CONNX Planning Guide

Table Of Contents

Introduction 1
CONNX Architecture 2
View from 10,000 Feet 2
A Typical Architecture Example 6
CONNX Planning 8
General Planning Concepts 8
Component Configuration 9
Service Configuration 10
Estimating Throughput Requirements 10
Estimating Component Requirements 12
Estimating Server Requirements 13
Estimating License Requirements 14
Planning Component Installation 15
Planning the SQL Catalog (CONNX CDD) 16
Scalability Concerns 17
CONNX Administration 18
General Administration Concepts 18
Development and Testing 19
Deployment 20
CDD Management 21
General CDD Management Concepts 21
Controlling Access 22
Updating Indexes 23
CONNX Performance 24
General Performance Concepts 24
Query Tuning 25
CPU 26
Memory 27
Input and Output 28
Disk I/O 28
Network I/O 28
Connection Pooling 28
CONNX Scenarios 29
General Scenario Information 29
Scenario 1: Local and remote CONNX servers 30
Scenario 2: .NET via SQL Server 32
Scenario 3: Unix Client 33
Scenario 4: Oracle and Java 34
Scenario 5: ASP .NET 36

Scenario 6: NTIER 37
Scenario 7: JBOSS and Hibernate on Linux 38
Scenario 8: JDBC Servlet and Websphere 39
Scenario 9: JDBC Web Services and Tomcat 40
Scenario 10: RCI Embedded COBOL 41
Scenario 11: RCI Embedded C 42
Index 43

Introduction

The CONNX Enterprise Planning guide is designed for CONNX product administrators. It can help them to develop their CONNX installation and maintenance strategy.

Development of a strategy is a very broad exercise. This document is intended to be relatively comprehensive; it provides guidance that is not included in the CONNX Installation Guide or the CONNX User Guide.

Managing data in the modern enterprise requires an effective combination of processes, people and technology. Interoperability, the ability of diverse systems or products to work together with minimal customer effort, is a critical property of any component in the enterprise. The problem would be simple if everyone used the same limited tool set under strict control but this rarely happens.

Legacy, mainframe, and non-relational databases continue to provide mission-critical functionality to organizations competing in the twenty-first century's fast-paced, international marketplace. In today's dynamic global business climate, leading-edge organizations require access to all their business information - no matter where that information resides. And if an enterprise relies on previous-era components, this adds to data management complexity.

CONNX provides a standardized interface to business data regardless of changes in the underlying database or structures. CONNX offers .NET or JDBC functionality to customer sites using both relational and non-relational data stores as if data access was being performed on a single relational database. CONNX enables you to present all your data to familiar standards-based tools on any platform. This flexibility gives you the freedom to choose the business intelligence and development solutions that are best for your business needs. CONNX can handle large amounts of data in many formats. It is easily adaptable to all environments thereby enabling organizations to scale from project to enterprise; from two-tiered to n-tiered; and from one to many data sources.

CONNX Architecture

View from 10,000 Feet

CONNX has positioned itself in the marketplace as "Simplified Data Access" for over 20 years But what does "Simplified Data Access" really mean?

- Real-time, read/write, SQL access to non-relational and relational data sources, using open data access standards (ODBC, OLE DB, JDBC and .NET).
- Connecting to multiple databases, using virtually any ODBC/OLE DB standards-based compliant tool/application.
- Integrating multiple disparate data sources no matter where they reside, giving a single SQL view of the data as if it exists in a single relational database.
- No disruption to existing systems or extensive programming.

This high-level "view" of CONNX helps to explain how it's done.

Here is a selection of relational and legacy ISAM/hierarchical data sources that CONNX supports:

RELATIONAL	DB2, SqlServer, Oracle, MySQL, Sybase, Informix

HIERACHICAL w/ Record Locking	Oracle DBMS

ISAM with Transaction support ISAM with Record Locking ISAM with File Locking ISAM with Journaling	ADABAS, RDB VSAM, CISAM DataFlex RMS

It is difficult to provide end users a consistent data access interface because different data sources have different external capabilities (such as record locking). A significant development effort would be required to extract a single value from both legacy systems data and direct modern relational API. CONNX provides end users a consistent interface to access their data no matter what the source, shielding the user from the complexity, development and quality assurance costs while providing consistent access methods and functionality across all the supported DBMS systems.

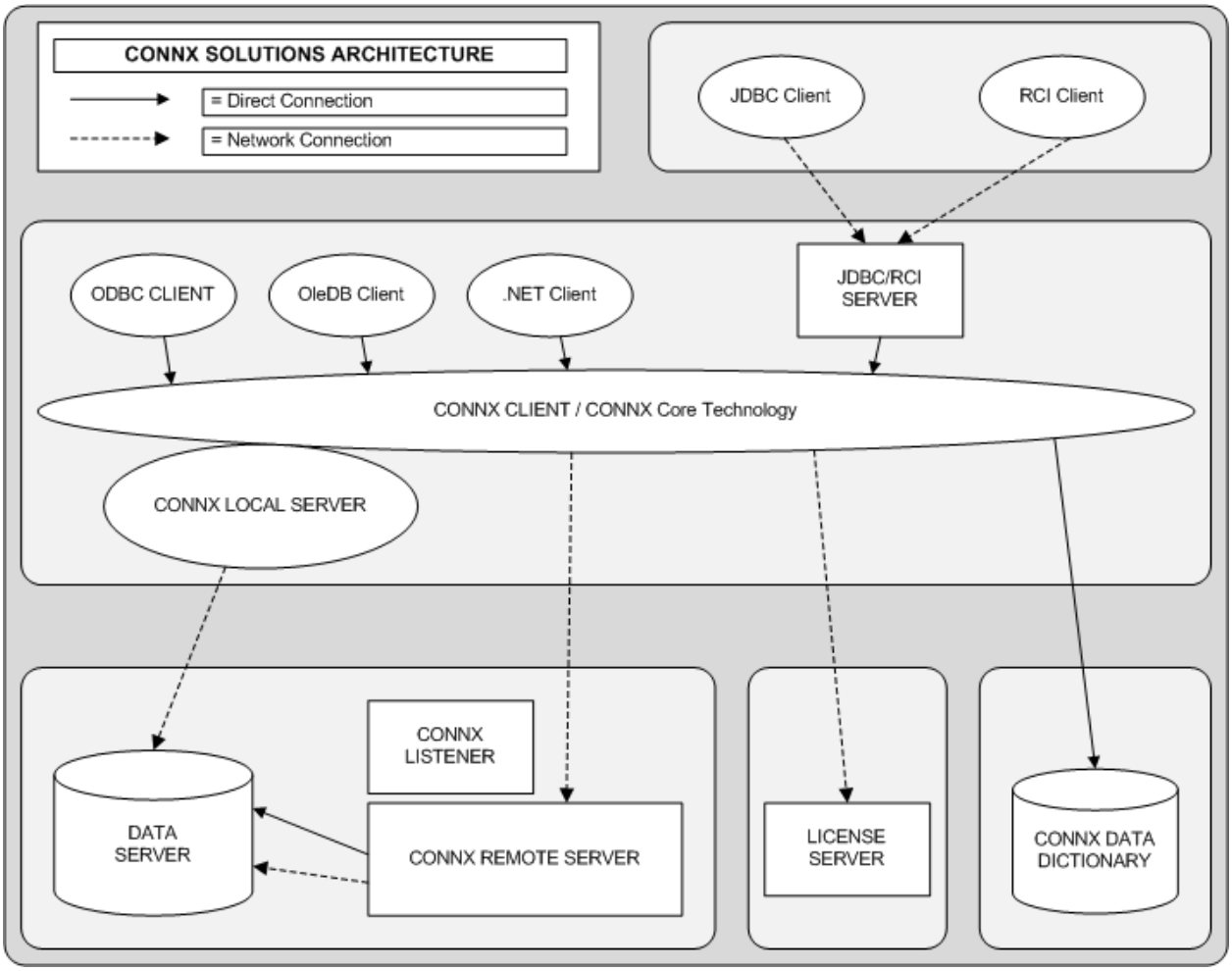


Figure 1. Organizational chart of CONNX Client and Server Components

As Figure 1 shows, CONNX uses a client/server based networked architecture. The CONNX client can be installed on every desktop in the organization or only on a select number of machines that serve data through a web interface or SOA application.

CONNX supports a variety of enterprise organizational models.

If all components reside on the same machine, it may be possible to use either a special in-memory pipe for faster communication, or the operating system loop-back device for ultra-fast communications.

CONNX supports the following access methods:

- ODBC
- OLEDB
- .NET
- JDBC
- RCI/ Embedded (currently only available to ADABAS users)

All the access methods will route through the client and then to a server which has the most direct contact with any back-end database.

A CONNX server may either be a process separate from the CONNX Client or it may be a DLL/library extension to the CONNX Client. Usually non-relational ISAM databases are implemented in a separate CONNX Remote server process and relational databases are implemented as DLL /library extensions to the CONNX Client. Any supported database with an API that services a network protocol will be implemented as a DLL/library extension to the CONNX Client. The workload is about the same whether using a DLL/library extension or a remote server .

The CONNX client and server perform the following distinct functions:

CONNX CLIENT	Data Conversion SQL to native translation Metadata lookups (requires CDD access) Query Plan creation Data Joining
CONNX SERVER	Physical Data Access Indexed data filtering Non-indexed data filtering Distributed Data Joining

The DLL/library extension workload is determined by the capabilities of the back-end database:

Relational data source	Filtering and physical data access is off-loaded to the relational database rather than being done by the CONNX client
CONNX Remote server process (usually a legacy ISAM database)	CONNX server is responsible for all of the work listed above in the CONNX server column.

The CONNX Remote server contains a CONNX Listener component, a separate process that spawns the correct type of independent server processes to service user requests. The CONNX Listener requires a minimal amount of system resources.

The JDBC/RCI server can be considered a CONNX client with a listener attached, with the additional functionality and protocol support needed by JDBC and RCI applications. The JDBC/RCI server supports multiple threads of execution and directly links to the end user's CONNX application if they share the same machine.

The CONNX License server service is a networked application that combines the elements of both a listener and a server. The CONNX License server requires a minimal amount of system resources.

The CONNX Client and CONNX License server service are available on Windows, Linux, Linux Z-Frame, AIX, Solaris, HPUX-RISC, and HPUX-Itanium. Because the CONNX CDD manager and license configuration utilities are only available on Windows, CONNX requires at least one Windows machine to create and manage CDDs and to install and configure CONNX licenses. Windows and UNIX have separate CONNX client configurations and configuration utilities.

A Typical Architecture Example

Figure 2 outlines a typical (moderately complex) customer site.

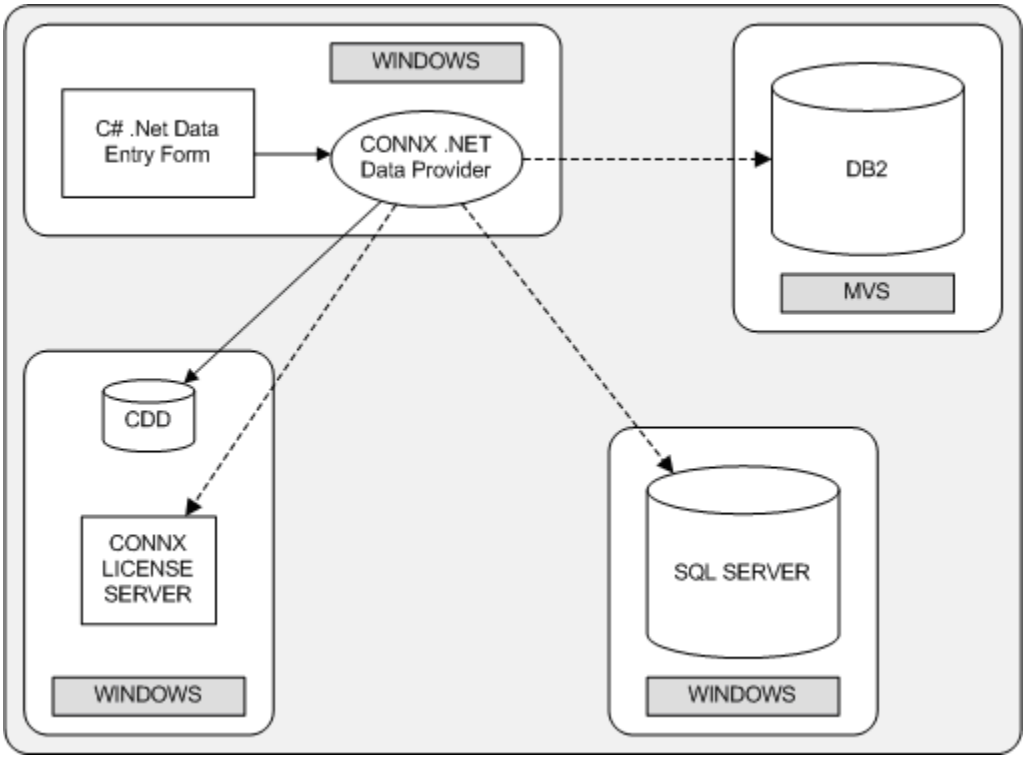


Figure 2. C# DataEntry form on Windows using the CONNX .NET Data Provider connecting to SQLServer on Windows and DB2 on MVS.

This typical scenario has four machines:

- The user application (C# DataEntry form) resides on Windows. The CONNX Client is installed on the user application machine.
- The DB2 database resides on a MVS machine
- The SQL Server database resides on a Windows machine
- The CONNX CDD and License server service reside on a Windows machine. This machine has a shared folder (with read/write access permissions) for CDD access.

In this scenario, the customer selected a machine to run the CONNX License server with a guaranteed uptime level. Due to the CONNX networked architecture, there are no CONNX specific components required on either database machine.

This diagram does not show the development configuration and any test and production environments. In those environments, the CDD may not reside on the same machine as the license server; if it did, each deployment phase might have its own directory.

This example illustrates some of the typical architectural and management challenges in a real-world scenario. This gives us a view of the architecture that we can refer to in the remaining sections as we

begin to unravel the complexity of Enterprise Planning. A comprehensive examination of typical deployment scenarios are outlined in the section entitled **CONNX Scenarios**.

CONNX Planning

General Planning Concepts

A successful configuration plan contains reasonable platform, network and software component forecasts for the production environment.

The CONNX Planning section provides the criteria for reviewing project scope, making initial resource and cost estimates and defining the hardware and software components needed to support a CONNX product install. This section also defines criteria for preparing the development and test environments and deploying the CONNX product family to production.

Component Configuration

When planning the CONNX install service configuration, here are some considerations:

1. The CONNX Client must be installed on every machine that accesses data. This may mean many CONNX Client installs for a desktop application or only a few installs for organizations using distributed or web service framework. Each configuration has its particular installation, administration and maintenance constraints.
2. The CDD repository location is very important. The CDDs must be available to every client machine; you may need to open a share on a file server and set appropriate access permissions. You may need multiple CDD copies (the development, test and deploy cycles may each require a separate copy); this can increase administration complexity.
3. A CONNX Remote server must be installed on remote hosts when a local server DLL cannot access data. You may have to open up a range of firewall ports and coordinate across departments to gain access to a particular machine.
4. The CONNX License server must be installed on a single server that has access to the LAN. Choose a dedicated server to ensure uninterrupted access to CONNX features.

The following discussion will be broken down into the constituent aspects of a typical CONNX installation. Please refer to Figure 1 (Organizational chart of CONNX Client and Server Components) throughout the discussion to help facilitate a clearer understanding.

Service Configuration

Estimating Throughput Requirements

For a graphic representation of the CONNX architecture, see Organizational chart of CONNX Client and Server Components.

In order to estimate CONNX throughput, you must know how much bandwidth is available and how much bandwidth is being used.

CONNX is designed to reduce the amount of data traveling across the network. Most database servers have excellent data-bus bandwidth to move data quickly from the disk into resident memory and CONNX takes advantage of this speed. CONNX is able to tune queries which can result in greater speed.

After you determine the system requirements for your federated server environment, to plan the throughput requirements requires a plan for each of the components (including the CONNX client machines, each of the database and remote servers, and the CDD). This can be challenging because you must consider:

- the possible configurations' complexity
- the autonomous nature of the different remote data sources
- how great the throughput workload will be. This is even more difficult if you do not have CONNX currently installed.

No matter how large the available bandwidth, data can only flow as fast as its slowest component.

To estimate bandwidth usage, calculate the number of bytes per query and the processing power per query.

- Bytes per query indicates how much data is flowing across the network and how much temporary storage is required to hold a result set while it is being processed or consumed on the client or the server. Data can be stored either in resident memory (which is orders of a magnitude faster than disk access) or in temporary disk files. The greater the amount of data that must be stored in temporary disk files, the greater the risk of disk utilization. And as the amount of data that flows over the network increases, the amount of time required to return the result set to the client increases as well.

  Bytes per query can be calculated with some accuracy depending on the CONNX application environment. The following are responsible for most of the bandwidth usage:

  - Number of columns,
  - summed bytes per column,
  - number and cardinality of tables involved, and (ultimately)
  - number of rows returned, contribute to bandwidth usage.

  In an ad-hoc query environment, hard estimates are unlikely as opposed to estimates in a tightly controlled production application environment where defined throughput statistics are possible.

- Processing power per query indicates how computationally expensive a particular result set is to produce. To measure processing power per query, use the query plan; reading the query plan and tuning queries for performance is beyond the scope of this document.

Important: Query tuning is crucial to reducing processing power use. CONNX will perform exactly as your queries ask it to perform; if queries are not tuned for performance, processing power will suffer.

General Guidelines for Estimating Throughput Requirements

1. Timing information is critical to any estimation of throughput. If you cannot gain access to low level code to isolate the bottleneck, make an adjustment to the query parameters and get a benchmark estimate by contrast. When you add timing information to an application, be careful as it can interfere with query execution.

2. Monitoring a variety of targeted queries with some degree of control over byte width and query plan will provide a much clearer view than merely selecting all of the columns and all of the records from a single table.

3. A query that can be held entirely in resident memory will be faster than one that has to use temporary disk storage. Unless your environment has the appropriate amount of production hardware, hundreds of concurrent users will dramatically increase disk utilization. Reduce the number of columns and number of result set rows wherever possible in a production environment.

4. Select the appropriate reporting intervals for effective throughput estimation. Effective estimation means filtering out the extreme utilizations and focusing on the median or second highest utilizations, depending on the degree of risk tolerated in sizing estimates. Estimation accuracy depends on gathering accurate metrics over a representative (large enough sample) interval, and being able to project a future workload that includes the number of concurrent users, the frequency of existing queries, and the profile and frequency of new queries.

5. Primary decisions about which records match the criteria of a SQL query request are made on the server. The order of tables processed in a join is decided on the client as are other activities like data conversion and sorting. These CONNX core design elements can influence the choices made in allocating hardware or in placement of components.

Estimating Component Requirements

For a graphic representation of the CONNX architecture, see Organizational chart of CONNX Client and Server Components.

Install the CONNX client component on any machine that will retrieve data.

For standalone desktop applications install the CONNX client on any machine where the desktop application is installed.

For distributed architecture applications (multiple machines retrieve data to be viewed on a user desktop) such as Web applications, there are two common scenarios:

- Each Web server running server side scripts that access data would require its own copy of the CONNX Client.
- The Web server is connected to an SOA service that runs on a separate set of machines; each of the separate set of machines would require its own copy of the CONNX Client.

Each machine running a standalone application requires its own copy of the CONNX Client.

When there is a distributed or web application, choose the location of the CONNX Client based on performance requirements. Even though the CONNX core offloads much of the work to the database or remote server, the entire returned data set may have to reside on a client machine. If there are hundreds of users connecting to a web server simultaneously accessing data from that same machine through CONNX then hundreds of data sets may be called into memory on the same machine. This may also include temporary tables required for multi-table joins and sorting.

Estimating Server Requirements

For a graphic representation of the CONNX architecture, see Organizational chart of CONNX Client and Server Components.

Because the CONNX Listener automatically spawns CONNX Remote servers to handle incoming requests, you may not need multiple listeners per machine.

The CONNX Remote server is normally located on the database machine because CONNX Remote servers are tightly coupled with the database. or in the case of some ISAM storage methods, the file system. ADABAS is the only exception; if you have installed Entire-Network, the database and the CONNX Remote server may physically reside on different machines.

Estimating License Requirements

For a graphic representation of the CONNX architecture, see Organizational chart of CONNX Client and Server Components.

License usage for the CONNX product family is based on database access. Each database defined in a CDD represents a separate connection (no matter what database type) and requires one license seat per back-end type (even if the database definition is actually a clone of an existing database) when a connection has been made to the CDD. Connection count may be limited to a subset of defined databases within the CDD by using the **application** field on the CONNX Integrated Logon screen or during the CONNX Data Source setup. More information on the use of the **application** feature can be found in the CONNX User Guide.

For web server or distributed access you can purchase an NTIER license for each server; this allows unlimited connections to that server. NTIER licensing is appropriate for web based enterprise applications or applications distributed in a large-scale SOA project.

You may wish to consider purchasing redundant or fail-over licenses so that you have a primary and a back-up license server; this can help ensure uninterrupted access to CONNX features. It can also make sense to purchase separate licenses for your development and testing systems as well as your production system.

Planning Component Installation

For a graphic representation of the CONNX architecture, see Organizational chart of CONNX Client and Server Components.

As you install the CONNX component installation across the enterprise, you may have to coordinate with other departments.

Server or mainframe administrators may have to install the CONNX Remote server component; network administrators may coordinate the CONNX Client rollout. Depending on your organization's environment, you may need to plan for a phased CONNX component rollout - first to a test environment and then from test to production.

The CONNX API is used to communicate between the CONNX Client and the CONNX Remote server; it determines the compatibility between client and server components. Compatibility breaks occur when an incompatible version is installed.

- The CONNX Client is backwards compatible
- The CONNX Remote server is not forward compatible.
- For JDBC users, the client jar is not forward compatible but the JDBC/RCI server is backwards compatible.

For the best performance and feature quality, the CONNX Client and the CONNX Remote server should be on the same version level, although this is not a system requirement.

Planning the SQL Catalog (CONNX CDD)

In CONNX, the SQL Catalog is called the CONNX CDD.

The CONNX CDD is the cornerstone of the CONNX product family. It contains the metadata that describe all of the organization's enterprise data.

Your installation can have multiple CDDs. You might create one CDD for the development environment databases, a second CDD that is an exact clone of the development CDD, and a third CDD containing an entirely different set of databases for your production environment. An organization usually has a collection of ad-hoc CDDs it uses for off-line development and testing along with a master CDD it uses in production.

The CONNX Client determines the current view of the CDD. The current CONNX Client connection contains the current view of the CDD. If the CDD is modified while this connection is running, the CONNX Client will not recognize the changes until a new connection is spawned.

Keep the production and development CDDs that you consider project resources in a controlled access environment under a revision control system such as a source repository.

The CDD is the CONNX sole point of entry to data. Only the CONNX CDD Manager can create the initial database connection required for data access. Determining where to install the CDD Manager is as important as deciding where to keep the CDDs.

When you enable connection pooling (instead of reusing an existing connection), CONNX creates a fresh connection for any altered CDD so only the current CDD version is the metadata repository connection.

CONNX CDDs may also be linked together in a parent/child relationship.

Scalability Concerns

Scalability is an integral part of the CONNX design and architecture. Not only can CONNX continue to function effectively and efficiently when the environment is resized, but CONNX can take full advantage of the added resources.

The heart of CONNX is a distributed SQL engine. The CONNX SQL engine takes advantage of both the client and the server resources, and is designed to reduce the amount of data traveling across the network.

With its distributed architecture, CONNX can take advantage of available client CPU cycles, freeing server resources to handle other tasks. CONNX's collaborative design unites the best client and server features and increases overall query performance.

CONNX Administration

General Administration Concepts

This section discusses CONNX in the classic development/test/deploy/administrate cycle.

Promoting an application from a development environment to an enterprise production environment requires:

- planning
- preparation
- coordination
- testing

In this multi-step process, the application:

- is deployed into one or more intermediate environments where it is integrated with other applications
- undergoes rigorous testing
- runs in configurations that are scaled up or in which application resources are increasingly distributed.

CONNX provides world class technical support to help you with your enterprise-wide project plans. Contact your sales representative for more information.

Development and Testing

To ensure a smoother transition to your production environment, first integrate CONNX into your development and testing environments.

Having CONNX integrated into your separate development and testing environments will help your enterprise. In most cases, you can identify problems and needed feature requests without impacting your production environment. If you believe there is a problem in the product or in the application and you need technical support's help, it is easier to isolate the problem to a simple reproducible example in a non-production environment.

CONNX uses standard APIs which minimizes the CONNX learning curve. CONNX assumes that you are familiar with ODBC, JDBC, OleDB and .NET. You will find the ODBC, JDBC, OleDB and .NET documentation for your environment extremely helpful if you run into problems.

The development and testing environments are usually isolated from the production environment, and often from each other. Each isolated environment requires its own CDDs and its own set of CONNX licenses.

CONNX uses the CDD as a metadata repository to make different data sources appear as one. Treat the CDD like code and add it to a source repository and back it up on a regular basis. If CDD changes break an application you may need to roll back to a previous CDD version.

Deployment

When you've finished testing, you can deploy the final application to the production server. This may require coordinating and scheduling time with several departments.

Whenever you deploy to a new machine on a separate or different environment, be careful of the operating system differences that can skew results. Here are some examples of potential problems:

- If the testing UNIX and the production UNIX systems have different locale settings, the application may not perform at all, or it may produce different results. Because of environment differences, the same system library search path settings on one UNIX machine will produce correct results but on another machine will produce the wrong results.
- Different Windows systems can have different directory permissions.
- Different mainframe environments may require technical resources or administrative scheduling to install the server in the correct data set or library.
- VMS test area installations could require the individual install option; whoever install this must understand DCL .

CDD Management

General CDD Management Concepts

The CDD is a snapshot of the metadata at import time. Over time, the CONNX tables structures will change. This section describes several strategies you can use to manage table structure changes in a controlled fashion.

If your organization has discrete development, test, and production environments, maintain three separate data dictionaries for each environment. This isolates each environment from each other; you can test any changes in one environment without affecting the others. When it comes time to propagate changes from development CDD to test CDD, or from a test CDD to a production CDD, there are several techniques.

- If there are no changes to the metadata (exactly the same between the environments), the CDD file can be copied from one name to another.

  Note: This usually occurs when the database or server name changes, but nothing else has changed

  Use the CDD manager to change the name of the database and/or server by selected the red database cylinder and making the appropriate changes. The exact metadata tested in one environment will be used in the next environment.

  This option will require some downtime, as any connections to the CDD will need to be closed before a new copy can replace the old. Shut down the JDBC Server and any client connections to the data dictionary. This is typically done during a nighttime batch window or a planned weekend outage.

- If the metadata between the two environments are different enough that a CDD file copy will not work, use the data dictionary manager to re-import the changed tables.

  This will require some downtime, as any connections to the CDD (such as the JDBC Server or any client connections to the data dictionary) will need to be closed before a new copy can replace the old version. This is typically done during a nighttime batch window or a planned weekend outage.

- For some data adaptors, such as ADABAS, the metadata can be changed by using DDL SQL statements, such as **Create Table Description** or **Create Table Cluster Description**. Using these DDL statements, old metadata definitions can be removed, and new definitions created. Since the DDL statements can be issued using a variety of interfaces to CONNX (JDBC, ODBC, .NET, OLEDB), there is no downtime required for this type of metadata update. The CDDs can be updated while online and in use, simply by using the required SQL statements.

Controlling Access

Some care should be exercised with the modification of a production CDD. To prevent unwanted modification of the CDD contents via the CONNX CDD manager, use the security measures outlined in the CONNX User Guide and the CONNX Installation Guide .

Controlling physical access to a CDD may be a concern. A CDD must be made available to any CONNX client that will be accessing data. The CDD is actually a database, and read/write access is required to the directory that contains the CDD, and the CDD file itself.

The following is a list of major security considerations:

1. Access to the Data Dictionary Manager
   The Data Dictionary Manager can only be installed when someone is in possession of a CONNX License file. Without access to a CONNX license file, the installation routine will not make the CONNX Data dictionary manager available. Minimize the number of individuals who have access to the CONNX data dictionary manager by physically securing the license files provided.

2. Securing created Data Dictionary files with file server security mechanism.
   Typical CDD files reside on a central network share so all users can access the same CDD. Secure this file share so that only authorized users have access to the CDDs.

3. Securing Data Dictionary Files via internal password
   Data Dictionary Files are encrypted for security reasons. You can also password protected Data Dictionary Files. If you set a password on the data dictionary, do not lose it as you will not be able to modify the CDD without it.

The tables, views and other SQL entities within the CDD can be secured a variety of ways. Please refer to the CONNX User Guide for more details.

Updating Indexes

In many cases the structure of the columns in the table remains intact, but indexes are changed, added or removed.

Index metadata can be refreshed on any table by using the **Refresh index** button in the data dictionary manager, without performing a complete re-import of the object metadata.

Index information for the complete data dictionary can be refreshed by using the **Update Statistics** option in the data dictionary manager. This is equivalent to selecting **Refresh index** on every table in the CDD.

The **Refresh Index** and **Update Statistics** functions are only available via the CDD Manager.

CONNX Performance

General Performance Concepts

CONNX has been developed with performance in mind and strives to maintain the highest level performance in the marketplace.

Getting the highest performance is most important in applications that require fast user response. For any application that relies on multiple machines or resources, system performance is only as good as the slowest component.

Because operating system and networking environments vary, we are unable to provide a performance calculator or hard statistics that would work for all our customers. Developing and monitoring performance for a particular installation is best performed on-site in your specific environment. Customer can modify their current environment so that CONNX can achieve peak performance.

This chapter gives general performance improvement guidelines. Use the remaining sections for high level planning. We will detail some specific concepts and methods that can be employed to increase or maintain CONNX performance.

Query Tuning

SQL query tuning is an essential performance component.

Because query tuning specifics depend on the underlying database structure, this document cannot provide a comprehensive examination of query tuning. Your database documentation should contain specific query tuning information; there are also many third party books on SQL query tuning available for purchase.

Much of the specific database information on SQL query tuning is also applicable to CONNX. For example, the natural relationships between sort order and index selection can help you decide when and how to use an ORDER BY clause. And understanding the relationship between table cardinality, index selection, join order, and usage of various operators is essential to understand SQL query performance.

CONNX provides access to the query plan via the {**statistics**} function. Please refer to the CONNX User Guide for more information.

CPU

Even though multi-processors and memory are relatively inexpensive, CPU speed can still be a problem.

Raw computing power may be a problem depending on the application and expected result set size. Some CONNX operations (like sorting and aggregate operators) may be CPU intensive. Although CONNX core technology uses optimized algorithms, some operators must access each record no matter how large the data set.

Many organizations have multiple intensive applications running concurrently on a single machine. If you reduce the number of intensive applications concurrently running on a single machine (by adding more hardware to the network) processing should speed up.

On average, the CONNX remote servers use less CPU then the client. You can minimize some of the computing cost if you offload processing to the backend database. This is an internal CONNX design feature.

The CONNX Remote servers tend to be installed on legacy systems. There are many specific legacy CONNX hardware configurations such as RMS, DBMS and RDB running on VAX hardware running OpenVMS. Remember that the fastest VAX is much slower than today's average PC.

In some cases it may be possible to cluster machines or to use a Network File System implementation such that the data on a slow machine can be accessed from a much faster machine.

In a later section we will examine local versus remote ISAM access.

Memory

Random Access Memory (RAM), or memory, can cause performance problems if application memory requirements exceed the RAM capacity. This will cause the results to be swapped to disk.

Because the CONNX Client may be required to hold several record sets in resident memory to complete join or sorting operations, it has the largest memory requirement. To minimize memory usage when the result set is large enough, use disk based algorithms that sort in-memory pointers to the data.

The CONNX Listener uses very little memory and should never dominate resident resources.

CONNX Servers will batch record sets and may hold a number of resources they need to maintain the source database in a cursor state.

Input and Output

Disk I/O

Disk access has traditionally always been the bottle neck for database related systems. This includes retrieval and updating of physical data, storing temporary result sets and logging. For more information, see Estimating Throughput Requirements.

Network I/O

Usually local LAN access with Ethernet is not a bottleneck.

The system loopback device is an internal CONNX feature. If you use the system loopback device, it can speed up component communications. If you plan to use the loopback interface, install the components locally.

Depending on your application's scale, you may get the best results if you create a distributed setup where the database resides on a different machine than the CONNX data access components.

Due to system resource thrashing, overall performance may decrease if too many applications are competing for system resources.

For more information, see Estimating Throughput Requirements.

Connection Pooling

To speed up establishing a connection, enable connection pooling. Connection pooling is much faster because many of the internal objects needed to facilitate a connection have been allocated and initialized. The JDBC/RCI server has connection pooling enabled by default.

The CONNX Remote server is spawned for the life of a connection. Normally, when the connection ends, the remote server drops away. For pooled connections, the remote server does not drop away but none of the original SQL processing context remains.

For more information on connection pooling, refer to the CONNX User Guide.

CONNX Scenarios

General Scenario Information

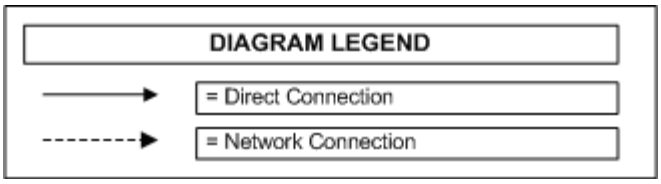
In the following scenarios we will outline some typical customer environments.

These examples show how you can design and administer CONNX using a variety of client, sever and database platforms. These are not meant to be an exhaustive examination of possible architectural configurations.

We are giving you templates for what is possible. There are few limits to the number of ways in which CONNX is employed in the enterprise. You can infer more complex (or less complex) scenarios from these relatively simple examples.

Most examples connect to two data sources, one local server and one remote server to establish a sense for each type of connectivity.

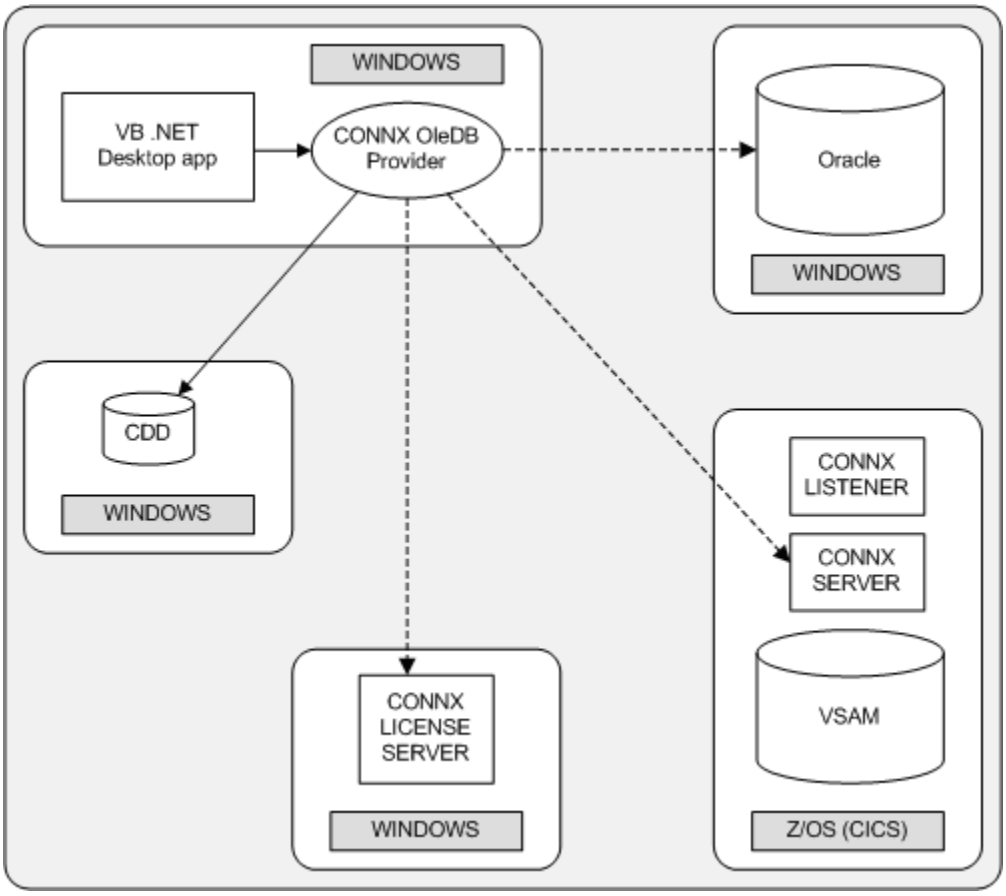
Scenarios use this diagram format



Scenario 1: Local and remote CONNX servers

General Scenario Information contains useful information about the scenarios.

This scenario shows a distributed VB .NET desktop application using OleDB to connect to both Oracle on Windows and to VSAM on Z/OS (CICS.)



The desktop application uses both local and remote CONNX servers. The CDD is located on a different file server than either the CONNX Client or the CONNX License server.

If your CDD is being shared by multiple .NET applications, pay special attention to security. If the CDD configuration and the file server directory have the wrong security settings, your applications won't have the correct CDD access.

The license server can be installed on its own machine. Doing so could lower the application downtime risk.

The VB .NET distributed desktop application uses OleDB to connect to both Oracle on Windows and VSAM on Z/OS (CICS).

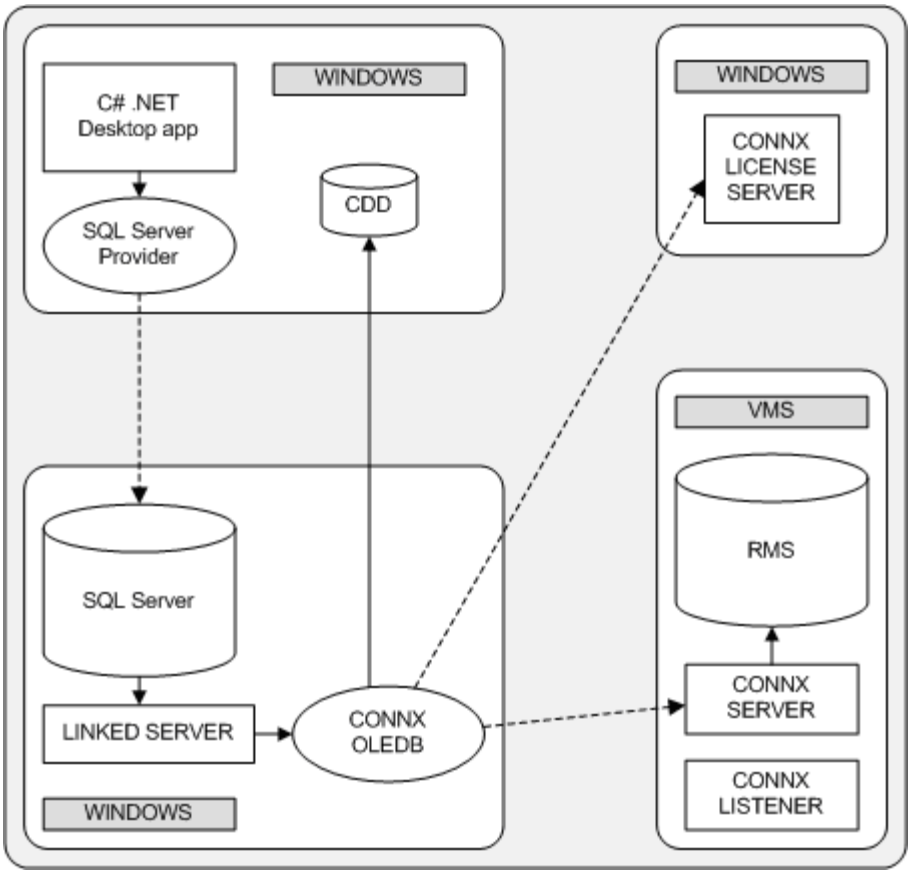
The CDD is on a file server that is accessible to the CONNX Client and the CONNX License server is on a separate machine.

Your organization's CDD may be shared by multiple NET application instances. Providing the necessary security provisions for the CDD configuration and for the directory share on the file server will ensure correct access to the CDD. The license server need not be installed on a separate machine although you may choose to do so to guarantee machine uptime.

Scenario 2: .NET via SQL Server

General Scenario Information contains useful information about the scenarios.

This scenario shows a NET application using the SQL Server Provider connecting to RMS on VMS via a SQLServer linked server and SQLServer on Windows. This .NET application indirectly uses CONNX via SQL Server linked server technology.



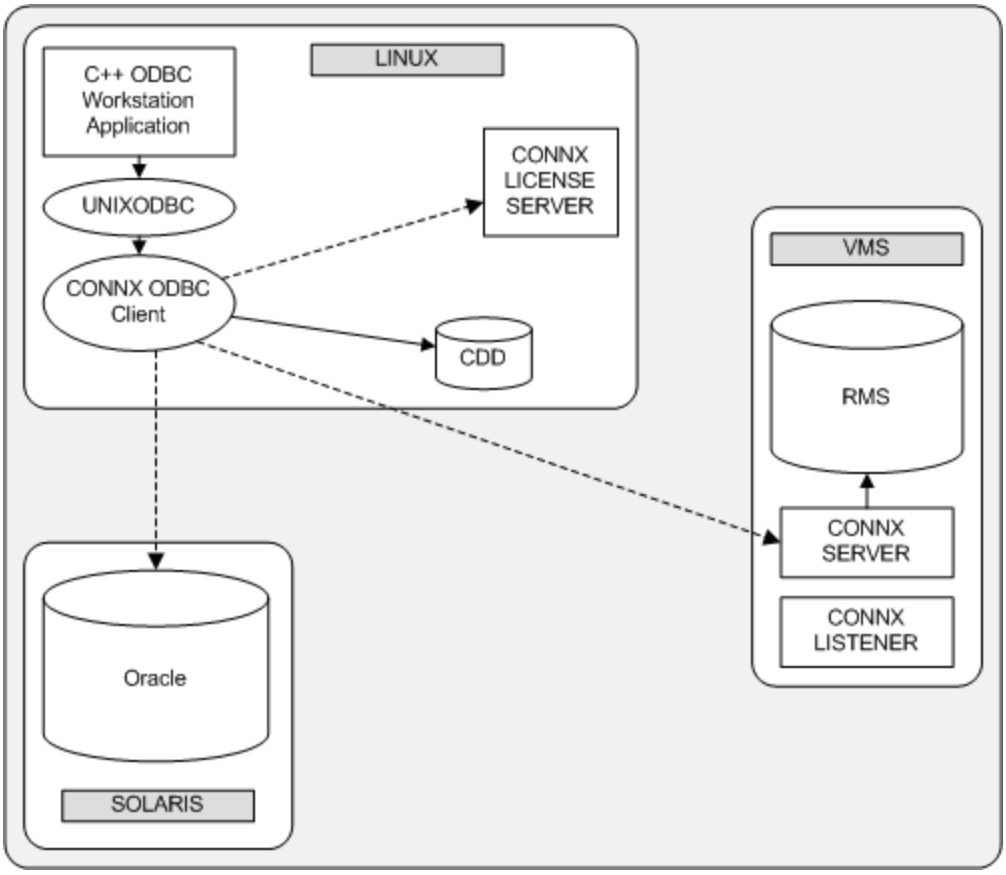
With CONNX, a legacy data source is accessed as if it were just another relational table. The user can create stored procedures and other routines as if the disparate data sets all belonged to the same database. CONNX provides the seamless connectivity and access to the legacy data source.

In this scenario, the CDD is on the user's machine. This is typical in a development environment where CDD changes are stored on the user's local machine and propagated to the server when development is complete. In the production environment, move the CDD closer to the CONNX Client or to a dedicated file server.

Scenario 3: Unix Client

General Scenario Information contains useful information about the scenarios.

This scenario shows a C++ ODBC Application using UNIXODBC on Linux connecting to Oracle on Solaris and RMS on VMS.



There are no Windows machines in our configuration diagram. When you need to configure the license server or the CDD, there must be at least one Windows machine available. It can be removed when configuration is complete.

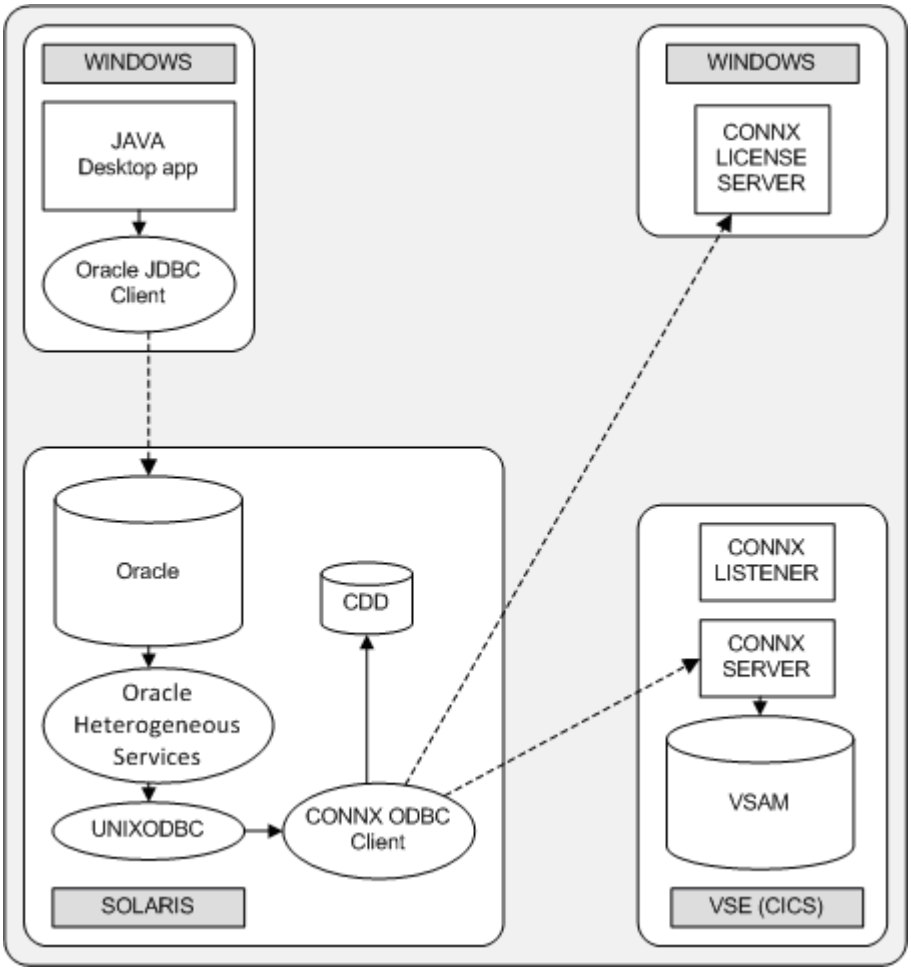
UNIXODBC is the driver manager in our scenario. If the driver configuration manager allows it, any available ODBC compliant driver manager should work.

In the example, the license server runs on the user application machine. This is a very common configuration. Only one CONNX License server will be installed on the LAN.

Scenario 4: Oracle and Java

General Scenario Information contains useful information about the scenarios.

This scenario shows a JAVA standalone application on Windows using the Oracle JDBC Client connecting to Oracle on Solaris and accessing VSAM under VSE (CICS) via Oracle Heterogeneous Services.



This Oracle Heterogeneous Services example in this scenario is analogous to the SQL Server linked servers example outlined in a previous scenario.

The Java desktop application uses the Oracle JDBC driver to connect to an Oracle database and access both Oracle and VSAM data.

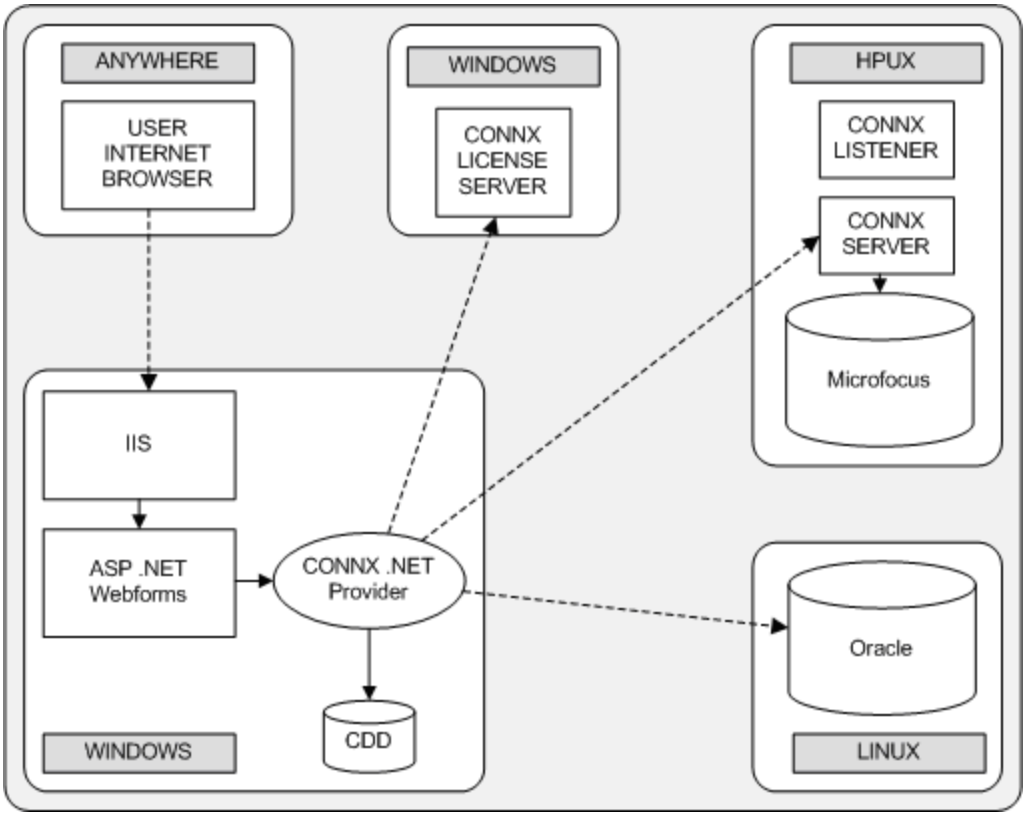
In this scenario, a legacy data source is accessed as if it were just another relational table. The user can create stored procedures and other routines as if the disparate data sets all belonged to the same database. CONNX provides the seamless connectivity and access to the legacy data source.

Store the CDD where the CONNX Core Client technology can access it; in our scenario, the CONNX client and CDD both reside on the Solaris machine.

Scenario 5: ASP .NET

General Scenario Information contains useful information about the scenarios.

This scenario shows an IIS ASP .NET application (webforms) connecting to Oracle on Linux and Microfocus on HPUX.

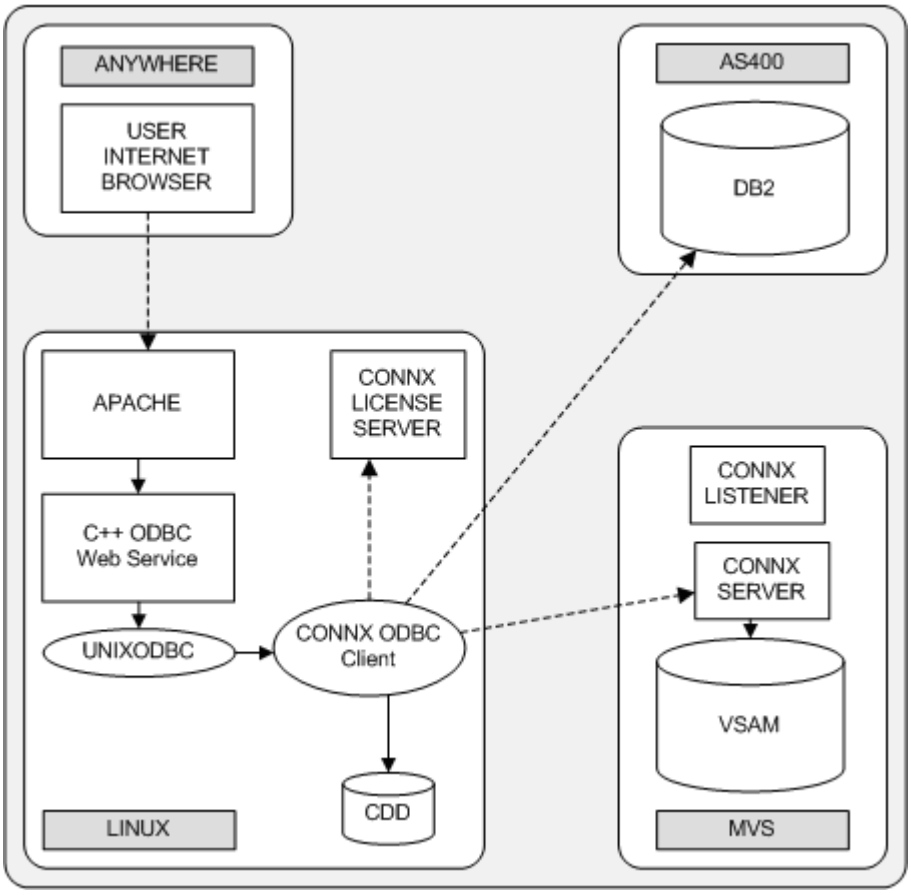


In this scenario, a legacy data source is accessed as if it were just another relational table. The user can create stored procedures and other routines as if the disparate data sets all belonged to the same database. CONNX provides the seamless connectivity and access to the legacy data source.

Scenario 6: NTIER

General Scenario Information contains useful information about the scenarios.

This scenario shows a C++ ODBC Web Service Application using UNIXODBC with Apache on Linux connecting to DB2 on AS400 and VSAM on MVS.



This architectural layout describes a web or NTIER application. The C++ ODBC Web Service uses NTIER because NTIER had been enabled in this machine's sqlregistry (CONNX's configuration repository for UNIX systems).

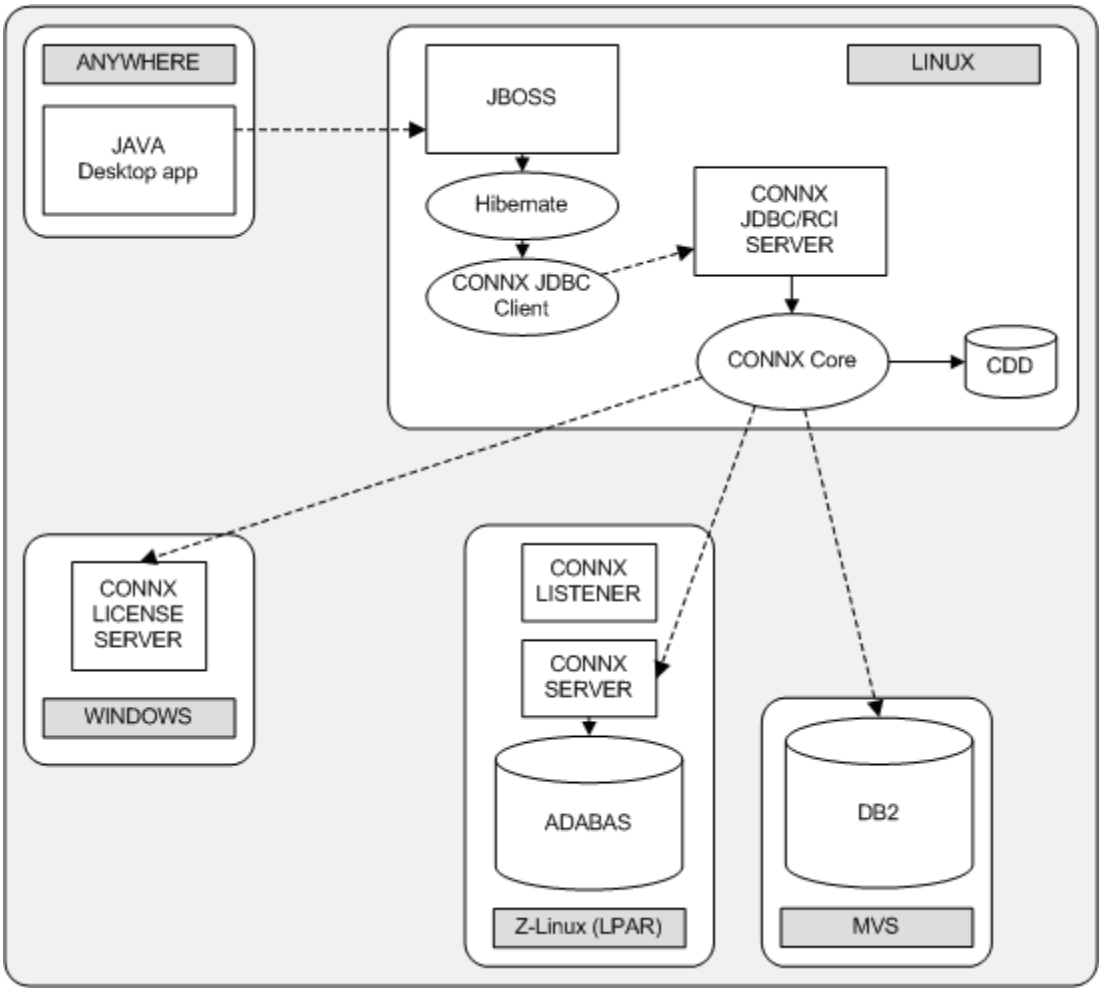
NTIER gives the end user unlimited connections from a designated client machine. Each NTIER license is single machine specific.

The NTIER license is ideal for web server use because it allows unlimited connections, although only one session can be active at a time. If you want to have multiple active sessions, consider purchasing multiple NTIER licenses.

Scenario 7: JBOSS and Hibernate on Linux

General Scenario Information contains useful information about the scenarios.

This scenario shows a JBOSS application using Hibernate on Linux and the JDBC/RCI server on Linux connecting to ADABAS on Z-Linux and DB2 on MVS.



CONNX supports Java connectivity to any CONNX Client platform.

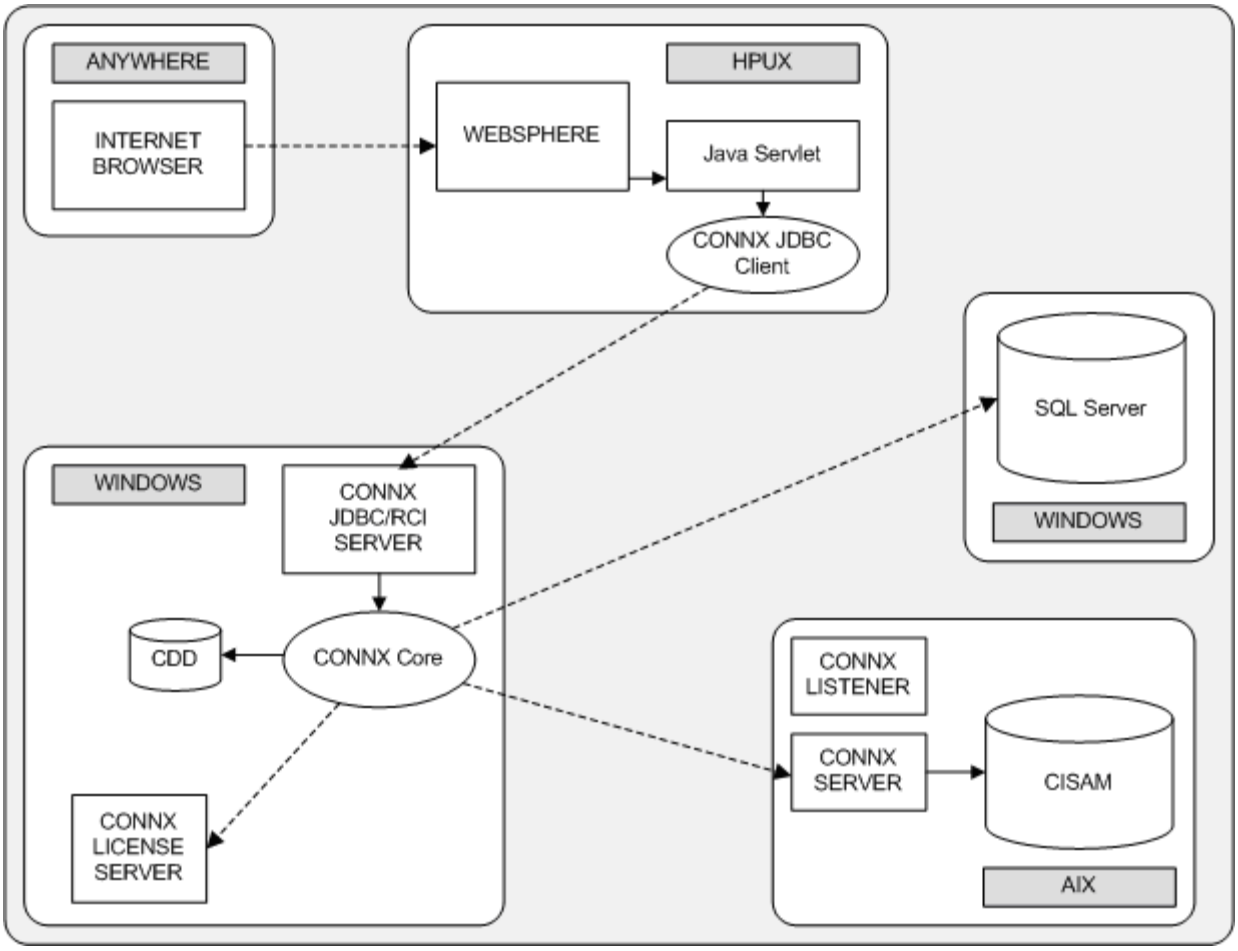
CONNX for JDBC is a Type 3 networked driver. Java access to CONNX is supported from any platform that supports Java version 1.3 and above, including VMS.

Our scenario uses Hibernate within JBOSS, a popular open-source Java Service Container. The CONNX JDBC Client jar and the JDBC/RCI Server are both located on the same machine. This allows for very fast communication between client and the JDBC/RCI middleware via the system network loopback interface.

Scenario 8: JDBC Servlet and Websphere

General Scenario Information contains useful information about the scenarios.

This scenario shows both a JDBC servlet application using Websphere on HPUX and the JDBC/RCI server on Windows connecting to SQLServer on Windows and CISAM on AIX.

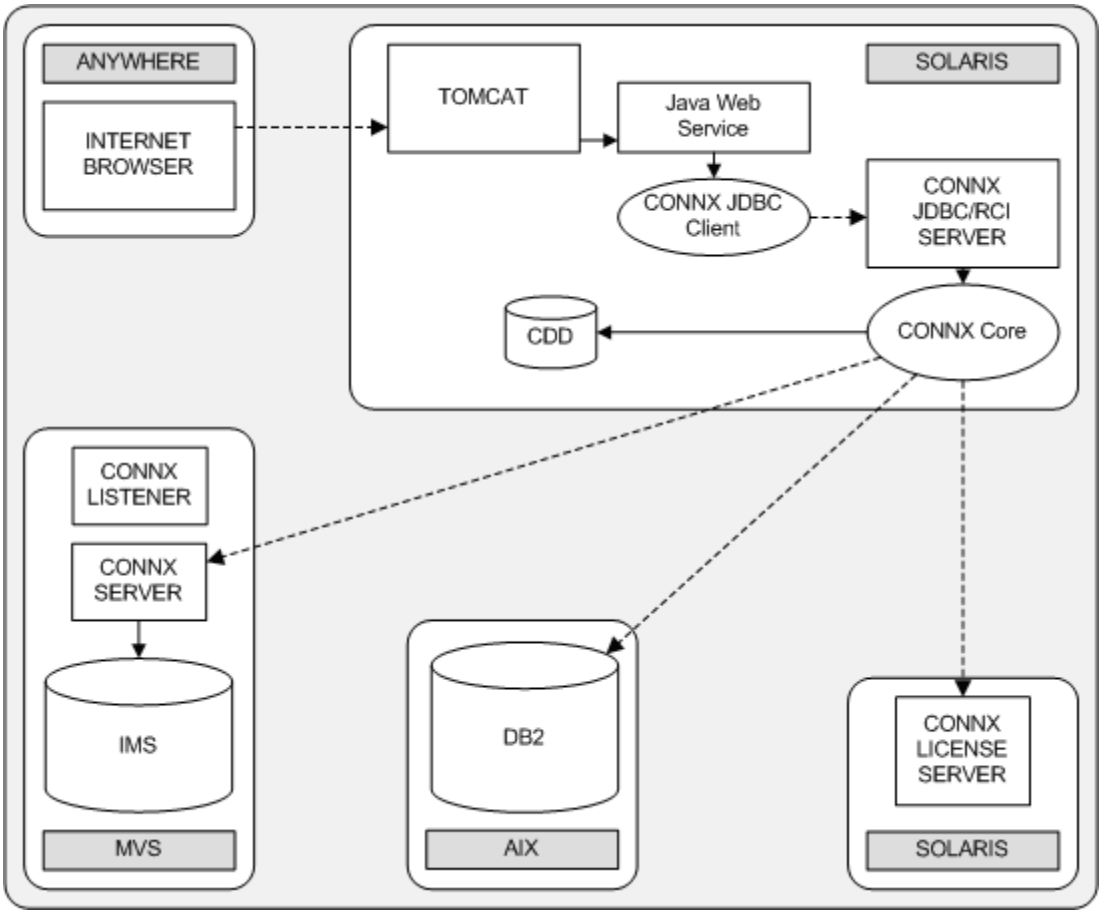


IBM WebSphere is used as a Java Web Container to host a Java Servlet. The CONNX JDBC jar can communicate over the network to a CONNX JDBC/RCI server running on the same computer as a CONNX Client.

Scenario 9: JDBC Web Services and Tomcat

General Scenario Information contains useful information about the scenarios.

This scenario shows both a JDBC web service application using Tomcat on Solaris and the JDBC/RCI server on Solaris connecting to DB2 on AIX and IMS on MVS.

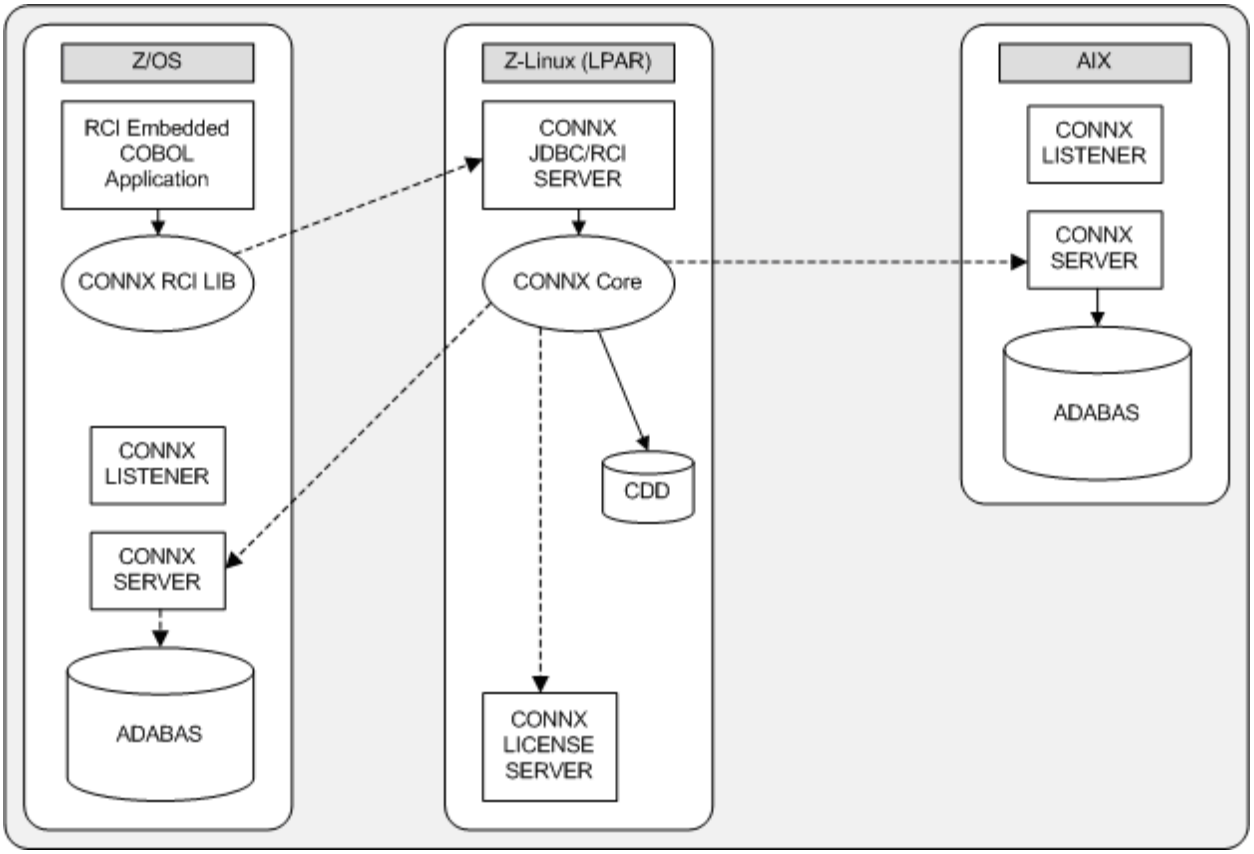


In this example, Tomcat, a popular open-source implementation from Apache, is used as a Web Container.

Scenario 10: RCI Embedded COBOL

General Scenario Information contains useful information about the scenarios.

This scenario shows both a RCI Embedded COBOL application on Z/OS and the JDBC/RCI server on Z-Linux in another LPAR connecting to ADABAS on Z/OS and ADABAS on AIX.



The CONNX RCI for Embedded SQL is only available to CONNX for ADABAS customers.

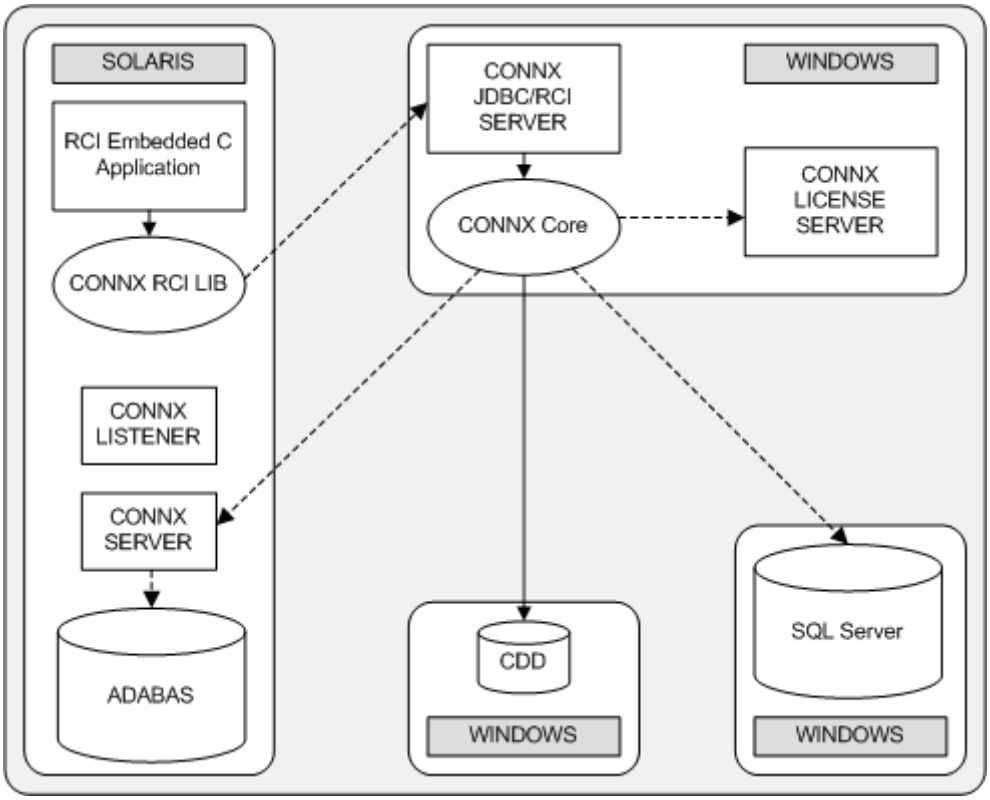
In this example we are running Z/OS in one VM LPAR and Z-Linux in another VM LPAR so they are considered to be physically located on one machine. We are connecting to ADABAS data that resides on both Z/OS and AIX.

We are also connecting to ADABAS data via a COBOL application that has embedded SQL statements that have been precompiled for the RCI. This technology allows for fast database development utilizing 3GL languages.

Scenario 11: RCI Embedded C

General Scenario Information contains useful information about the scenarios.

This scenario shows both a RCI Embedded C application on Solaris and the JDBC/RCI server on Windows connecting to SQL Server on Windows and ADABAS on Solaris.



The CONNX RCI for embedded SQL is only available to CONNX for ADABAS customers.

In this example we have a C application that has embedded SQL statements that have been precompiled for the RCI. Our C application can access ADABAS and SQL Server data joined together as if the data was from the same homogenous database.

Index

3

3GL....41
  utilizing....41

A

access....9, 14, 22, 30, 34
  CDD....22, 30
  CONNX....9, 14, 22
  Controlling....22
  Data Dictionary Manager....22
  LAN....9
  VSAM....34
ADABAS....2, 13, 21, 41, 42
  connecting....41
  CONNX....41, 42
ADABAS on AIX....41
ADABAS on Solaris....42
ADABAS on Z/OS....41
ADABAS on Z-Linux....38
AIX....2, 41
Although CONNX Core....26
and/or....21
Apache....40
Apache on Linux....37
API....2, 19
ASP....36

B

backend....14, 26

C

C# DataEntry....6
called....16
  CONNX CDD....16
CDD....2, 6, 9, 10, 14, 16, 19, 21, 22, 23, 30, 32, 33, 34
  access....22, 30
  contains....22
  keep....16
  move....32
  Store....34
  Treat....19
  Use....21
  view....16
CDD file....21, 22
CDD Manager....16, 23
  install....16
changes....21
  metadata....21
chart....2
  CONNX Client....2
CICS....30, 34
CISAM....2
CISAM on AIX....39
client/server....2
COBOL application....41
Component Configuration....9
Component Installation....15
  Planning....15
Component Requirements....12
  Estimating....12
connect....34, 36, 41
  ADABAS....41
  Oracle....34, 36
Connection Pooling....28
CONNX....1, 2, 6, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18, 19, 21, 22, 24, 25, 26, 28, 29, 30, 32, 34, 36, 37, 38, 41, 42
  access....9, 14, 22
  ADABAS....41, 42
  deploying....8
  Having....19
  including....10
  install....12, 15
  JDBC....38
  part....17
  planning....9
  set....19
CONNX API....15
CONNX application....2, 10
  depending....10
CONNX CDD....2, 6, 16, 22
  called....16
CONNX CDD Manager....16

CONNX Client 2, 6, 9, 12, 15, 16, 27, 30, 32, 38, 39
chart....2
copy....12
CONNX Client rollout....15
coordinate....15
CONNX Core Client....34
CONNX Data....22
make....22
CONNX Data Source....14
during....14
CONNX Enterprise Planning....1
CONNX Installation Guide....1, 22
CONNX Integrated Logon....14
CONNX JDBC....39
CONNX JDBC Client....38
CONNX JDBC/RCI....39
CONNX License....2, 6, 9, 30, 33
run....6
CONNX License file....22
CONNX Listener....2, 13, 27
contains....2
CONNX Planning....8
CONNX RCI....41, 42
Embedded SQL....41
CONNX Remote....2, 9, 13, 15, 26, 28
install....15
CONNX SERVER....2
CONNX Servers....27
CONNX SQL....17
CONNX User Guide....1, 14, 22, 25, 28
refer....22, 25
contains....2, 16, 22
CDD....22
CONNX Listener....2
metadata....16
Controlling....22
Access....22
coordinate....15
CONNX Client rollout....15
copy....12
CONNX Client....12
CPU....17, 26
Create Table Cluster Description....21
Create Table Description....21
D
Data Conversion....2
Data Dictionary....22
Data Dictionary Files....22
Data Dictionary Manager....22
Access....22
Data Joining....2
DataFlex....2
dataset....12
datasets....12
DB2....2, 6
DB2 on AIX....40
DB2 on AS400....37
DB2 on MVS....6, 38
DBMS....2, 26
DCL....20
DDL....21
DDL SQL....21
depending....10
CONNX application....10
deploying....8
CONNX....8
Deployment....20
Development....19
development/test/deploy/administrate....18
Different Windows....20
Disk I/O....28
Distributed Data Joining....2
DLL....2, 9
DLL/library....2
during....14
CONNX Data Source....14
E
Embedded....2
Embedded SQL....41
CONNX RCI....41
Enterprise Planning....6
enterprise-wide....18
Entire-Network....13

Estimating....10, 12, 13, 14
Component Requirements....12
License Requirements....14
Server Requirements....13
Throughput Requirements....10
Estimating Throughput Requirements....10
General Guidelines....10
Ethernet....28
exceed....27
RAM....27
F
Feet....2
Figure....2, 9
refer....9
File Locking....2
find....19
ODBC....19
firewall....9
G
General Administration Concepts....18
General CDD Management Concepts....21
General Guidelines....10
Estimating Throughput Requirements....10
General Performance Concepts....24
General Planning Concepts....8
General Scenario Information....29
guarantee....30
uptime....30
H
Having....19
CONNX....19
Hibernate on Linux....38
HIERACHICAL....2
HPUX-Itanium....2
HPUX-RISC....2
I
IBM WebSphere....39
IIS ASP....36
shows....36
IMS on MVS....40
including....10, 38
CONNX....10
VMS....38
Index metadata....23
Indexes....23
Updating....23
Informix....2
Input....28
install....12, 15, 16
CDD Manager....16
CONNX....12, 15
CONNX Remote....15
Interoperability....1
Introduction....1
ISAM....2, 13
ISAM/hierarchical....2
J
Java....34, 38
shows....34
Java Service Container....38
Java Servlet....39
Java Web Container....39
JBOSS....38
JBOSS application....38
shows....38
JDBC....1, 2, 15, 19, 21, 38, 40
CONNX....38
JDBC Server....21
JDBC Servlet....39
JDBC servlet application....39
JDBC Web Services....40
JDBC/RCI....2, 15, 28, 38, 39, 40, 41, 42
JDBC/RCI middleware....38
JDBC/RCI Server....38
Journaling....2
K
keep....16
CDDs....16
L
LAN....9, 28, 33
access....9
License....6
License Requirements....14
Estimating....14

Linux....2, 36, 38
Linux Z-Frame....2
loopback....28, 38
LPAR....41
M
make....22
  CONNX Data....22
Memory....27
metadata....2, 16, 19, 21, 23
  changes....21
  contains....16
  type....21
meta-data....16
Microfocus on HPUX....36
mission-critical....1
move....32
  CDD....32
multi-processors....26
MVS....6
MySQL....2
N
NET....1, 2, 19, 21, 30, 32, 36
NET application....30, 32, 36
  shows....32
NET Data Provider....6
Network File System....26
  use....26
Network I/O....28
NTIER....14, 37
  purchase....14
NTIER application....37
n-tiered....1
O
ODBC....2, 19, 21, 33
  find....19
ODBC Application....33
ODBC Web Service....37
ODBC Web Service Application....37
ODBC/OLE DB standards-based....2
offload....26
offloads....12
OLE DB....2
OleDB....2, 19, 21, 30
online....21
OpenVMS....26
  running....26
Oracle....2, 34, 36
  connect....34, 36
Oracle DBMS....2
Oracle Heterogeneous Services....34
Oracle JDBC....34
  uses....34
Oracle JDBC Client....34
Oracle on Solaris....33, 34
Oracle on Windows....30
ORDER BY....25
  use....25
Output....28
P
parent/child....16
part....17
  CONNX....17
PC....26
Physical Data Access....2
Planning....9, 15, 16
  Component Installation....15
  CONNX....9
  SQL Catalog....16
precompiled....41, 42
  RCI....41, 42
purchase....14
  NTIER....14
Q
Query Plan....2
Query Tuning....25
R
RAM....27
  exceed....27
Random Access Memory....27
RCI....2, 41, 42
  precompiled....41, 42
RCI Embedded....42
RCI Embedded COBOL....41
RCI Embedded COBOL application on Z/OS....41

RDB 2, 26
read/write 6, 22
Real-time, read/write 2
Record Locking 2
refer 9, 22, 25
  CONNX User Guide 22, 25
  Figure 9
Refresh Index 23
Refresh indexes 23
RELATIONAL 2
RMS 2, 26
RMS on VMS 32, 33
rolloutfirst 15
running 6, 26, 41
  CONNX License 6
  OpenVMS 26
  Z/OS 41
S
Scalability 17
Scalability Concerns 17
Scenario 30, 32, 33, 34, 36, 37, 38, 39, 40
Scenario 10 41
Scenario 11 42
Server Components 2
Server Requirements 13
  Estimating 13
set 19
  CONNX 19
shows 32, 34, 36, 38
  IIS ASP 36
  JAVA 34
  JBOSS application 38
  NET application 32
Simplified Data Access 2
SOA 12, 14
SOA application 2
Solaris 2, 34, 40, 42
SQL 2, 10, 17, 21, 22, 25, 28, 41, 42
SQL Catalog 16
  Planning 16
SQL Server 6, 32, 34, 42
SQL Server on Windows 42
SQL Server Provider 32
sqlregistry 37
SQLServer 2, 32
SQLServer on Windows 6, 32, 39
standards-based 1
Store 34
  CDD 34
Sybase 2
T
testing 19, 20
  UNIX 20
These CONNX 10
Throughput Requirements 10
  Estimating 10
today's 1, 26
Tomcat 40
Tomcat on Solaris 40
tool/application 2
Transaction 2
Treat 19
  CDD 19
Type 21, 38
  metadata 21
Typical Architecture Example 6
Typical Scenarios 6
Typically CDD 22
U
UNIX 2, 20, 37
  testing 20
Unix Client 33
UNIXODBC 33, 37
UNIXODBC on Linux 33
Update Statistics 23
Updating 23
  Indexes 23
uptime 6, 30
  guarantee 30
user's 32
uses 21, 25, 26, 34
  CDD 21
  Network File System 26
  Oracle JDBC 34

ORDER BY 25
utilizing 41
3GL 41
V
VAX 26
VB 30
View 2, 16
CDD 16
VM LPAR 41
VMS 20, 38
including 38
VSAM 2, 34
accessing 34
VSAM on MVS 37
VSAM on Z/OS 30
VSE 34
W
Web 12
Web Container 40
webforms 36
Websphere 39
Websphere on HPUX 39
Windows 2, 6, 33, 34, 39, 42
Z
Z/OS 41
running 41
Z-Linux 41