchable / seekable attribute.	Re-import the file into the CONNX CDD and re-try the SQL statement. If the error persists, follow procedure A01.
GetFirstRecord() Error 1033: The file pointer is NULL.	The target VSAM, QSAM, or PDS file pointer is NULL, due to an open() error, a dataset sharing conflict with another job, process, or task, or an internal logic error.	Re-try the SQL statement. If the error persists, follow procedure A01.
GetFirstRecord() Error 1037: The target file is not a VSAM data set.	Either there was a CONNX file import error, or the DDNAME or fully-qualified data set name in the CONNX execution JCL points to a non-VSAM data set.	Re-import the file into the CONNX CDD and re-try the SQL statement. If the error persists, follow procedure A01.
DoReadNextPrev() Error 1051: afread() error: 0 bytes returned and ferror() != 0.	The afread() file I/O function returned 0 bytes plus an error return code.	If the error is reproducible, follow procedure A01.
DoReadNextPrev() Error 1052: afread() error: bytes returned > 0 and ferror() != 0.	The afread() file I/O function returned a positive byte count and an error return code.	If the error is reproducible, follow procedure A01.
DoReadNextPrev() Error 1053: afread() error: return code < 0.	The afread() file I/O function returned an error code.	If the error is reproducible, follow procedure A01.

DoReadNextPrev() Error 1054: osget() error: return code != 0 (OK) or -1 (EOF).	An attempt to read the target file using the SAS/C BSAM record-oriented interface function returned an error code which was not 0 (OK) or -1 (EOF).	If the error is reproducible, follow procedure A01.
DoReadNextPrev() Error 1055: The target file is not a VSAM data set.	An attempt to read the target file failed because it is not a VSAM data set. Either the file attributes of the data set have been changed since it was last imported into the CONNX CDD, or this is an internal logic error.	Verify that the DDNAME or physical data set name stored in the CONNX CDD file points to a VSAM KSDS, ESDS, or RRDS dataset. If the target dataset is VSAM, then report the error to CONNX Technical Support.
DoReadNextPrev() Error 1056: The target file is not a QSAM dataset or a PDS member.	An attempt to read the target file failed because it is not a QSAM physical sequential data set or a fully-qualified PDS(member). Either the file attributes of the data set have been changed since it was last imported into the CONNX CDD, or this is an internal logic error.	Verify that the DDNAME or physical data set name stored in the CONNX CDD file points to a QSAM physical sequential data set or a fully-qualified PDS(member). If the target dataset organization (QSAM or PDS(member)) is correct, then report the error to CONNX Technical Support.

VSAM: Started Task VSAM / QSAM / PDS: File Update Error Messages

Text	Cause	Action
UpdateCurrentRecord() Error 1080: kreplace() return code is non-zero.	An error occurred during an update-in-place attempt against a VSAM KSDS, ESDS, or RRDS file.	If the error is reproducible, follow procedure A01.
UpdateCurrentRecord() Error 1081: The record buffer pointer is NULL.	This is an internal logic error message.	Report the error message to CONNX Technical Support. On request, follow procedure A01.
UpdateCurrentRecord() Error 1083: The record length is invalid.	This is an internal logic error message.	Report the error message to CONNX Technical Support. On request, follow procedure A01.
UpdateCurrentRecord() Error 1084: An attempt to change the target VSAM file key field during an Update is not permitted; retry as an Insert + Delete.	The current UPDATE SQL statement contains one or more SET column = value clauses which reference all or part of a unique key. This operation is not permitted against VSAM KSDS files.	Revise the invoking application to code an INSERT followed by a DELETE of the current record.
UpdateCurrentRecord() Error 1085: Read for Update failed; Update bypassed.	The UPDATE SQL statement referenced a VSAM record which was not currently locked. An attempt to read the record with an exclusive lock failed. The record is either locked by the current or another client application.	Revise the client application to issue a COMMIT prior to the UPDATE, and try again. If the problem persists, browse the STDERR JES output or physical sequential data set for SAS/C runtime library error messages. Also browse the Started Task / Batch job JES output and the system log for dataset allocation or sharing conflict messages. If the error is reproducible, follow procedure A01.
UpdateCurrentRecord() Error 1086: Update is not supported for QSAM physical sequential and partitioned data sets.	The invoking client application sent an SQL UPDATE statement against a non-VSAM (QSAM physical sequential or PDS) file. UPDATE is only supported for VSAM data sets.	Remove the UPDATE statement from the invoking client application.
UpdateCurrentRecord() Error 1087: The target file is not a VSAM data set.	An attempt to update the target file failed because it is not a VSAM data set. Either the file attributes of the data set have been changed since it was last imported into the CONNX CDD, or this is an internal logic error.	Verify that the DDNAME or physical data set name stored in the CONNX CDD file points to a VSAM data set. If the target dataset is a VSAM KSDS, ESDS, or RRDS, then report the error to CONNX Technical Support.

VSAM: Started Task VSAM / QSAM / PDS: File Delete Error Messages

Text	Cause	Action
DeleteCurrentRecord() Error 1090: kdelete() return code is non-zero.	An SQL DELETE statement against a VSAM file returned an error code.	kdelete() fails when invoked against an ESDS. Verify that the target file is a VSAM KSDS or an RRDS. If the error is reproducible, follow procedure A01 below.
DeleteCurrentRecord() Error 1092: Delete is not supported for Entry-Sequenced Data Sets (ESDS).	The client application attempted an SQL DELETE against an ESDS.	Verify that the target data set is an ESDS; if it is, then remove the DELETE statements against the ESDS from the invoking application. If the target data set is not an ESDS, report the error scenario to CONNX Technical Support.
DeleteCurrentRecord() Error 1093: Delete is not supported for QSAM physical sequential and partitioned data sets.	The client application attempted an SQL DELETE against a QSAM physical sequential or a partitioned data set.	Verify that the target data set is QSAM PS or PDS; if it is a QSAM PS or a PDS, then remove the DELETE statements against the data set from the invoking application. If the target data set is not a QSAM PS or a PDS, report the error scenario to CONNX Technical Support.
DeleteCurrentRecord() Error 1094: The target file is not a VSAM data set.	An SQL DELETE attempt against the target file failed because it is not a VSAM KSDS or RRDS data set. Either the file attributes of the data set have been changed since it was last imported into the CONNX CDD, or this is an internal logic error.	Verify that the DDNAME or physical data set name stored in the CONNX CDD file points to a VSAM KSDS or RRDS data set. If the target data set is a VSAM KSDS or RRDS, then report the error to CONNX Technical Support.

VSAM: Started Task VSAM / QSAM / PDS: File Insert Error Messages

Text	Cause	Action
InsertRecord() Error 1100: kinsert() return code is non-zero.	SQL INSERT attempt failed against a VSAM KSDS, ESDS, or RRDS file.	If the file is a KSDS, verify that the key is unique. If the error is reproducible, follow procedure A01.
InsertRecord() Error 1101: The target VSAM file does not have the insert or add attribute.	The most recent import or open() of the target file did not return an insert or add attribute. The insert is rejected.	Ask your security administrator to verify that the target VSAM file is updatable. Re-import the target VSAM file into the CONNX CDD and retry the insert.
InsertRecord() Error 1102: osput() return code is non-zero.	The low-level BSAM record-oriented interface function osput() returned a non-zero value. This is either a file I/O (-2) or a record length error (-3).	If the error is reproducible, follow procedure A01.
InsertRecord() Error 1103: The record buffer pointer is NULL.	This is an internal logic error returned from the CONNX VSAM or QSAM InsertRecord() function.	Report the error message to CONNX Technical Support.
InsertRecord() Error 1104: The record length is invalid.	This is an internal logic error returned from the CONNX VSAM InsertRecord() function. The record length for an insert against an ESDS or an RRDS file is zero or negative.	Report the error message to CONNX Technical Support.
InsertRecord() Error 1105: fseek() to end-of-file failed.	Prior to processing an insert request to a QSAM physical sequential (PS) file, CONNX attempted to position the file pointer to end-of-file via the fseek() function, which failed with a non-zero return code.	If the error is reproducible, follow procedure A01.
InsertRecord() Error 1107: The record length is <= 0.	This is an internal logic error. The CONNX QSAM Insert() function was invoked with a non-positive record length.	Report the error message to CONNX Technical Support.
InsertRecord() Error 1108: Insert is not supported for partitioned data sets.	The invoking application attempted to execute an SQL INSERT against a fully-qualified PDS(member). SQL INSERTs are only supported for VSAM and QSAM physical sequential data sets.	Remove the SQL Insert statement from the invoking CONNX client application.
InsertRecord() Error 1109: The target file is not a QSAM data set.	The invoking application attempted to execute an SQL INSERT against a data set	Verify that the CONNX CDD table entry refers to a JCL DDNAME or a fully-qualified

	which is not a QSAM physical sequential data set.	data set name which is a QSAM physical sequential data set. If the target data set is a QSAM physical sequential file, re-import the file into the CONNX CDD and retry the INSERT. If the error message persists, report it to CONNX Technical Support.
InsertRecord() Error 1160: The target file is not a VSAM data set.	Either there is an error in the CONNX CDD import or file open function, or the data set attributes have changed since the last import into the CONNX CDD file.	Re-import the target file COBOL copybook + JCL DDNAME or fully-qualified VSAM physical data set name into the CONNX CDD file. If the error is reproducible, follow procedure A01.

VSAM: Started Task VSAM / QSAM / PDS: File Close Error Messages

Text	Cause	Action
Close() Error 1110: fclose() return code != 0.	CONNX attempted to close a VSAM / QSAM / PDS file via the fclose() function, which returned a non-zero return code.	If the error is reproducible, follow procedure A01.
Close() Error 1111: osclose() return code != 0.	CONNX invoked the BSAM Record-Oriented Interface function osclose() to close an open DCB (Dataset Control Block). The function returned a non-zero return code, probably because it could not flush the file buffers.	If the error is reproducible, follow procedure A01.

VSAM: Started Task VSAM / QSAM / PDS: File Locking Error Messages

Text	Cause	Action
UnlockCurrentRecord() Error 1120: kseek() / fseek() with SEEK_CUR return code is non-zero.	CONNX attempted to unlock the current record for a VSAM, QSAM physical sequential, or PDS(member) file by invoking kseek() or fseek() function to seek to the current record pointer. The function returned a non-zero error return code.	If the error is reproducible, follow procedure A01.

Started Task VSAM / QSAM / PDS: Memory Allocation Error Messages

Text	Cause	Action
File I/O Error: GetRecordBufferRO() returned a NULL record pointer.	CONNX attempted to allocate a read-only I/O record buffer. The function returned a NULL pointer, probably due to a virtual memory shortage or to a non-positive record length.	If the error is reproducible, follow procedure A01.

VSAM: Started Task VSAM / QSAM / PDS: Transaction Error Messages

Text	Cause	Action
EndTransaction() Error 1140: CONNX for VSAM / QSAM / PDS does not support Rollback.	The invoking client application requested a Rollback to undo changes to one or more VSAM / QSAM / PDS(member) data sets. CONNX only supports Rollback for journalled CICS / VSAM files.	Remove the Rollback request from the invoking client application.
EndTransaction() Error 1141: Invalid transaction type.	This is an internal logic error: the EndTransaction() function was invoked with an invalid parameter type.	Report the error message to CONNX Technical Support.

VSAM: Started Task VSAM / QSAM / PDS: Error Message Diagnostic Action Table

Action Code A01:

Enable the CONNX DEBUG Trace and Send the Trace Output to CONNX Technical Support

If the SAS/C STDERR DDNAME points to a JES SYSOUT dataset or to a physical sequential data set, browse the output for SAS/C runtime library error messages. In some cases, these messages may be sufficent to diagnose the error scenario.

By default, both the STDERR and STDOUT DDNAMES are dummied out in JCL procedures CNXVSAM and CNXVSCT, which reside in the CONNX JCL dataset (default name = CONNX.VVRR.STASK.CNTL).

```
//STDERR       DD DUMMY
//STDOUT       DD DUMMY
```

If the CONNX VSAM / QSAM / PDS listener program is executed as a started task, change the started task DDNAME definitions to point to the pre-allocated *.STDERR and *.STDOUT datasets; for example:

```
//STDERR DD DSN=CONNX.VVRR.STASK.STDERR,DISP=SHR
//STDOUT DD DSN=CONNX.VVRR.STASK.STDOUT,DISP=SHR
```

If the CONNX VSAM / QSAM / PDS listener program is executed as a batch job, add overrides to the execution JCL member (default = CNXVSAMJ):

```
//CNXRUNB.STDERR DD DSN=CONNX.VVRR.STASK.STDERR,DISP=SHR
//CNXRUNB.STDOUT DD DSN=CONNX.VVRR.STASK.STDOUT,DISP=SHR
```

Pointing DDNAME STDERR to SYSOUT or to a sequential dataset will capture SAS/C runtime library error messages. For further diagnostic messages, enable the CONNX debug trace logic by setting the CONNX DEBUG environment variable to 1. To do so, refer to the CONNX online help file Installation Guide and drill down to:

Step 5: Install IBM Mainframe-Compatible Server + CONNX for VSAM / QSAM / PDS +

Part 3: CONNX TSO Configuration Utility + CONNX Environment Variables + To define unique CONNX environment variables per Started Task / Batch Job.

Find the current control record member used to define CONNX environment variables in the CONNX *.CNTL partitioned data set (default = CNXPARMS) and change the =DEBUG environment variable to 1:

```
=DEBUG=1
```

Next, stop the CONNX VSAM / QSAM / PDS Started Task / Batch job TCP/IP listener program (CNXRUNB). Refer to the CONNX online help file Installation Guide and drill down to:

Step 5: Install IBM Mainframe-Compatible Server + CONNX for VSAM / QSAM / PDS +

Part 3: CONNX TSO Configuration Utility + To stop the CONNX Started Task/Batch Job TCP/IP Listener

Restart the the CONNX VSAM / QSAM / PDS Started Task / Batch job TCP/IP listener program (CNXRUNB) by submitting the execution JCL (CNXVSAMJ) or by starting the started task (CNXVSAM).

Re-run the problem scenario from the client PC(s). If the trace logic has been successfully enabled, the *.STDOUT physical sequential dataset will contain CONNX debug messages; additional SAS/C runtime library error messages, if any, will be written to the *.STDERR physical sequential dataset.

Send both trace output datasets via email to CONNX Technical Support at support@connx.com .

After successfully generating a CONNX trace, remember to disable the CONNX trace logic by resetting the DEBUG environment variable in the CNXPARMS member of the CONNX *.CNTL partitioned data set to 0:

```
=DEBUG=0
```

Next, reinstate the STDERR and STDOUT DDNAMEs to DUMMY in the CNXVSAM JCL procedure:

```
//STDERR       DD DUMMY
//STDOUT       DD DUMMY
```

If the CONNX VSAM / QSAM / PDS listener program is executed as a batch job, remove the overrides to the execution JCL member (default = CNXVSAMJ):

```
//CNXRUNB.STDERR DD DSN=CONNX.VVRR.STASK.STDERR,DISP=SHR
//CNXRUNB.STDOUT DD DSN=CONNX.VVRR.STASK.STDOUT,DISP=SHR
```

Finally, stop and restart the CONNX VSAM / QSAM / PDS TCP/IP listener program as documented above.

Appendix A - Technical Support

Technical Support

CONNX products can be installed only as licensed by CONNX Solutions. Software cannot be transferred from one computer to another or to another portion of your network before registration procedures are completed and without the written consent of CONNX Solutions.

This guide assumes that you are familiar with the operating procedures of your computer and with the Windows environment. For information about your file system and the Microsoft® Windows® environment, refer to the Windows documentation.

If you encounter problems with the installation or operation of CONNX, contact an authorized CONNX reseller or CONNX Technical Support as follows:

Telephone: (888) 930-2727 (For users with a Maintenance Support Contract.)

Telephone: (425) 519-6600 (For international inquiries.)

Fax for all users: (425) 519-6601

E-mail: support@connx.com

If you purchased CONNX through a value-added reseller (VAR), please contact them for support.

CONNX technical support personnel are available Monday through Friday, between 6:00 a.m. and 5:00 p.m., Pacific Time. If you have questions regarding your maintenance support, see your CONNX contract for details.

It is important that you register your copy of CONNX either by completing the registration card supplied with the CONNX CD-ROM or by visiting the CONNX Web site at www.connx.com

Proper registration ensures receipt of Service Pack notices, tips and hints for CONNX operation, related CONNX information, and uninterrupted technical support.

Adabas SQL Gateway users:

Product support and technical assistance for the Adabas SQL Gateway (CONNX for Adabas) are available through your local Software AG Regional Support Center, Software AG´s ServLine24, or your Software AG Account Manager.

Access to ServLine24 can be found at the following addresses:

https://empower.softwareag.com/default.asp

http://www.softwareag.com

Adabas SQL Gateway support can also be reached via e-mail at support@softwareag.com.

Trial copies and pre-release versions are covered by separate contracts. Please contact your Software AG Account Manager for more information.

Appendix B - Trademarks and Copyrights

Trademarks and Copyright

Legal Notices

This notice is valid for all products of the Adabas®, Bolero®, EntireX®, Natural® and Tamino® (the "Product") product line and associated Software AG products.

IMPORTANT: PLEASE READ BEFORE INSTALLING THE SOFTWARE

You are not allowed to install or use the Product without a corresponding license agreement.

If you have entered into a license agreement with one of Software AG's subsidiaries or distributors (the "License Agreement") the installation and use of the Product is subject to your acceptance of additional terms which are provided for you in the section Additional Terms.

If you have not entered into a License Agreement you must execute the license agreement with Software AG which is provided for you in the section Software AG License Agreement for Trial Versions of Software AG Software ONLY.

Copyright Notice

Adabas®, Bolero®, EntireX®, Natural® and Tamino® are Products of Software AG, a German corporation with its principal place of business at Uhlandstrasse 12, 64297 Darmstadt.

Software AG is the sole owner of all industrial property rights and copyright to the Products and accompanying user documentation or has the respective distribution rights. References made in or on the Products to the copyright or to other industrial property rights must not be altered, deleted or obliterated in any manner.

Trademark Notice

The name Software AG and/or all Software AG Products are either trademarks or registered trademarks of Software AG. Other product and company names mentioned herein may be the trademarks of their respective owners.

CONNX, FTL, InfoNaut, InfoNaut Professional, and CONNX Data Synchronization are trademarks or registered trademarks of Software AG, in the USA and other countries.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at http://softwareag.com/licenses.

This software may include portions of third-party products. For third-party copyright notices, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at http://softwareag.com/licenses and/or in the root installation directory of the licensed product(s).

Use, reproduction, transfer, publication or disclosure is prohibited except as specifically provided for in your License Agreement with Software AG.

The following are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries:Microsoft, Microsoft Access, Microsoft Excel, Microsoft Internet Information Server,

Microsoft .NET, Microsoft SQL Server, Microsoft Visual Basic, Microsoft Visual C++, Microsoft Visual FoxPro, Microsoft Visual Studio, Microsoft Windows.

The following are either registered trademarks or trademarks of their respective companies or organizations in the United States and/or other countries: Adabas - Software AG; Extra!, Attachmate - Attachmate Corporation; Borland C++ - Inprise Corporation; Crystal Reports - Seagate Software; DataFlex - Data Access Corporation; DB2, AIX, AS/400, C-ISAM, OS/2, OS/390, OS/400, IBM Communications Server, IBM Personal Communications, IBM eNetwork Personal Communications, IBM eNetwork Communications Server - International Business Machines Corporation; Dharma ODBC Integrator - Dharma Systems, Inc.; FOCUS -- IBI, Inc.; HP-UX -- Hewlett-Packard Company; Impromptu - Cognos Incorporated; Itanium - Intel Corporation; JetForm - JetForm Corporation; Linux - Linus Torvalds; NetWare for SAA, Novell - Novell, Inc.; Micro Focus -- Micro Focus International Limited; Oracle, Oracle Rdb, Oracle CODASYL DBMS, Oracle Developer, Form Builder, and Report Builder - Oracle Corporation; Paradox - Corel Corporation; Pathworks, Network File Transfer, RMS, DECnet, VMS, OpenVMS, VAXServer, AlphaServer - Compaq Computer Corporation; PowerBuilder - Sybase, Inc.; POISE (People Oriented Information Systems for Education) -- Campus America/Jenzabar; POWER*flex* - POWER*flex* Corporation; RUMBA, Wall Data - NetManage, Inc.; SCO -- Caldera Systems, Inc.; SCT, Plus2000 - Systems and Computer Technology Corporation; SIS - University of California Berkeley; Solaris, Sun - Sun Microsystems, Inc.; Unix - The Open Group.

Portions of the **PostgreSQL Data Base Management System** are copyrighted © 1996-2002, PostgreSQL Global Development Group
Portions Copyright © 1994-7 Regents of the University of California
Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.
IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING LOST PROFITS, ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE UNIVERSITY OF CALIFORNIA SPECIFICALLY DISCLAIMS ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE SOFTWARE PROVIDED HEREUNDER IS ON AN "AS IS" BASIS, AND THE UNIVERSITY OF CALIFORNIA HAS NO OBLIGATIONS TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

SAS Institute Inc.
SAS/C OnlineDocTM, Release 7.00, Cary, NC: SAS Institute Inc., 2001.
SAS/C OnlineDocTM, Release 7.00, April 2001
SAS/C OnlineDocTM, Release 7.50, April 2004
Copyright © 2001 by SAS Institute Inc., Cary, NC, USA.
All rights reserved. Produced in the United States of America.
U.S. Government Restricted Rights Notice
Use, duplication, or disclosure of this software and related documentation by the U.S. government is subject to the Agreement with SAS Institute and the restrictions set forth in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June 1987).
SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.
April 2001
SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

dbCrypto
Copyright (c) 2002, Dr Brian Gladman, Worcester, UK.
All rights reserved.

LICENSE TERMS
The free distribution and use of this software in both source and binary
form is allowed (with or without changes) provided that:
1. distributions of this source code include the above copyright
   notice, this list of conditions and the following disclaimer;
2. distributions in binary form include the above copyright
   notice, this list of conditions and the following disclaimer
   in the documentation and/or other associated materials;
3. the copyright holder's name is not used to endorse products
   built using this software without specific written permission.
ALTERNATIVELY, provided that this notice is retained in full, this product
may be distributed under the terms of the GNU General Public License (GPL),
in which case the provisions of the GPL apply INSTEAD OF those given above.
DISCLAIMER
This software is provided 'as is' with no explicit or implied warranties
in respect of its properties, including, but not limited to, correctness
and/or fitness for purpose.

SIS

Copyright (c) 1989-1994, 2017 The Regents of the University of California.
All rights reserved.

Permission is hereby granted, without written agreement and without
license or royalty fees, to use, copy, modify, and distribute this
software and its documentation for any purpose, provided that the
above copyright notice and the following two paragraphs appear in all
copies of this software.

IN NO EVENT SHALL THE UNIVERSITY OF CALIFORNIA BE LIABLE TO ANY PARTY
FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES
ARISING OUT OF THE USE OF THIS SOFTWARE AND ITS DOCUMENTATION, EVEN IF
THE UNIVERSITY OF CALIFORNIA HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.

ICU Library

COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1995-2005 International Business Machines Corporation and others

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO

EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION

WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

Hiredis

Copyright (c) 2009-2011, Salvatore Sanfilippo <antirez at gmail dot com>

Copyright (c) 2010-2011, Pieter Noordhuis <pcnoordhuis at gmail dot com>

All rights reserved.

Redistribution and use in source and binary forms, with or without

modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice,
  this list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice,
  this list of conditions and the following disclaimer in the documentation
  and/or other materials provided with the distribution.
* Neither the name of Redis nor the names of its contributors may be used
  to endorse or promote products derived from this software without specific
  prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR

ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

OCILib

Licensed under Apache 2.0

SAFMQ

Copyright 2004-2007 Matthew J. Battey, Omaha, NE

Licensed under the Apache License, Version 2.0 (the "License");

you may not use this file except in compliance with the License.

You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software

distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and

limitations under the License.

FreeTDS

Copyright (C) 1998-1999 Brian Bruns

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Library General Public License for more details. You should have received a copy of the GNU Library General Public License along with this library; if not, write to the Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.

ACE Library

Copyright and Licensing Information for ACE(TM), TAO(TM), CIAO(TM), DAnCE(TM), and CoSMIC(TM)

ACE(TM), TAO(TM), CIAO(TM), DAnCE>(TM), and CoSMIC(TM) (henceforth referred to as "DOC software") are copyrighted by Douglas C. Schmidt and his research group at Washington University, University of California, Irvine, and Vanderbilt University, Copyright (c) 1993-2009, all rights reserved. Since DOC software is open-source, freely available software, you are free to use, modify, copy, and distribute--perpetually and irrevocably--the DOC software source code and object code produced from the source, as well as copy and distribute modified versions of this software. You must, however, include this copyright statement along with any code built using DOC software that you release. No copyright statement needs to be provided if you just ship binary executables of your software products.

You can use DOC software in commercial and/or binary software releases and are under no obligation to redistribute any of your source code that is built using DOC software. Note, however, that you may not misappropriate the DOC software code, such as copyrighting it yourself or claiming authorship of the DOC software code, in a way that will prevent DOC software from being distributed freely using an open-source development model. You needn't inform anyone that you're using DOC software in your software, though we encourage you to let us know so we can promote your project in the DOC software success stories.

The ACE, TAO, CIAO, DAnCE, and CoSMIC web sites are maintained by the DOC Group at the Institute for Software Integrated Systems (ISIS) and the Center for Distributed Object Computing of Washington University, St. Louis for the development of open-source software as part of the open-source software community. Submissions are provided by the submitter ``as is'' with no warranties whatsoever, including any warranty of merchantability, noninfringement of third party intellectual property, or fitness for any particular purpose. In no event shall the submitter be liable for any direct, indirect, special, exemplary, punitive, or consequential damages, including without limitation, lost profits, even if advised of the possibility of such damages. Likewise, DOC software is provided as is with no warranties of any kind, including the warranties of design, merchantability, and fitness for a particular purpose, noninfringement, or arising from a course of dealing, usage or trade practice. Washington University, UC Irvine, Vanderbilt University, their employees, and students shall have no liability with respect to the infringement of copyrights, trade secrets or any patents by DOC software or any part thereof. Moreover, in no event will Washington University, UC Irvine, or Vanderbilt University, their employees, or students be liable for any lost revenue or profits or other special, indirect and consequential damages.

DOC software is provided with no support and without any obligation on the part of Washington University, UC Irvine, Vanderbilt University, their employees, or students to assist in its use, correction, modification, or enhancement. A number of companies around the world provide commercial support for DOC software, however. DOC software is Y2K-compliant, as long as the underlying OS platform is Y2K-compliant. Likewise, DOC software is compliant with the new US daylight savings rule passed by Congress as "The Energy Policy Act of 2005," which established new daylight savings times (DST) rules for the United States that expand DST as of March 2007. Since DOC software obtains time/date and calendaring information from operating systems users will not be affected by the new DST rules as long as they upgrade their operating systems accordingly.

The names ACE(TM), TAO(TM), CIAO(TM), DAnCE(TM), CoSMIC(TM), Washington University, UC Irvine, and Vanderbilt University, may not be used to endorse or promote products or services derived from this source without express written permission from Washington University, UC Irvine, or Vanderbilt University. This license grants no permission to call products or services derived from this source ACE(TM), TAO(TM), CIAO(TM), DAnCE(TM), or CoSMIC(TM), nor does it grant permission for the name Washington University, UC Irvine, or Vanderbilt University to appear in their names.

UMac

Copyright (c) 1999-2000 Ted Krovetz (tdk@acm.org)

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee, is hereby granted, provided that the above copyright notice appears in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of the University of California and Ted Krovetz not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

The Regents of the University of California and Ted Krovetz disclaim all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall the University of California or Ted Krovetz be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

Gigabase

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the Software), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHOR OF THIS SOFTWARE BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

System.Net.FtpClient

Copyright (c) 2013 J.P. Trosclair

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

SharpSSH

Copyright (c) 2002-2015 Atsuhiko Yamanaka, JCraft,Inc.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The names of the authors may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL JCRAFT, INC. OR ANY CONTRIBUTORS TO THIS SOFTWARE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT

LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Mentalis

Copyright © 2002-2007, The Mentalis.org Team

All rights reserved.

http://www.mentalis.org/

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

- Neither the name of the Mentalis.org Team, nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER

CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

NLog

Copyright (c) 2004-2016 Jaroslaw Kowalski <jaak@jkowalski.net>, Kim Christensen, Julian Verdurmen

All rights reserved.

Redistribution and use in source and binary forms, with or without

modification, are permitted provided that the following conditions

are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of Jaroslaw Kowalski nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR

CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

MariaDB ODBC Driver

The MariaDB client libraries for C, Java and ODBC are distributed under the LGPL license, version 2.1 or later. The LGPL license allows you to distribute these MariaDB client libraries freely with any application.

About Software AG, Inc.

Based in Reston, Va., Software AG, Inc. is a wholly owned subsidiary of Software AG headquartered in Darmstadt, Germany. Founded in 1969, Software AG continues to be Europe's largest and a leading global provider of system software and services enabling enterprise data integration and management. Software AG's products control the central IT processes of thousands of renowned companies worldwide including Lufthansa, Siemens, Citibank, Merck, DaimlerChrysler, Sony, BP and Telefonica. Software AG develops products and solutions that support the XML (Extensible Markup Language) standard. XML simplifies the exchange of documents and data as well as the integration of cutting-edge Web applications into traditional IT architectures. In 2003, the corporation achieved 422 million euros in total revenue. Software AG has representation in 59 countries worldwide, and currently employs a staff of approximately 2,600.

Software AG and/or all Software AG products are either trademarks or registered trademarks of Software AG.

All other product and company names mentioned herein are used for identification purposes only, and may be the trademarks or service marks of their respective owners.

Appendix C - GDPR and Personal Data

General Data Protection Regulation

‘Personal data’ means any information relating to an identified or identifiable natural person (‘data subject’); an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person.

Personal data in CONNX can be found in two locations:

1) The CONNX Data Dictionary, under users.

2) Log files for CONNX, including the CONNX data server log files (cnxrun.log, cnxrms.log, cnxrdb.log, cnxdbms.log), mainframe logs in either sysprint or TDQ stdout on VSE, or the JDBC server log files (jdbcserver.log or jdbcserver64.log).

When a natural person leaves an organization, their personal data can be removed with the following steps:

1) Remove the user from all CONNX data dictionaries.

2) Delete all CONNX client and server log files as listed above.

Appendix D - Glossary

Terms and Abbreviations

ActiveX
Adabas
ADO
AIX
AnyNet
API
Applet
Application Requester (AR)
Application Server (AS)
APPC
Big-endian
Bind
CCSID
CICS
CMS
COM
Compiler
Conversation
Data Access Engine
Data set name
DB2
Database Management System (DBMS)
DBCS
DCT
DDM
DHTML
DOS/VSE
DRDA
DRDA connection
Dynamic SQL
ERP
ESDS
Extrapartition transient data queue
Heterogeneous data sources
HPUX
HTML
Java
JavaScript
JCL

JCT
JDBC
JDK
JES
JNI
JVM
KSDS
LDS
Linux
Listener process
Little-endian
Localhost
Logical Unit (LU)
Logical Unit of Work (LUW)
Logical Unit Type 6.2 (LU 6.2)
MBCS
MDAC
MVS
ODBC
OLE DB
Package
PDS
PLT
Port
QSAM
RDB
Relational Database Name (Rdb Name)
RDO
RDS
recordset
RRDS
SBCS
SCO
SIT
SNA
Socket
Solaris
SQL
System/370
System/390
TCP/IP
Transaction program (TP)

trusted applet
TSO
UDA
UNC
Unit of work
Unix
VM
VSE
Well-known port
Windows DNA
z/OS

Index

D

E

T